TUGboat

Volume 41, Number 1 / 2020

TeX Users Group

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

Individual memberships

2020 dues for individual members are as follows:

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

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

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

Journal subscriptions

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

Institutional memberships

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

Trademarks

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

[printing date: April 2020]

Printed in U.S.A.

Addresses

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

Telephone
+1 503 223-9994

Fax
+1 815 301-3568

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

Electronic Mail

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

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

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

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

Stay tuned for The TEX Tuneup of 2021!

Donald Knuth, “The TEX tuneup of 2014”, *TUGboat* 35:1 (2014)

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP

Editor Barbara Beeton

Volume 41, Number 1, 2020

Portland, Oregon, U.S.A.

TUGboat editorial information
This regular issue (Vol. 41, No. 1) is the first issue of the 2020 volume year. The deadline for the second issue in Vol. 41 is August 3, 2020.

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

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

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

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

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

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

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

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

TUG Institutional Members

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

From the president

Boris Veytsman

I am writing this letter in March 2020 amid the preparation for the coming coronavirus outbreak. Here in the San Francisco Bay Area schools are closed, theaters, museums and libraries are shuttered, sporting events and conferences are canceled. Some people, including me, are privileged to be able to work from home. Unfortunately many are not. We are recommended to practice social distancing: to minimize our gatherings and avoid others.

A friend of mine chose this moment to re-read the *Decameron* by Boccaccio; a very appropriate reading. Let me remind you of the plot of the book: in the wake of plague epidemics in Florence, seven young women and three young men practice social distancing. They escape to the countryside and entertain themselves with stories for ten days of the two weeks they quarantine themselves. Ten narrators times ten days produce one hundred stories, a veritable feast.

Written in the middle of the 14th century, the *Decameron* was initially distributed as handwritten manuscripts. Rhiannon Daniels in her *Boccaccio and the Book Production and Reading in Italy 1340–1520* (Legenda, 2009) mentions 14 extant manuscripts dated to the 14th century, 46 manuscripts dated to the 15th century and seven manuscripts dated to the 16th century, when printed editions became more popular. The wide readership of the *Decameron* led to democratic editions: most of the extant manuscripts were done on paper rather than parchment. The first printed edition of the *Decameron*, according to Daniels, is dated to 1470, just a decade and a half after Gutenberg's Bible. Afterwards the book was reprinted many times during the 15th and 16th centuries. Unlike Gutenberg's Bible, the early editions of the *Decameron* were typeset in Roman type, as befits a humanist book. Daniels writes about the tendency of early printers to make the book widely available by lowering the price: the move from the single column design to the two column one to decrease the page count, the introduction of woodcut illustration instead of manual ones, etc. This tendency mirrors the tendency of the handwritten editions to democratize the book, making it accessible to a wide readership.

An important feature of the *Decameron* is its beauty. Each story is interesting in itself (many later writers including Chaucer borrowed from them), but their subtle interplay with each other and the personalities of their narrators are superb.

Of course, not many Florence inhabitants had the means to practice social distancing in the way Boccaccio's protagonists did. John Henderson's *Florence Under Siege: Surviving Plague in an Early Modern City* (Yale, 2019), recently reviewed by Erin Maglaque in the *London Review of Books* (**42**:4, 2020), is a good companion to the *Decameron.* While the science of the 13th century did not provide much knowledge about the plague (the role of fleas was discovered much later), the steps taken by the city health board *Sanitá* are impressive and very rational. By cordoning the city, *Sanitá* got time to prepare for the inevitable outbreak. The board studied the response of other cities hit before Florence and learned their lessons. It quarantined the families of sick and dead in their homes. The churches were closed. Instead, portable altars were erected on street corners: priests conducted Mass from there, and the people said *Amen* from behind the doors. Confessions were also taken through doors or windows, with priests covering their mouths and noses with waxed cloth. While the medieval medicine (theriac, ground pearls, crushed scorpions, etc.) was probably not very effective, another idea of *Sanitá* likely was. Assuming that poor nutrition might provoke the disease, the health board spent enormous sums of money (partially from the draconian fines for quarantine violations) to feed the quarantined people. While some rich Florentines complained that many city's poor never ate as well as during the plague, this measure doubtlessly helped to reduce the number of violations and to increase the immunity of well fed people. The death rate in Florence was 12% of the population; for comparison for Venice it was 33%, in Milan 46%, in Verona 61%.

I am impressed by three features of these stories. First, the beauty, which persisted in the plague infested years. Second, the science and rationality of the response. Third, the solidarity: we overcome infection when we understand that we all are in the same boat. I think they deeply resonate with us, TEX people. TEX was born from the striving for beauty and rationality. Following Knuth, we use rational methods to create beautiful pages in service of presentation of beautiful thoughts. Our software is free and available to all, which stresses our solidarity as humans. In our small community we try to embody the ideals that helped us to overcome the travails of the past.

And a last word. When I read papers on COVID, I habitually check how they are typeset. When I see TEX I feel a pride that somehow our efforts contributed to the common task.

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

Editorial comments

Barbara Beeton

Our plague year

As I write this, all activity and commerce are shut down over much of the world. Schools are closed, students have been sent home, and I've just eaten my last restaurant meal until who knows when. More relevant to this audience, the DANTE and BachoTEX meetings have been cancelled, to the great disappointment of many eager participants. We shall have to wait, to find out what the next few months will bring, before we'll know whether this catastrophe will affect TUG 2020 in Rochester as well.

Elsewhere in these pages are some items that deserve mention.

- The announcement of the Grapholinguistics conference, scheduled for June, will be held electronically rather than in Paris. Watch the linked website for further announcements.
- The general shutdown has extended to my local libraries. Since the published hyphenation authority exists only on paper (Webster's *Third International Dictionary*), and I don't own a copy, it hasn't been possible to verify some of the proposed additions to the hyphenation exception list. The current addendum is less complete than intended, but this is somewhat balanced by the update of the cumulative version on CTAN. When resources are once again available, both an additional update and cumulative list will be released.
- The history of the differential "d", promised in the last issue, also requires library access. Like the hyphens, completion of this item depends on the return of normal access.

Not triggered by the virus but nonetheless important for their own sakes, and in the first case, timely, these articles deserve attention.

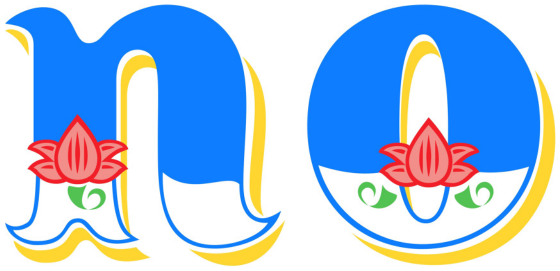
- The next review by Don Knuth of the TEX/METAFONT complex is scheduled for 2021. To help prepare, David Fuchs invites (see his article in this issue, pp. 8–11) users of plain TEX and METAFONT to search their archives for documents making use of unusual TEXniques that can in turn be used to exercise nooks and crannies of the Knuthian engines that might have slipped through the cracks explored by the `trip` and `trap` tests.
- This issue's "DuckBoat" has a new subtitle, "Beginner's Pond", and a comment that the "articles may appear too trivial for a serious journal like the *TUGboat*." I call your attention to this item in the *TUGboat* Editor's wish list:[1] "More tutorials and expository material, particularly for new users and users who aren't intending to become TeX wizards." So, Prof. van Duck, don't apologize!

Updike Award for Student Type Design

For several years now, a challenge has been made to typographic design students to develop a typeface influenced by materials in the Updike Collection on the History of Printing, one of the Special Collections at the Providence Public Library. The most recent award ceremony, the fifth, was held on 25 October 2019, accompanied by a panel discussion, "Revival Meeting", featuring type designers who have created digital types based on historical types from the earliest days of European printing up to the demise of the general use of metal type.

Awards were presented to three students:

- Malika Nanda (first place), for *Mojaris*, a decorative face inspired by models she encountered in her native Bombay.



One inspiration was a slipper with an upturned pointed toe. The influence can be seen in the shape of the curvy serifs.

- Benjamin Tuttle, for *No Gothic*, a bold sans serif face, with alternate shapes for the capital "G" and both the upper- and lowercase "S".

[1] `https://tug.org/tugboat/wish.html`

- Liam Spradlin, for *Girassol*, a sans serif display face, inspired by street signs seen in Portuguese cities.

Presentation of the awards was followed by questions to the three finalists, asking why they chose the particular styles, and what inspired them to undertake font design. Two of the responses cited an attraction to graphical objects observed in the student's surroundings; the third noted the desire to have a unique and idiosyncratic medium to be used in personal communication. These goals are met admirably.

Of the four participants in the panel discussion, three are active designers: Paul Barnes (Commercial Type), Marie Otsuka (Occupant Fonts), and David Jonathan Ross (Font of the Month Club). The fourth, Elizabeth Carey Smith (Creative Director at the Bank of New York) acted as moderator.

One discussion topic was the reason for revivals. From an educational point of view, it teaches an understanding of the structure of a font, and experience in how to get a feeling for what looks correct. Besides, it's hard work.

Another reason is that a revival might be commissioned. When undertaking a commission, an important aspect of the work is to determine what it is that the person making the commission really likes about the font. Often this is the look of the printed specimen, in which case strict adherence to the actual type matrix is probably not advisable.

Questions from the audience included a request for recommendations for making a successful career of font design. A logical response was to develop families rather than single fonts; when more options are available, if a client likes one after using it, there are more that can be acquired.

The next award will be presented in 2021, and will thereafter be scheduled every two years. Details, including the guidelines for entering the competition, are given on the library's website.[2]

Questions for Paul Barnes: *Nature*, revivals

After the presentation, I got a chance to talk briefly with Paul Barnes. His firm has designed the new type for the journal *Nature*,[3] and he observed that all the symbols drove them crazy. (Christian Schwartz, Barnes' collaborator in the foundry, was responsible for the symbols.) The font, like most custom designs, is under exclusive license to the commissioning organization, through 2024.

I asked whether *Nature* accepted LaTeX as an input medium. Yes, but the production flow goes through several conversions. The final archival files are held in a standardized form of XML, and for the print version, the material is flowed into InDesign.

On the subject of revivals, an online video of an interesting lecture Barnes gave last summer at Cooper Union[4] further illuminates the topic of the evening's discussion.

Resources — finding things

Identifying a font Earlier this year, I was sent on a quest by Mike Spivak to determine the fonts used to typeset one of his books on differential equations. The results of this search were equivocal; I was able to identify the main text and math fonts (Monotype Baskerville, with digits from some version of Times used in math and text, but curiously the Baskerville digits used for page numbers), but failed to identify the large digits used for the chapter numbers. (I later learned that it may have been a custom creation.) However, I *did* learn about some resources for identifying fonts.

- WhatTheFont![5] will search a database to match the image of a character presented on a smart phone or other online device. (It works well for letters and digits; I recognized the symbols as what was available for Monotype at that period, so didn't test any, but will have to try that sometime.)

- Identifont[6] presents several questions that narrow down the possibilities and then shows potential matches to be examined. Unfortunately, "italic" is considered to be script or calligraphic, so this was unsatisfactory for the topic of my quest, where matching the italic "N" was essential to the identification.

- `fonts.com`[7] has a very extensive collection of fonts, although when I rechecked, no "identifier" tools. However, if the name of the basic roman style is known or suspected, the corresponding bold and italic are likely to be there for checking; a sample text can be input to match against the subject example.

[2] `https://www.provlib.org/research-collections/special-collections/updike/updike-prize-student-type-design/`

[3] `https://www.nature.com/articles/d41586-019-03083-5`

[4] `https://coopertype.org/event/the_past_is_the_present`

[5] `https://www.myfonts.com/WhatTheFont/`

[6] `http://www.identifont.com/`

[7] `https://www.fonts.com/`

(LA)TEX Q & A — TopTeX A new Question and Answer website has appeared.[8] Describing itself as "a friendly community for TeX questions and answers", it is part of the open source and not-for-profit platform `topanswers.xyz`.

The scope is similar to `tex.stackexchange.com`, but there are differences. As an "independent" site, its direction depends more on the needs and wishes of its participants than on any corporate policy. One feature mentioned for possible future adoption is that general, encyclopedic, articles will be welcome; these are explicitly discouraged on StackExchange. This is still a work in progress, one that will be controlled by the members.

When the site is opened in a browser, the immediate reaction is one of surprise at the bright color. (My view might be influenced by a very old browser on a laptop with an out-of-support operating system; new hardware is on order.) The screen real estate is split in two, with a wider panel on the left; this is devoted to the list of questions. The right side is a narrower column holding the main chat. To access an item on either column it is necessary to scroll, using either a mouse or touch pad; I haven't yet found a way to scroll from the keyboard, but I haven't asked either.

As with any new site, navigation takes a bit of getting used to, but nothing is too deeply hidden. One feature that seems to be absent is a profile area where a participant can share some personal background; it can take a newcomer a while to determine the strengths and interests of other participants. Most participants identify themselves by nicknames, but this isn't any different from other forums or discussion lists. I've learned that enhancing the profile features is on the to-do list, but other improvements have higher priority.

An attractive enhancement, appearing just before this *TUGboat* issue is sent to the printer, is the addition of a non-English "sister community". The language is Marathi, and it will be presented in the Devanagari script. (The announcement illustrates the "blog" category of posted items.)

This is still a work in progress, and will grow in directions still unknown, but responsive to its participants. It will be refreshing to not be surprised by the kinds of changes that have appeared with little warning on the StackExchange complex. One inquiry I made, about a feature that I thought was suboptimal, was quickly accepted as a bug by a member of the support group. Give it a try.

[8] `https://topanswers.xyz/tex`

Thoughts on asking questions in a public forum

Anyone using TEX is going to have questions. Long-time users have often developed their own personal networks, but even so, it is sometimes helpful to reach out to a public forum. There are several such forums available to TEX users — managed online Q & A sites like StackExchange and the new TopTeX, older mailing lists like `texhax`, and public "notice boards" like `comp.text.tex`. Each of these has its own advantages and audience. But some things are common, if one wants to get a quick, accurate, and useful answer.

- A small compilable file that gives the result or illustrates the question as described. This was the primary topic of the first DuckBoat column.[9]
- The context in which the answer is needed. For example, is it to be submitted for publication in a particular journal, is it for a thesis, is it for some personal use, something else?
- What is the subject area: (pure) mathematics, physics (theoretical or experimental), linguistics, ... The notations and conventions may be different for seemingly similar circumstances.
- What is your deadline? Please don't leave this until the last minute! Plan ahead!

The goal of a question is to get an accurate, useful answer. The more precise the question, the more likely it is to attract potential helpers.

Even if a problem is interesting, and someone looking at it is familiar with the area and thinks they know the answer, it's a real drag to have to guess the document class, only to be told later that it isn't the one being used. After being contradicted in this manner, a helper will be less likely to react favorably the next time.

Value the time that the helper will invest in answering your question. That will increase the likelihood that your next question will be met with respect.

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

[9] `https://tug.org/TUGboat/tb38-3/tb120duck.pdf`

Reporting bugs for Don Knuth (as soon as possible)

Karl Berry

Abstract

Please report TEX system bugs intended for Don Knuth as soon as possible.

1 The next TEX tuneup

At the end of the last TEX tuneup report [3], Don Knuth mentioned that the next tuneup will be released in 2021. He asks for the collected submissions towards the end of the previous year [1], that is, 2020, that is, this year! I am the one collecting bugs this time, having been passed the TEX entomologist's microscope from the inestimable Barbara Beeton. I do not have an exact cut-off date, but it would be reasonable for you to assume that anything submitted after November 1 will not be warmly welcomed.

To reduce the number of reports at the deadline, I ask that they be submitted as soon as possible. Handling them takes time, since colleagues and I review each and every one before putting it in Don's pile. There are two ways to submit reports:

1. Send email to the public mailing list, `tex-k@tug.org` (`https://lists.tug.org/tex-k`). This is preferred, since then many people, not just me, can review the report (and often reply faster).
2. Send email to me, `karl@freefriends.org`, if you want your report (name, email, contents) to be private. It is fine to do this if you prefer; much better than not reporting the bug at all.

2 Research before sending

Please do a general web search on your report before you send it in, choosing the relevant search terms, etc., to the best of your ability.

If the web doesn't have anything, please check Don's detailed errata listings, available at `https://ctan.org/pkg/knuth-errata` or, in TEX Live, in `texmf-dist/source/generic/knuth/errata`. Just try searching (e.g., `grep`) for keywords; it may not be easy to make your way through the errata files, but do the best you can.

In all cases, what's of interest is true bugs in the code, typos or other outright errors in the text, etc. Requests for enhancements, new features, etc., are not going to be reviewed by Don. However, if a report is in the gray area between bug and feature, please go ahead and send it, and we will review it.

Some enhancements, including but not limited to increasing array sizes and other configuration issues, can be (and have been) made in TEX Live and other implementations. It's fine to also send such requests to `tex-k@tug.org` or my address.

3 Software for which to send reports

Of course the most important programs for which I'm collecting bug reports are TEX and Metafont themselves: The contents of Volumes A–E of *Computers & Typesetting*. (Volume A is equivalent to *The TEXbook* in softcover; Volume C is equivalent to *The METAFONTbook* in softcover. Volume B contains the source code for TEX and the prose that explains each part; Volume D contains the analogous code and prose for METAFONT. Volume E contains complete METAFONT code for the original Computer Modern fonts.)

The other `WEB` programs still maintained by Don are also fair game. The complete list:

```
dvitype gftodvi gftopk gftype mf mft pltotf
pooltype tangle tex tftopl vftovp vptovf weave
```

The main exception is BIBTEX; reports for that should be sent to the public list `biblio@tug.org` (`https://lists.tug.org/biblio`); Oren Patashnik, the BIBTEX author, will see them there. Or you can send them to me. (No particular deadlines or tuneup release dates for BIBTEX.) Reports for other Stanford `WEB` programs can also be sent to `tex-k` or me.

4 CWEB(bin)

Don also still maintains the original CWEB programs written by him and Silvio Levy, and will accept bug reports for them, at the usual low priority; *TAOCP* must take precedence.

With Don's agreement, in 2019 we changed the CWEB distributed in TEX Live to come from the `cwebbin` package, developed by Andreas Scherer at `https://github.com/ascherer/cwebbin`. It is compatible with the original CWEB, and supports more C and C++ language variants, has enhanced internationalization, etc. Reports for `cwebbin` should go to `tex-k@tug.org` or the `github` project above.

Thanks, and let the bug reports flow.

References

[1] D. Knuth. Computers & Typesetting. `https://www-cs-faculty.stanford.edu/~knuth/abcde.html`

[2] D. Knuth. The TEX tuneup of 2008. *TUGboat* 29(2):233–238, 2008. `https://tug.org/TUGboat/tb29-2/tb92knut.pdf`

[3] D. Knuth. The TEX tuneup of 2014. *TUGboat* 35(1):5–8, 2014. `https://tug.org/TUGboat/tb35-1/tb109knut.pdf`

◇ Karl Berry
`karl (at) freefriends dot org`
`https://tug.org/texmfbug`

Beyond Trip and Trap: Testing the urtext WEB sources

David Fuchs

Abstract

Finding some undefined behavior and other tricky bugs in Donald Knuth's original WEB sources of TeX, METAFONT, &c, and a request for plain TeX and METAFONT input files.

1 Preparing for the next TeX tuneup

In anticipation of TeX and METAFONT approaching versions π and e, respectively, I've been working on some tools to help find any remaining "undefined behavior" bugs in the core code, so they can be addressed for posterity.

In the remainder of this note, I'll mostly refer to TeX for brevity, but everything said here applies to the set of principal Stanford WEB programs: Tangle, Weave, TeX, METAFONT, TFtoPL, PLtoTF, DVItype, and GFtype.

2 Range checks

As usual with this sort of tool, the first kind of "undefined behavior" to watch for at runtime is any attempt to read from an uninitialized variable. (One caveat, however: TeX and METAFONT do this intentionally at one place, with the *ready_already* variable.) This is straightforward to handle using an auxiliary bit per variable that indicates whether it has been written to yet. Each field in a record (C `struct`) gets its own bit, as they can be assigned to individually.

Next, each array has a declared range of valid index values, so each access to an array should check that the index is properly in range. Also, each read of an array element should check that that individual element has been initialized, as per the first item. C programmers may be unaware that Pascal allows each individual array to be non-zero-based (e.g., *weight*: **array** [1957..2020] **of** *pounds*), but this doesn't add any significant complication.

Additionally, Pascal is somewhat unusual in that it allows the programmer to specify that a scalar variable will store only a certain range of values. C has somewhat similar functionality with "bitfields" in records, where each integer field can be specified as taking a given number of bits, but Pascal completely generalizes this to ranges, exactly like array subscripts; for instance, *year_of_interest*: 1957..2020 declares a variable that may contain values only in the given range. As TeX and METAFONT make liberal use of this feature, it's also worth checking at runtime that each assignment to such a variable is within its declared range of valid values.

Similarly, if a procedure has a formal parameter with a limited range, at runtime each actual parameter of each call should be checked for validity. Checking that returned values from functions also match their signatures rounds out the list of range checks. It's also worth checking for overflow on each addition, subtraction, and multiplication operation (leaving the hardware to watch for division by zero).

3 Toolchain background and an edge case: empty change files

To do this checking, I wrote a purpose-built transpiler that inputs Knuthian Pascal and outputs very vanilla C code that implements all these runtime checks. The transpiler is about 7,600 lines of C and the runtime library about 2,500 lines.

TeX and METAFONT come with their own test input files, trip.tex and trap.mf, with instructions on how to use each to verify ports. These test files are designed to execute every line of code (with minor exceptions such as fatal error messages), and thus provide one good way to try out the transpiler output. Of course, generating the code involves running Tangle on all of the above WEB files, which tests it; and doing the complete trip and trap test regimens runs the rest of the programs; except for Weave, which I tested separately on all the above WEB files as well. Finally, I also ran a number of large documents that require only original (traditional? ur? un-enhanced? Knuthian?) TeX.

All of the Pascal files are generated directly from Tangle using empty change files, so everything is tested exactly as in DEK's master sources, unmodified, with two exceptions: First, *ready_already*, as mentioned above, gets specially tagged as being "unmemchecked", so it won't trigger a failure when it's read before being written. Second, the line of "dirty Pascal" code in the `tex.web` module ⟨Display the value of *glue_set*(p)⟩ that tries to detect invalid floating-point values that a bug may have caused to be stored in a floating point field, has been removed, as this attempt itself would be detected as trying to access the wrong variant (discussed below). Of course, as this later module is indexed under "system dependencies" and "dirty Pascal" in DEK's sources, this is an expected spot for a platform-specific change.

And one bug was found, manifested in both Tangle and Weave: they share a bunch of code that deals with text file reading, which is where the bug appeared. The module ⟨Read from *web_file* and maybe turn on *changing*⟩ contains this code:

```
...
else if $limit = change\_limit$ then
  if $buffer[0] = change\_buffer[0]$ then
    if $change\_limit > 0$ then $check\_change$;
...
```

to see if the current line from the `WEB` file matches the first line after an `@x` in the change file. First it tests that the line lengths match, then that the first characters match, and only then does it go to all the expense of actually calling a function (*check_change*) that sees if the lines fully match. (Computers used to be slow, so one could argue that this was a reasonable optimization, rather than paying the price of just calling *check_change* every time.) The bug occurs when the change file is completely empty, in which case *change_buffer*[0] is uninitialized when this code tries to read it. It's interesting to note that I would not have bumped into this bug if not for the fact that the transpiler handles DEK's code directly, and thus most of my change files are in fact empty.

However, rather to my dismay, in normal, non-undefined-behavior-checking-mode, no matter what junk might happen to be in *change_buffer*[0], Tangle and Weave still operate properly (since *change_limit* will be zero in this case, so *check_change* won't be called anyway). So no one will ever be affected by this bug, leaving the philosophical question as to whether it's actually a bug or not. A redeeming aspect is that to fix this quasi-bug, one can simply remove the middle line of the code shown! (Since, happily, *lines_dont_match* ultimately checks the first character as appropriate anyway.) This may be the first bug I've ever encountered where simply removing code fixes it. In fact, the **if** $limit = change\_limit$ **then** check can be removed, too!

A few cases of fetching uninitialized variables were also detected in METAFONT. But in all these cases, the value fetched doesn't ever get looked at as METAFONT continues to execute. For the record: The procedure *recycle_value* starts with

if $t < dependent$ **then** $v := value(p)$

and then uses v in some of the subsequent cases in its switch statement. It turns out that in some of the cases where v is not used, $value(p)$ hadn't ever been assigned to. Note that the existing code tries to avoid the fetch when it's not going to use the value, but the condition $t < dependent$ isn't correct. The other cases are in *copy_path*, its mirror *htap_ypoc*, and *scan_expression*, all of which copy *right*/*left* x/y values, some of which may not have been previously written to (when the source record came from, say, *new_knot*). Again, these copied, uninitialized values are not subsequently used.

4 Variant records and homegrown memory management

Another potential case of "undefined behavior" concerns "variant records" (known as "unions" to C programmers). Every time a member field of a union is read from, we must check that the most recent write to that variable was to that same member. This is very important for TeX and METAFONT, as they make extensive use of variant records; see particularly the definition of *memory_word*.

TeX and METAFONT never allocate dynamic memory or deal with pointers, so we needn't worry about checking pointer dereference validity. But they do their own form of memory management within the big *mem* array; see *get_node* and *free_node* for how a linked-list of free blocks is maintained. Note that this whole scheme uses indexes into the *mem* array as "pointers", so they can be stored in 2 bytes. (Or, for larger capacity TeXs, 4 bytes, which used to make it harder to explain why "real" pointers weren't used, but recently many platforms have switched to using 8-byte pointers exclusively, so TeX's 4-byte pseudo-pointers are again a space-saver.)

Of course, this approach to memory management is not typical. In addition to pointers being smaller, TeX implements a zero-overhead allocation scheme: as compared with most implementations of *malloc* in C libraries, where each allocated item takes 8 or more extra bytes of storage beyond what the caller requested, in TeX there are no extra bytes per allocation. Recall that TeX was developed on a machine with a data address space of only about half a megabyte, so it's a tight squeeze to fit in an entire macro package, hyphenation rules, font metrics, etc., while leaving enough room to store a whole page's worth of boxes and glue, etc. Every byte counted, and it's fair to say that things are packed to the gills (this can also be seen in Weave, which makes two passes over the input file rather than try to fit everything into memory).

As mentioned above, the mechanism for catching uninitialized variables involves keeping an extra bit of information per variable to indicate if it's "readable" or not. It's fairly straightforward to add another extra bit to control variables' "writable" status. Then, by augmenting the code in *get_node* and *free_node* to make special calls that turn the "writable" bits on and off, respectively, for the entire node being allocated or freed, access-after-free errors get caught automatically. Additionally, we can arrange by using a huge *mem* array that no freed slot is ever reused later as part a subsequent allocation, thereby ensuring that there's no chance of an illicit read ever getting lucky and going undetected. This requires about 200 lines in TeX's change file, including more specific safeties, such as setting the static glue specs *zero_glue*, *fil_glue*, . . . , *fil_neg_glue* to be not writable (and all tests of TeX were run with and without this change, just to be sure that no bugs get hidden by it; ditto for subsequent changes mentioned herein).

The entire suite of programs pass all these checks, as one might expect, given their robustness in the field. Also, if I recall correctly, in the 1980s there was an especially good Pascal compiler from DEC for their VAX/VMS systems that was able to detect these sorts of errors, and someone in the TeX user community reported a few bugs of this sort that it found.

But there's yet another, more subtle, type of problem left to consider. The issue with keeping track of which member of a union is "active" has already been mentioned. But TeX goes a step further, and re-uses members for different purposes in different contexts. For instance, the *in_state_record* is how TeX keeps track of the current line of input from a file, with member fields that tell where the line begins and ends, and where the next character to read is within those limits. But when the current input is instead from a macro, these same fields get used to now keep track of the start of the macro's token list, and where along the way the next token is to be fetched from, etc. Some of the fields are given new names with a simple `WEB` macro that redirects to the old name; other fields just get reused with the same name (such as *start*, which is a fine name to indicate either the start of a token list or start of a line, even though in one case it's a pointer into *mem*, and in the other an index into the input buffer of characters). But either way, how do we make sure that a value stored with one meaning isn't attempted to be interpreted with the other meaning?

The answer is to manually introduce new member fields in separate variants to distinguish the two contexts, thus reducing the problem to one that's already been addressed. This takes some manual labor to examine every use of each symbol, and assign it to one of the two variants, but it's not too onerous, resulting in fewer than 200 lines in the change file.

Quite a bit more extreme are the *memory_words* in *mem*; they have more than three dozen different possible interpretations: *height*, *width*, *depth*, *glue_stretch*, *glue_shrink*, *penalty*, etc., etc. The interesting twist here is that multiple node types share various fields under a common name and offset; both boxes and rules have *width*, *height*, and *depth*, but only a box has a *shift_amount*; and a kern also has a *width*, but no *height* or *depth*. The trick here is to put each of these field types into its own variant in *memory_word*, so they get checked individually. So the *width* of a box is the same sort of thing as the *width* of a rule, but different than the other sorts of things that other node types have at offset 1.

While separating out all the different uses of *memory_word*, we get an additional opportunity for checking ranges. For example, consider TeX's "delimiter fields", which hold family and character values. These normally get stored in byte-sized fields in a *memory_word* record, but the *fam* is always supposed to be in the range 0..15. So, the newly introduced variant can actually specify that its *fam* field is of this range type, with the result that all the checking logic will get kicked off by the transpiler. TeX has other similar cases, including some where the permissible values are more like an enumeration, and are in fact turned into one; for instance the *stretch_order* and *shrink_order* fields of a glue specification take values of the enumerated type *glue_ord*.

Quite a bit of manual effort was involved with this set of alterations, requiring over 1100 lines in TeX's change file, but with satisfactory results.

5 An edgier case: unbalanced braces at end of file

And, again, all this additional checking finds a bug, this time in TeX. It's in the use of the input state (discussed above). If the module ⟨Input the next line of *read_file*[m] ⟩ encounters an end-of-file at a time when braces haven't been balanced, a call to the *error* reporting routine is made. But this happens before the formalities of setting up the input state to properly represent the input line have happened. So, an uninitialized field of the *in_state_record* gets read, leading to the failure.

In fact, the Trip test hits this very situation (not surprisingly, since it attempts to hit every line of code in TeX, including each error message). It was never noticed because of a happy coincidence in which the junk values happen to produce a reasonable result, and TeX continues uninjured. Results could be more disastrous with non-Trip inputs. The same bug can occur with the fatal error "`*** (cannot \read from terminal in nonstop modes)`" occurs, but that's even more of an edge case, and isn't tested for (as all fatal errors are not).

☡☡ In particular, ⟨Input and store tokens from the next line of the file⟩ hasn't yet fallen through to the code that sets *loc* and *limit* when the *error* call happens. When *error* does ⟨Pseudoprint the line⟩, we're in trouble, since it thinks that *loc* and *limit* tell where the contents of the problematic line are.

☡☡ When a non-checking TeX gets to line 415 of trip.tex, where the "File ended within `\read`" case is tested, it kind of lucks out: *limit* happens to be a left-over zero (from a left-over *param_start*, no doubt), while *loc* is a big number (similarly representing a left-over token location), and *start* is actually a correct pointer into *buffer*. So ⟨Pseudoprint the line⟩ randomly checks *buffer*[0] for *end_line_char*, then in any case skips its for loop, and happily shows the two empty context lines (lines 6167–68 in `trip.log`):

```
l.6167 <read 0>
l.6168
```

To see this bug in action, we can create a file `unbal.tex` containing a single "`{`" character, and create a second file `readbug.tex` containing:

```
\catcode`{=1 \catcode`}=2 \catcode`#=6
\openin1 unbal
\def\A#1#2#3#4#5#6#7#8#9{\read1to \x}
\def\B#1#2#3#4#5#6#7#8#9{\A#1#2#3#4#5#6#7#8#9 \relax}
\def\C#1#2#3#4#5#6#7#8#9{\B#1#2#3#4#5#6#7#8#9 \relax}
\def\D#1#2#3#4#5#6#7#8#9{\C#1#2#3#4#5#6#7#8#9 \relax}
\def\E#1#2#3#4#5#6#7#8#9{\D#1#2#3#4#5#6#7#8#9 \relax}
\E123456789
```

Then, running `virtex readbug` results in a nonsense `<read 1>` context:

```
! File ended within \read.
<read 1> {^^M#5#6#7#8#9{\D#
```

The trick here is that all those parameters cause *param_start* to be 36, which then gets used as a bogus value for *limit* when *show_context* is called, resulting in unrelated stuff in *buffer* being shown as the context.

6 More stress: checking constants

Thus far, all of the testing has used as input only files which are part of the basic TEX distribution as it came from Stanford. So, Tangle and Weave get tested with all of the WEB sources; TEX and METAFONT have their Trip and Trap test files, but TEX also runs all the Weave output, as well as *The TEXbook* and *The METAFONTbook*, while METAFONT runs all of Computer Modern at various resolutions, and so on.

This represents a lot of stress, but doesn't hit everything. One additional direction I have tried pushing was to see that the ⟨Check the "constant" values for consistency⟩ checks were complete and accurate. Nothing of much importance showed up, other than there being no check that *buf_size* is at least big enough to hold the longest built-in primitive name (which happens to be a tie, at 21 characters each, between "abovedisplayshortskip" and "belowdisplayshortskip").

Finally, the only deeply-embedded constant in TEX that can't be changed at all (as far as I have noticed, anyway) is that the hyphenation routines only work on words of length up to 64. Or is it 63? Or is it only on the first 64 letters of a word? And does that mean only 63 possible hyphenation points, or would it include a possible hyphen after the 64th letter of a longer word? And where is this specified in *The TEXbook*? Trying to figure this out by reading the code is a challenge, as there are various 63's, 64's, and 65's scattered about (the latter having to do with adding sentinels to the word being hyphenated so that the pattern matching has something to match for beginning- and ending-of-word; and/or adding a byte that indicates the "current language" when looking up hyphenation exceptions).

So, I created a test file containing the four lines (abridged here):

```
\lefthyphenmin=0 \righthyphenmin=0
\hyphenation{-a-b-...-y-z-a-b-...-y-z-a-b-...-y-z}
\showhyphens{ab...yzab...yzab...yz}
\end
```

and tried it out. Sure enough, a bug occurs: the code tries to store a value that's not in the declared range of the receiving variable. In particular, in the *hyphenate* function, when ⟨Look for the word *hc*[1..*hn*] in the exception table, and **goto** *found* (with *hyf* containing the hyphens) if an entry is found⟩ is called, *hn* can already be 63, but then this module increments it to 64 for a while (to fit the *cur_lang* byte), which puts *hn* out of range for a *small_number*.

On common architectures, this bug probably won't actually change TEX's behavior, since *hn* will no doubt be stored in a full byte, which means it will be able to actually store the value 64 properly. By the way, I'd guess this bug got introduced when multiple-language support was added to the hyphenation code; the arrays grew by one to be able to append the language byte, but the declaration of *hn* got overlooked (easy to do, as it didn't show an explicit 0..63 range).

7 Need more `\input`

A big limitation in all this is the small number of plain TEX and METAFONT files I've been able to use as test input. The issue is that this is absolutely plain, original TEX and METAFONT, as created by DEK, without any of the added features of pdfTEX, MetaPost, etc. So, I'm on the lookout for TEX documents that use only macro packages that work on unmodified TEX. Any help in this regard would be appreciated, and I'm happy to share credit for finding any bugs that your devious macros or lengthy tome might turn up!

◇ David Fuchs
`plain-tex-tests (at) tug dot org`
`https://tug.org/texmfbug/`

Grapholinguistics, TEX, and a June 2020 conference

Yannis Haralambous

Abstract

This paper presents the conference *Grapholinguistics in the 21st Century* that was to take place in Paris, in June 2020. With the global health situation, it will now be held as a video conference (`https://grafematik2020.sciencesconf.org`). We give an introduction to the discipline of grapholinguistics, the history and the topics of the conference, and we close with the fundamental question: why should a TEX user join the conference?

1 What is Grapholinguistics?

Grapholinguistics is the discipline dealing with the study of the written modality of language.

At this point, the reader may ask some very pertinent questions: "Why have I never heard of grapholinguistics?" "If this is a subfield of linguistics, like psycholinguistics or sociolinguistics, why isn't it taught in Universities?" "And why libraries do not abound of books about it?" To answer these questions we have to go back to the period 1906–1911, when the Swiss linguist Ferdinand de Saussure was lecturing in room B105 of the University of Geneva. His lectures set the foundations of modern linguistics. They were published posthumously in 1916, as the notorious *Cours de linguistique générale* (translated as *Course in General Linguistics*, or *CLG* for the initiated [33]).

In his work, Saussure violently attacked writing:

> Language and writing are two distinct systems of signs; the second exists for the sole purpose of representing the first. The linguistic object is not both the written and the spoken forms of words; the spoken forms alone constitute the object. But the spoken word is so intimately bound to its written image that the latter manages to usurp the main role. [...] The preceding discussion boils down to this: writing obscures language; it is not a guise for language but a disguise. [33, p. 23–24, 30]

For him, language is oral, period. Writing is just an accidental secondary representation of language, one that betrays it and hides its true nature. His arguments were that (a) all human cultures have spoken languages, while only a small number of them write, (b) writing appeared much later than speech in human history. These are historical facts, but chronological precedence is less important than the undeniable fact that writing was the spark that ignited culture and technology as we currently experience it.

Saussure being the founder of modern linguistics, his ideas were followed by generations of allegiant linguists. In every current Linguistics textbook the two lowest-level subdisciplines of this science are Phonetics and Phonology. Phonetics studies all sounds humans can produce in order to communicate (these sounds are called *phones*), and phonology studies systems of distinct equivalence classes of sounds used by languages (these equivalence classes are called *phonemes*).

The next level after phonology is morphology, the study of minimal units of meaning (called *morphemes*), such as [table] and [s] in the word "tables" (the morpheme [s] being the suffix of plural number). No linguist ever bothered to ask "are spoken and written morphemes different?" This would be heretical behavior: according to Saussure, morphemes are built out of phonemes, the only "true" building blocks of language, and writing them on paper is only a convention, a necessary evil, a curse in our civilization that would be in better health if it stopped using writing in the first place (as in Bradbury's *Fahrenheit 451*).

Thus, linguists have inherited Saussure's disdain of the written word and this resulted in an ideology that French linguist Jacques Anis [2] calls *phonocentrism*. Phonocentrism argues, among other things, that the ideal writing system would be a phonetic one: /ðə mˈoːɹ ðə ɹˈɪtʔtnˌɹɛpɹɪzɛnteɪʃən ʌv lˈæŋgwɪdʒ ɪz klˈoʊs tə ðə spˈoʊkən wˈʌn ðə bˈɛɾɚ ænd mˈoːɹ ɪfˈɪʃənt ɪt ˈɪz/.[1] Once this argument is taken for granted, the next step is very naturally the simplification of writing systems: why bother with complex correspondences between graphemes (the elementary units of writing) and phonemes? Why not write a language as it is pronounced? For example, to pronounce Japanese you need only 32 phonemes (27 consonants and 5 vowels), why then learn tens of thousands of kanji characters?

The road to hell is paved with good intentions, and phonocentrism has caused a lot of misery, such as the 1982 monotonic reform in Greece, where accents and breathings were abandoned because, according to phonocentric dogma, they were inactive on the phonemic level [16]. In China and Japan there are regular initiatives to abandon the sinographic writing system; fortunately, none of them have been taken

Originally written for *MAPS* 50 (2020). Reprinted with permission.

[1] = the more the written representation of language is close to the spoken one, the better and more efficient it is.

seriously [18]. There is even a Simplified Spelling Society (founded in 1908, in London), that publishes a journal. Even if most linguists today do not necessarily share Saussure's scornful position vis-à-vis writing, he did succeed in moving writing outside the scope of scientific study for more than half a century.

Linguists in France, Germany, Japan, started to escape the phonocentric ideology only as late as the 1980s [2, 8, 10, 12]. Using again Anis's terminology [2], some linguists have adopted the "autonomistic" principle, that states that writing is as important as speech, and that we can study the former without necessarily referring to the latter; others have adopted a less radical position, called "phonographism", which states that writing is important but to study it we necessarily need to consider its interaction with speech.

The difference between the two approaches becomes clear when we look at the way these two currents define the minimal unit of the writing system, called *grapheme*. For autonomists, a grapheme is defined analogously to phonemes: we start by considering drawings created for communication purposes, called *graphs* [24], and then we build equivalence classes of graphs needed to build a system for a specific language, and we call them *graphemes*. For phonographists [5], *graphemes* are defined as merely written representations of phonemes or of morphemes; in the French word *chats*, pronounced /ʃˈa/, ⟨ch⟩ is a grapheme since it represents the phoneme /ʃ/ and ⟨s⟩ is a grapheme since it represents the (mute) morpheme [s] of plural number.

In analogy to phonology the new discipline that studies writing from a systemic point of view should be called "graphology", but unfortunately that name was already taken by a pseudo-science. Many names have been proposed ("graphemics", "graphematics", "grammatology", "graphonomics", etc.). In this paper we will keep the name "graphemics" for the discipline that stands at the same level as "phonology", and "graphetics" [24] for the discipline that stands at the same level as "phonetics".

The discipline of grapholinguistics goes a step further: it aims to study aspects of language that are particular to its written representation, at all levels of linguistics, starting with graphetics, graphemics, and continuing with morphology, syntax, semantics,

2 The conference *Grapholinguistics in the 21st Century*

2.1 The 2018 conference and proceedings

In August 2016 the author began to contact researchers in the domain of grapholinguistics, advancing the idea of a conference in the field, and more specifically a conference that would be interdisciplinary and bring together people from linguistics, computer science, typography and other areas. Their reactions were immediately very positive and encouraging. There was consensus in favor of such an event.

The 2018 conference took place in Brest, from June 14 to June 15. It lasted only two days, but these days were very intense: the keynote speakers were Florian Coulmas (*The Best Writing System of the World*, a provocative title for a very insightful talk [9]) and Christa Dürscheid (*Image, Writing, Unicode*, a talk involving emojis and Unicode as the guardian of the future of writing [11]). Both Florian and Christa are leading researchers in the field, and they both have written seminal books ([8] and [10]). Besides the keynote talks, we had 20 regular talks, from scientists and scholars coming from all around the world (Europe,[2] the US, India, Japan, China).

All talks were recorded. The interested reader can find the recordings on YouTube via the conference Web site.[3]

After the conference the author was in search for a publisher for the proceedings. This turned out to be a nightmare: one notorious scientific publisher would accept and publish only the technical papers; another famous publisher specializing in linguistics considered the topic of graphemics to be unworthy of his publication goals; then there was a third notorious publisher who accepted immediately but asked a ridiculously high amount of money to "cover the editorial fees". Others would publish a book with chapters but not proceedings... It became clear that the only way of publishing decently the proceedings of a conference in such a topic would be to build one's own infrastructure.

And this is what has been done. The author's wife Tereza founded a publishing house, called *Fluxus Editions* and based in Brest. During the spring of 2019, conference participants expanded their talks into research papers and the proceedings were published in November 2019, as the first volume of the *Grapholinguistics and Its Applications* Series (ISSN 2534–5192).

The cover of the book displays a beautiful and enormous work of calligraphy by the Japanese artist Yuichi Inoue, discovered in the summer of 2019 in a beautiful museum in the Japanese town Niigata. To best appreciate this kind of calligraphy, the reader is encouraged to watch the YouTube video `https:`

[2] We do not mention the UK separately because at that time it was still part of Europe, but yes, there were attendees from the UK.

[3] `http://conferences.telecom-bretagne.eu/grafematik/`

Figure 1: The cover of the 2018 Proceedings

//www.youtube.com/watch?v=Fnhg5hKp4WY where e[4] will realize the effort and suffering it takes to move a brush probably weighing over 20 kg in order to paint a 1.5 meter tall Chinese character. That calligraphy was chosen for the cover of the book because it is an artifact at the limits of writing, big, sublimely clumsy, hard to decipher, deeply human.

The book is published in OpenEdition mode, i.e., the PDF of the book is freely available on the publisher's Web site[5] and paper copies of the book can be bought on Amazon, printed on demand. Other books will follow, such as an important manifesto of grapholinguistics by Dimitrios Meletis (*The Nature of Writing: A Theory of Grapholinguistics*, [25]) and a major classic of the field: Gérard Blanchard's *Sémiologie de la typographie* [4]. Incidentally, both of these books started as PhD theses: the latter as a 1980 thesis at the Sorbonne in 1980, with Roland Barthes (and others) as advisor(s), and the former as 2019 thesis at the University of Graz, with Christa Dürscheid (and Bernhard Hurch).

[4] We use Spivak gender-neutral pronouns: e = he/she, eir = his/her, cf. https://en.wikipedia.org/wiki/Spivak_pronoun.

[5] http://www.fluxus-editions.fr/

2.2 The 2020 Conference

The 2018 conference being a success, except perhaps from a geographical point of view (Brest is 600 km away from Paris), it was originally decided to organize the next edition of the conference in Paris, but now via video due to the global health situation, from June 17 to June 19. It seems that this was a wise decision because at the moment this text is written, more than thrice as many submissions have been received than for Brest in 2018.

To give the reader a better idea of what the conference is about, here is an annotated list of topics:

2.2.1 Epistemology of grapholinguistics: history, onomastics, topics, interaction with other disciplines

What should we call this discipline? (As strange as it may seem, the issue of naming the discipline is a hot one, as can be seen by the following anecdote: a very famous grapholinguist emphatically left the program committee of the 2018 conference because the program committee was not inclined to use a different term than "graphemics", as e suggested...) How is grapholinguistics located vs. other disciplines? Meletis [24, p. 12] notes that in contrast to phonology, phonetics is often considered as being a natural science — should we consider that graphetics is a natural science as well?

2.2.2 Foundations of grapholinguistics, graphemics and graphetics

The first works on the foundations of grapholinguistics appeared in the eighties ([2] in France, [21] in Germany). This makes grapholinguistics a young discipline and there is still a lot to explore even on the foundational level.

2.2.3 History and typology of writing systems, comparative graphemics/graphetics

Exploring writing systems gives one an Indiana-Jones-like feeling; they can be as exotic as the Rongorongo script of Easter Island, and as common and universal as the Latin script and its ramifications; it is always a thrill to gain insight and to compare. Not to mention marginal cases: what about sign writing? (We all agree sign language is a language, but what about writing it down? [13].) Or air writing of kanji characters? ("Air writing" is making a spontaneous abstract gesture with the fingers to describe a kanji character [37].)

2.2.4 Semiotics of writing and of writing systems

How is meaning produced through writing? What are the main ways, and what are the alternatives of meaning production through this activity? As an example, the very interesting study [36] mentions a French flag with a circumflex accent in the middle. If you wonder what that is, it is actually two things: first transforming the circumflex accent as a symbol of the loss of values (after a spelling reform in 2016) and second using it as a graphical reminder of Petain's flag of Vichy (remember the Vichy water bottle Rick throws away after having killed Major Strasser in *Casablanca*?). The circumflex accent becomes an instrument of French nationalist propaganda.

2.2.5 Computational/formal graphemics/graphetics

Starting with Montague [26] and Chomsky [6, 7] in the late fifties and sixties, there have been many approaches to model language through mathematical structures. A first step in the formalization of graphemics in similar ways has been undertaken in 2001 by Richard Sproat [35]. This is a topic where much remains to be done.

2.2.6 Grapholinguistic theory of Unicode encoding

Whenever writing becomes digital, Unicode is involved. Browsing the Unicode charts one may have the impression that everything has been taken care of, and that one has the luxury of being able to write in any script of the world, whether current or extinct. But with great power comes great responsibility, and Unicode has made choices that will definitely affect writing systems for centuries to come. Therefore Unicode has to be studied as an agent in the grapholinguistics arena (e.g., [17]).

2.2.7 Orthographic reforms, theory and practice

Orthographic reforms are in the core of grapholinguistics since they change the way language is written (supposedly leaving oral language untouched but this ends up not being true[6]). Insisting on the fact that some spelling reforms (like the Greek monotonic reform [16]) have been disasters is pointless. But the story of how some populations managed to resist a reform and to return to the previous state of a writing system (cf. [22] for Malayalam) can be empowering. Studying the impact of a reform can prevent errors in future reforms.

[6] Moschonas in [27, p. 265] argues that the current tendency of pronouncing $\nu\tau$ as /d/ rather than as /nt/, in the Greek language, may come partly from the fact that according to reformed hyphenation rules, this digraph is not broken.

2.2.8 Writing and art / Writing in art

Everybody knows Magritte's “Ceci n'est pas une pipe”, a sentence written inside his painting “La Trahison des images” [the betrayal of images], underneath the image of a pipe. Writing inside painting is not new: Byzantine icons have done it for centuries. But writing also appears in comics, in movies, in sculpture (like the man-made-of-letters sculptures by the Catalan artist Jaume Plensa). And there is the use of typography in literature, as in the Dada or De Stijl movements, in Mallarmé's “Un coup de dès”, Apollinaire's “Calligrammes” and in many other works. An endless source of knowledge and excitement.

2.2.9 Sinographemics

All about the Chinese script and its extended family: Japanese kanji, Korean hancha, Vietnamese chữ nôm and chữ hán. Sinographemics is an important topic of the conference because there is so much to say about the nature, structure and usage of Chinese characters, a script used by 1.3 billion people.

2.2.10 Typographemics, typographetics

The study of the printed representation of language. Typography is only half a millennium old, but it is in part responsible for the fabulous technological and social advances of this period. Typography has developed its own codes and, before creating TeX and METAFONT, Donald E. Knuth has studied typographe[mt]ics in depth [19, 20]. As a subdiscipline of graphe[mt]ics, typographe[mt]ics becomes a subdiscipline of linguistics: the creative power of typography, scrutinized with scientific methods.

2.2.11 Texting, latinization, new forms of written language

Technology always carries the cultural signature of its creator(s). Computer science has evolved in Latin-alphabet-language countries, programming languages use it, and hence the Latin alphabet has become a trademark of modernity (and some will say, globalization). No wonder that people (and especially young people) using modern technologies, modern communication media, social networks, etc., have a tendency to use the Latin alphabet to express their vision of the world, even though their native language uses some other script. This behavior is interesting per se and raises the question of what will happen in the future.

2.2.12 ASCII art, emoticons and other pictorial uses of graphemes

Long before ASCII art, writing was used pictorially; see, for example, the wonderful anthology of typewriter art, by Barrie Tullett [38]. But there is also the opposite trend: instead of combining graphemes to form shapes (and graphical meaning), one can create new graphemes that encompass pictorial meaning; that is the case of emoticons and emojis. Are they graphemes? They sure are Unicode characters, and their emergence was very beneficial to the Unicode Consortium since they made it known to the masses.

2.2.13 The future of writing, of writing systems and styles

Futurology is a very exciting field because in the last decades its predictions have repeatedly been proven wrong. Will the future will be bright like in the movie *Bicentennial Man*, or post-apocalyptic like in *Mad Max*? And what about writing? Will our descendants, in a century or so, use only emojis, like Xu Bing in his book [3]? Or will Unicode make ours the best possible world, where every minority will safely preserve and nourish its own language and writing system, while English and the Latin alphabet become the de facto communication tool?

2.2.14 Graphemics/graphetics of science fiction and astrolinguistics

How did science-fiction authors imagine alien communication, or human communication in the future? What about signals from extraterrestrials, as in the movie *Contact*? Science fiction is just fiction, but there is a scientific discipline, namely *astrolinguistics*,[7] that takes the issue seriously: the reader can consult the book [29], which describes a logical approach to communication with other living entities. After all, we had better be ready before they arrive.

2.2.15 Graphemics/graphetics and font technologies

We now enter into more technical issues. Font technologies have always interested TeX users, since TeX has survived them all: GF, PK, PFB, TTF, OTF, ... (see [14] for more).[8] Fonts are bridges between characters and glyphs, between graphemes and graphs. They deserve a careful grapholinguistic study.

2.2.16 Graphemics/graphetics in steganography and computer security

Steganography is a cryptographic method whereby the very existence of a hidden message in a text is hidden: the goal is to transmit the message "under the nose" of a third person. Graphetic methods have been used for this, for example by adding supplementary line segments between connected letters in Arabic text, by moving around dots [34] or by varying keshideh widths [1], etc. Phishing can occur on the Unicode level, when homographic characters are used (characters with identical glyphs, such as Cyrillic ⟨a⟩ or Greek ⟨o⟩).

2.2.17 Graphemics/graphetics in experimental psychology and cognitive sciences

You probably have heard of dyslexia — there are special fonts for people suffering from it. How are they created, evaluated, used? [32] More generally, what can reading/writing and its deficiencies teach us about the way our brain works? [23] Can you imagine the pathology where a patient can draw Chinese characters without problem, but is unable to read them once written? And besides pathologies, there are many question about education: how should reading/writing be taught? Syllable-wise or letter-wise? Does the Joyo Kanji progression of kanji characters taught in school make sense? [31] And how does it affect the knowledge of Japanese language by the hundreds of millions of Japanese people who learned it that way?

2.2.18 Grapholinguistic applications in natural language processing and text mining

Last but not least comes computer science and the way it processes language. Until now, Natural Language Processing has paid very little attention to graphemes. It considers that data have an atomic level, namely the (Unicode) character. Glyphs do not matter, neither do styles (bold, italic, underline) or font sizes. This attitude will not last: texts are written by humans and artificial intelligence aspires to extract as much information as possible from them. Humans use glyphs and styles and font sizes. A text

[7] Not to be confused with *astroarcheology*, which is another pseudo-science. A fascinating one, but nevertheless not obeying scientific rules.

[8] But we shouldn't forget that a lion needs a lioness, and that the Great Master created TeX to work in a binary system: TeX and METAFONT. The lioness's genes flow in our blood and even though we use modern font technologies we aspire to more, and METAFONT is definitely more, an ideal still to be reached.

written in Comic Sans does not carry the same information as a text written in Monotype Ehrhardt. Sooner or later NLP will acknowledge this fact, and the conference may help to make this happen.

3 What has Grapholinguistics to do with TEX?

In his infinite creativity and productivity, Donald E. Knuth has not created, in TEX, simply a program for typesetting. He modeled the whole process of written document production. In grapholinguistic terms, he modeled graphs and graphemes (called "glyphs" and "characters" in TEX jargon), one-dimensional graphemic sequences (called "character strings" or "glyph strings") which he placed into abstract recipients called "hboxes", and two-dimensional graphemic sequences ("vboxes"). He also modeled grapholinguistic processes such as kerning, hyphenation, line breaking, page breaking, and so on.

Thus, besides being a programming language and a program, TEX is also an abstract model of the graphemic level of language. It is no wonder that the community of TEX aficionados has contributed much to the study of written language, even if the terminology used was not the one of grapholinguistics as it has emerged in the last thirty years.

Adapting TEX to various languages and writing systems has led to grapholinguistic studies of these languages and writing systems. As a simple example: the fact that ligatures between components of German words have to be broken (as in "Auflage") has been known in the TEX community at least since the 1980s [30] (and maybe even earlier). In grapholinguistic lingo, this becomes a principle: "ligatures are an intergraphemic but intramorphemic phenomenon" [17]. It should come as no surprise that grapholinguistic studies such as [28] cite TEX-related publications (such as the French *Cahiers GUTenberg*) among their references.

TEX is at the forefront of studies on the written language, and some day its contribution to the emerging discipline of grapholinguistics will be duly examined and acknowledged.

4 Why should TEX users attend the *Grapholinguistics in the 21st Century* conference?

TEX conferences are great places to meet people and exchange information. They are unforgettable events attracting pilgrims from all over the world. TEX conferences have a great advantage which is also their disadvantage: they deal mainly with TEX, its descendants and its applications. Linguists, historians, psychologists, educators, artists will occasionally visit TEX conferences, but mostly because they are themselves TEX users or developers. The *Grapholinguistics in the 21st Century* conference has a goal that goes beyond TEX meetings, namely to attract scientists and practitioners from various horizons, to discuss writing.

Of course, *Grapholinguistics in the 21st Century* also gathers people we are used to seeing in TEX meetings: typographers, font designers, Unicode aficionados. All in all, this conference aims to use grapholinguistics as the common ground for all kinds of people interested in the written word to exchange ideas. It is an interdisciplinary conference (and this is both a gift and a curse, as Mr. Monk would say) based on the principle that somebody interested in writing will be interested in writing in eir own domain but also in other domains, and therefore will be interested in meeting people dealing with writing in different ways. How many places on Earth are there where an historian of writing will meet a font designer, a linguist specializing in punctuation will meet a psychologist studying second-language learning in a different writing system, or an artist having invented a writing system and engraved it on the roof of the library of the Sidgwick campus of the University of Cambridge? None, in fact.

And speaking of places on Earth, the *Grapholinguistics in the 21st Century* goes a step farther and also considers writing outside our good old planet: Jessica Coon, one of the three keynote speakers of the conference, has been the linguistic advisor of the well-known science-fiction blockbuster *Arrival* by Denis Villeneuve, a $47M budget and $203M box office movie that was the first one in history to have a linguist in the leading role. This is a nice revenge on Saussure since the movie shows aliens communicating with humans through a dynamic writing system. (Aliens are heptapods throwing ink to a glass barrier between their liquid environment and a human-friendly environment, ink forms moving patterns that are analyzed by the linguist — the exceptional Amy Adams — who manages to communicate with them.) Unlike SF movies of the sixties where the whole universe is unsurprisingly speaking English, here a sophisticated writing system is used by aliens and we witness Amy Adams' efforts to decipher it. Jessica Coon is the (real-world) linguist who made this movie scientifically sound, and she will share her thoughts about *The Linguistics of Arrival: What an alien writing system can teach us about human language* with us.

Therefore the answer to the question "Why should TEX users attend the *Grapholinguistics in the 21st Century* conference?" is simply: "for the fun of it". Because TEX users[9] share an intimate love and care for the written word, and the conference will gather exactly this kind of people, now from all horizons.

Obviously, love and care for the written word is not restricted to TEX users. Therefore, oh gentle reader of this text, whether a TEX user or not, join us!

References

[1] A. Al-Azawi and M. Fadhil. Arabic text steganography using kashida extensions with Huffman code. *Journal of Applied Sciences* 10:436–439, 2010.

[2] J. Anis. *L'écriture, théories et descriptions.* De Boeck, 1988.

[3] X. Bing. *Book from the Ground: from point to point.* MIT Press, Cambridge, 2014.

[4] G. Blanchard. *Sémiologie de la typographie.* Fluxus Editions, Brest, to appear.

[5] N. Catach. L'écriture en tant que plurisystème, ou théorie de *L* prime. In *Pour une théorie de la langue écrite*, pp. 243–256, Paris, 1988. Éditions du CNRS.

[6] N. Chomsky. *Syntactic structures.* Mouton, 1957.

[7] N. Chomsky and M. Halle. *The Sound Pattern of English.* Harper & Row, 1968.

[8] F. Coulmas. *Writing Systems.* Cambridge University Press, 2003.

[9] F. Coulmas. „Die Buchstabenschrift ist an und für sich die intelligentere." Überlegungen zur Bewertung von Schriftsystemen. In Y. Haralambous, ed., *Proceedings of Graphemics in the 21st Century, Brest 2018*, pp. 1–16, Brest, 2019. Fluxus Editions. `https://doi.org/10.36824/2018-graf-coul`

[10] C. Dührscheid. *Einführung in die Schriftlinguistik.* Vandenhoeck & Ruprecht, 2016.

[11] C. Dürscheid and D. Meletis. Emojis: A Grapholinguistic Approach. In Y. Haralambous, ed., *Proceedings of Graphemics in the 21st Century, Brest 2018*, pp. 167–183, Brest, 2019. Fluxus Editions. `https://doi.org/10.36824/2018-graf-duer`

[12] H. Glück. *Schrift und Schriftlichkeit.* J.B. Metzler, Stuttgart, 1987.

[13] D. A. Grushkin. Writing signed languages: What for? What form? *American Annals of the Deaf* 161(5):509–527, 2017. `https://doi.org/10.1353/aad.2017.0001`

[14] Y. Haralambous. *Fonts & Encodings. From Advanced Typography to Unicode and Everything in Between.* O'Reilly, Sebastopol, CA, 2007.

[15] Y. Haralambous. TEX as a path, a talk given at Donald Knuth's 80th birthday celebration symposium. *TUGboat* 39(1):8–15, 2018. `https://tug.org/TUGboat/tb39-1/tb121haralambous-knuth80.pdf`

[16] Y. Haralambous. Phonocentrism in Greece: Side effects of two centuries of diglossia. poster presented at *AWLL12*, Cambridge, UK, `https://hal.archives-ouvertes.fr/hal-02480230`, 2019.

[17] Y. Haralambous and M. Dürst. Unicode from a linguistic point of view. In Y. Haralambous, ed., *Proceedings of Graphemics in the 21st Century, Brest 2018*, pp. 167–183, Brest, 2019. Fluxus Editions. `https://doi.org/10.36824/2018-graf-hara1`

[18] C. Holcombe. *A History of East-Asia.* Cambridge University Press, 2011.

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

[20] D. E. Knuth and M. F. Plass. Breaking paragraphs into lines. *Software—Practice and Experience* 11:1119–1184, 1981.

[21] M. Kohrt. *Problemgeschichte des Graphembegriffs und des frühen Phonembegriffs.* Niemeyer, Tübingen, 1985.

[22] K. Manohar and S. Thottingal. Malayalam Orthographic Reforms. Impact on Language and Popular Culture. In Y. Haralambous, ed., *Proceedings of Graphemics in the 21st Century, Brest 2018*, pp. 329–351, Brest, 2019. Fluxus Editions. `https://doi.org/10.36824/2018-graf-mano`

[23] D. Martin, ed. *Researching Dyslexia in Multilingual Settings.* Multilingual Matters, Bristol, Buffalo, Toronto, 2013.

[24] D. Meletis. *Graphetik. Form und Materialität von Schrift.* Verlag Werner Hülsbusch, Glückstadt, 2015.

[25] D. Meletis. *The Nature of Writing: A Theory of Grapholinguistics.* Fluxus Editions, Brest, to appear.

[9] See [15] for a description of the way TEX acts on people using it.

[26] R. Montague. English as a formal language. In B. Visentini et al., eds., *Linguaggi nella Società et nella Tecnica*, pp. 188–211. Edizioni di Comunità, 1970.

[27] S. Moschonas. *Ideology and Language.* Patakis, Athens, 2005. (In Greek).

[28] T. Nehrlich. Phänomenologie der Ligatur. Theorie und Praxis eines Schriftelements zwischen Letter und Lücke. In M. Giertier and R. Köppel, eds., *Von Lettern und Lücken: zur Ordnung der Schrift im Bleisatz*, pp. 13–38, Paderborn, 2012. Wilhelm Fink.

[29] A. Ollongren. *Astrolinguistics. Design of a Linguistic System for Interstellar Communication Based on Logic.* Springer, 2013.

[30] H. Partl. German TEX. *TUGboat* 9(1):70–72, Apr. 1988. `https://tug.org/TUGboat/tb09-1/tb20partl.pdf`

[31] S. R. Paxton. *Tackling the Kanji hurdle. An investigation of Kanji order and its role in facilitating the Kanji learning process.* Ph.D. thesis, Macquarie University, Sydney, Australia, 2015.

[32] L. Rello and R. Baeza-Yates. Good fonts for dyslexia. In *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility*, pp. 14:1–14:8, 2013. `https://doi.org/10.1145/2513383.2513447`

[33] F. d. Saussure. *Course in General Linguistics.* McGraw-Hill, New York, 1966. C. Bally, A. Secehahye, A. Riedlinger, eds. B. Wade, tr.

[34] M. Shirali-Shahreza and M. Shirali-Shahreza. A new approach to Persian/Arabic text steganography. In *5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering, Software Architecture and Reuse (ICIS-COMSAR'06)*, pp. 310–315, July 2006. `https://doi.org/10.1109/ICIS-COMSAR.2006.10`

[35] R. Sproat. *A Computational Theory of Writing Systems.* Cambridge University Press, Cambridge, 2000.

[36] C. Tebaldi. From #Je Suis Circonflex to #Je Suis Cornflakes: Racialization and Resemiotization in French Nationalist Twitter. In *AAA*, 2017. `https://umass.academia.edu/CatherineTebaldi`.

[37] M. Thomas. "Air Writing" and Second Language Learners' Knowledge of Japanese Kanji. *Japanese Language and Literature* 47(1):23–58, 2013.

[38] B. Tullett. *Typewriter Art: A Modern Anthology.* Laurence King Publishing, London, 2014.

◇ Yannis Haralambous
IMT Atlantique & CNRS UMR 6285
Lab-STICC
Technopôle Brest-Iroise CS 83818
29238 Brest Cedex 3
France
`yannis.haralambous (at) imt-atlantique.fr`
`https://grafematik2020.sciencesconf.org/`

The DuckBoat — Beginners' Pond: You do not need to be Neo to cope with a Ti*k*Z matrix

Herr Professor Paulinho van Duck

Abstract

In this installment, Prof. van Duck will show you some tips & tricks about the useful Ti*k*Z library `matrix`.

1 I am back!

Hi, (LA)TEX friends!

Did you miss me? Don't worry; I am back as quacky as usual!

Lately, I have been very busy at work. Moreover, I am helping my friend Carla with her thesis. Hence, I haven't much time to dedicate to the DuckBoats, so I decided to write only one article a year.

I have also changed the title of my column; now it is "Beginners' Pond," to better highlight the targeted audience.

Indeed, my articles may appear too trivial for a serious journal like the *TUGboat*, but, as the volleyball coach Julio Velasco once said, "there are no *easy* or *difficult*, but things someone *can* do or *cannot!*"

The name changing is also a way not to be linked only to TEX.SE, since new Q&A sites are emerging.

For example, as Barbara announced in her column, I would call your attention to Top Answers (`https://topanswers.xyz/tex`). The platform itself is open source, developed with the community in mind and totally non-profit. The TEX community there is still small, but eager to answer your questions, and would be more than happy to see new users. So if this sounds interesting to you, just drop by and have a look yourself.

My commitments did not prevent me from attending the GuIt *meeting 2019*, where I had the occasion to meet Bär once again, together with many other friends like Ulrike Fischer, her husband Gert, and prof. Enrico Gregorio.

It was a very special event because we celebrated prof. Claudio Beccari, who will retire from the group's official positions due to age limits, but he surely will not give up helping everyone on the Forum.

This time, I will show you a useful Ti*k*Z library, `matrix`. It allows building matrices of Ti*k*Z objects. It could be a convenient alternative for node positioning, and it is also useful, for example, to create math matrices with some graphical adding.

Before getting to the heart of the matter, I would like to highlight a handy post on TEX.SE Meta, which collects the most often referenced questions: `https://tex.meta.stackexchange.com/questions/2419/often-referenced-questions`.

They are grouped by topic, so it is straightforward to find what you are looking for; most of the posts listed should be a "must-read" for any beginner.

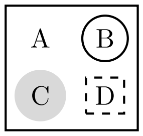
Hence, take a look at the *Often referenced questions* post on Meta before asking, maybe you will find a solution at once.

Last but not least, both newbies and experts are invited to contribute to keeping it up-to-date!

2 Quack Guide No. 5 The (Ti*k*Z) Matrix

A Ti*k*Z matrix is analogous to an ordinary `tabular`, but its elements are Ti*k*Z objects.

For example, you can build a table with nodes, paths, whatever you prefer:



```
\documentclass{standalone}
\usepackage{tikz}
\usetikzlibrary{matrix}
\begin{document}
  \begin{tikzpicture}
    \matrix[
      draw, thick,
      column sep=.2cm, row sep=.1cm
      ] {
      \node {A}; &
        \draw (0,0) circle (.3cm) node {B};\\
      \node[fill=gray!30, circle] {C}; &
        \node[draw, dashed] {D};\\
      };
  \end{tikzpicture}
\end{document}
```

Like any other Ti*k*Z library, the first thing is to load it with `\usetikzlibrary{matrix}`, after having loaded the package Ti*k*Z itself.

Then, in your `tikzpicture` environments, you will be able to use the command:

`\matrix[`⟨*options*⟩`] (`⟨*name*⟩`) at (`⟨*coords*⟩`)`
  `{`⟨*content*⟩`};`

As usual, you have some ⟨*options*⟩ in square brackets. For example, `column sep` and `row sep` set the space between the columns/rows. Since the matrix itself is a node, you may use other node options, such as `draw` or `thick`.

Note that here they refer to the matrix, they are not always inherited by the nodes it contains. For instance, the option thick has effect both on the matrix and its elements (see B and D in the previous image); whereas the draw only draws the border *of the matrix* and the path (B), but not the borders of the \nodes A and C. You can use node={⟨*options*⟩} to make some styles apply to every node of a matrix (or, in general, of a tikzpicture).

You may also give your matrix a ⟨*name*⟩, if needed, or position it at desired coordinates, ⟨*coords*⟩.

In the ⟨*content*⟩, there are the cells of your matrix, separated by &, and with \\ for ending the rows, as in a usual table. But pay attention: the \\ is mandatory also for the last row.

Another difference with respect to a tabular is that you do not have to state in advance the number of columns; you can add or remove as many columns you like, without the need to change any specification.

If the elements of your matrix are all nodes, you may use the option matrix of nodes, and put only the *text* of the nodes in the cells.

Indeed, it would be annoying to write
\node{...};
in every cell, but LaTeX is fun, quack! Boring activities are avoided as much as cats avoid water!

Matrices of nodes are by far the most popular; I will show you some examples in the following.

2.1 A convenient way for positioning

In the DuckBoat "The Morse code of Ti*k*Z" [1], I used the positioning library to place a node above/below/left/right to another.

It can be done very easily also with a Ti*k*Z matrix. The advantage of using a matrix is that, if you have to add a node between other existing nodes, you do not have to change the positioning option of the other nodes.

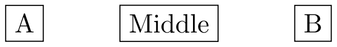
Let me explain better with an example. Suppose you are using right = of ... to position two nodes in the following way:

```
\documentclass{standalone}
\usepackage{tikz}
\usetikzlibrary{positioning}
\begin{document}
  \begin{tikzpicture}[nodes={draw}]
    \node (A) {A};
    \node[right= of A] {B};
  \end{tikzpicture}
\end{document}
```

The same result can be achieved also with a Ti*k*Z matrix:[1]

```
\usetikzlibrary{matrix} % <-- in preamble
...
\begin{tikzpicture}
  \matrix[
    matrix of nodes,
    nodes={draw},
    column sep=1cm
    ] {
    A & B\\
    };
\end{tikzpicture}
```

Now, if you would like to add a node in the middle, with the positioning library you have to change the reference node of B:



```
\usetikzlibrary{positioning} % <-- in preamble
...
\begin{tikzpicture}[nodes={draw}]
  \node (A) {A};
  \node[right= of A] (midnode) {Middle};
  \node[right= of midnode] {B};
\end{tikzpicture}
```

Of course, with only two nodes, it is not a big effort, but imagine to have a more complex picture, with many nodes linked together, it could become complicated, and tedious.

With a Ti*k*Z matrix you can simply add the new node as a new element of the matrix:

```
\usetikzlibrary{matrix} % <-- in preamble
...
\begin{tikzpicture}
  \matrix[
    matrix of nodes,
    nodes={draw},
    column sep=1cm
    ]{
    A & Middle & B\\
    };
\end{tikzpicture}
```

It is convenient, is it not?

As I said earlier, with column/rows sep you can set the width between columns/rows, the same value for all the columns/rows of your matrix.

If you would like to increase or decrease the distance between two specific rows, use [⟨*height*⟩] after \\, as in an ordinary table.

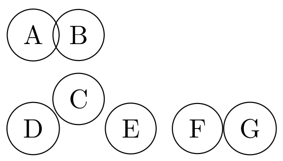
There is a little difference, instead, in the way to set the distance between two specific columns. In a

[1] Hereinafter, for convenience, I will show only the tikzpicture environments, unless something more is needed.

Ti*k*Z matrix, you cannot use the @-expression from `array` package (e.g., `@{\hspace{`⟨*width*⟩`}}`), because the parameter for table specification is not present.

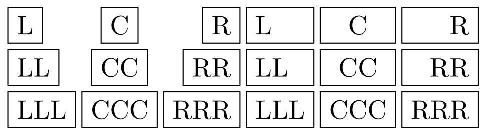
You can achieve the same result by adding, in the first row, the option `[`⟨*width*⟩`]` after the `&` which separates the two columns involved. If the first row has fewer elements of the other rows, you have to add the necessary empty elements to use this feature.

An example is worth a thousand words:



```
\begin{tikzpicture}
  \matrix[matrix of nodes,
    nodes={draw,circle}]{
      A &[-1mm] B  & &[2mm] \\[1.5mm]
      & C \\[-3.1mm]
      D &  & E & F & G\\};
\end{tikzpicture}
```

Since there is no table specification, you cannot set the column alignment with `l`, `c`, or `r` as usual, but you may use anchors or give a dimension to the nodes and align the text inside. Let's draw the nodes to see the difference between the two approaches:



```
\begin{tikzpicture}
  \matrix[
    matrix of nodes,
    nodes={draw},
    column sep=2pt, row sep=2pt,
    column 1/.style={nodes={anchor=base west}},
    column 2/.style={nodes={anchor=base}},
    column 3/.style={nodes={anchor=base east}},
    column 4/.style={
      nodes={text width=width("LLL"),
        align=left}},
    column 5/.style={
      nodes={text width=width("CCC"),
        align=center}},
    column 6/.style={
      nodes={text width=width("RRR"),
        align=right}},]{
    L   & C   & R   & L   & C   & R\\
    LL  & CC  & RR  & LL  & CC  & RR\\
    LLL & CCC & RRR & LLL & CCC & RRR\\
    };
\end{tikzpicture}
```

Please note that the option
`column` ⟨*m*⟩`/.style={`⟨*options*⟩`}`
sets a style for all the nodes in column ⟨*m*⟩, and the expression `width("`⟨*string*⟩`")` is the Ti*k*Z equivalent of `calc`'s `\widthof` to get the width of the box containing ⟨*string*⟩. There are also the analogous functions `height("`⟨*string*⟩`")` and `depth("`⟨*string*⟩`")`.

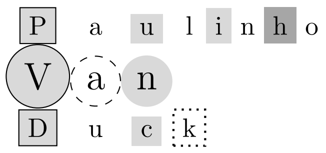
2.2 Formatting columns, rows, single cells

In the previous examples, we have already encountered `nodes={`⟨*options*⟩`}`, an abbreviation for
`every node/.append style=`⟨*options*⟩
(the `append` means that the style is added to the already existing options). We have also seen how to format the nodes of a column.

There are similar features for formatting a row or a specific cell. There are also options for formatting every even or odd column/row.

In the case of matrices of nodes, there is also an alternative way to set the options of a single node: putting `|[`⟨*options*⟩`]|` just before the node text.

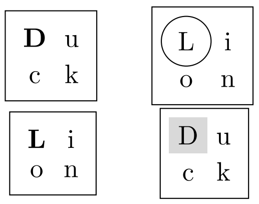
An example:



```
\begin{tikzpicture}
  \matrix[
    matrix of nodes,
    column 1/.style={nodes={draw}},
    row 2/.style={nodes={circle}},
    row 2 column 2/.style={
      nodes={draw, dashed}
      },
    every odd column/.style={
      nodes={fill=gray!30}
      },
    every even row/.style={
      nodes={font=\Large}
      },
    ]{
    P&a&u&l&i&n&|[fill=gray!70]|h&o\\
    V&a&n\\
    D&u&c&|[draw, dotted, thick]|k\\
    };
\end{tikzpicture}
```

If you have many matrices with the same formatting, you can create a style for the whole matrix and use it whenever you need it, or set a style for all your matrices with
`every matrix/.style={`⟨*options*⟩`}`.

As usual, you may set the style with `\tikzset`, if you'd like to use it in several of your Ti*k*Z pictures throughout your document, or as an option of the specific `tikzpicture` environment where the style is needed.



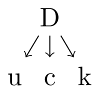
```
...
\tikzset{
% styles applied throughout the document
  every matrix/.style={
    matrix of nodes, draw
    },
  matrdoc/.style={
    row 1 column 1/.style={
      nodes={font=\bfseries}
      }
    },
  }
...
\begin{document}
\begin{tikzpicture}[
% style applied throughout this tikzpicture
  matrpic/.style={
    row 1 column 1/.style={
    nodes={draw, circle}}
    }
  ]
  \matrix[matrdoc] {
    D&u\\c&k\\
    };
  \matrix[matrpic] at (2,0) {
    L&i\\o&n\\
    };
\end{tikzpicture}

\begin{tikzpicture}[
% style applied throughout this tikzpicture
  matrpic/.style={
    row 1 column 1/.style={
      nodes={fill=gray!30}
      }
    }
  ]
  \matrix[matrdoc] {
    L&i\\o&n\\
    };
  \matrix[matrpic] at (2,0) {
    D&u\\c&k\\
    };
\end{tikzpicture}
\end{document}
```

2.3 Naming matrix elements

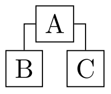
Once you give a ⟨*name*⟩ to your matrix, adding it in round brackets or with the option `name=`⟨*name*⟩, you can reference any cell with ⟨*name*⟩-⟨*n*⟩-⟨*m*⟩, where ⟨*n*⟩ is the row number of the cell and ⟨*m*⟩ the column number.

This notation is very convenient if you need to perform a repeated action on your cells:



```
\begin{tikzpicture}
  \matrix[
    matrix of nodes,
    text height=height("k")
    ] (mymatr) {
    &D\\[3mm]
    u&c&k\\
    };
  \foreach \myind in {1,2,3}
    \draw[->] (mymatr-1-2) --
      (mymatr-2-\myind);
\end{tikzpicture}
```

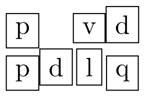
In addition, sometimes it is useful to give a mnemonic name to a node; you can do this by putting `|[name=`⟨*name*⟩`]|` before the node text:



```
\begin{tikzpicture}
  \matrix[
    matrix of nodes, nodes=draw
    ]{
    &[-1mm] |[name=start]|A &[-1mm]\\[1mm]
    |[name=nodeb]|B & & |[name=nodec]|C\\
    };
  \draw (start) -| (nodeb);
  \draw (start) -| (nodec);
\end{tikzpicture}
```

2.4 Tips & tricks about cell dimensions and borders

When the nodes are drawn, you would expect to have cells with homogeneous dimension, but the result could seem strange:



```
\begin{tikzpicture}
  \matrix[matrix of nodes, nodes={draw}]{
    p&&v&d\\p&d&l&q\\
    };
\end{tikzpicture}
```

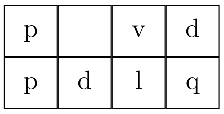
It happens because Ti*k*Z builds the bounding box according to the node content.

Explicitly setting the node `minimum width` or `text width` solves the problem for that dimension, but you cannot do the same for the height.

Well, if you read "The Morse code of Ti*k*Z", you already know the solution: use `text depth` and `text height`.

Another thing that, at first sight, may seem strange is that the empty cells have no border. It happens because Ti*k*Z considers the empty cells… empty!

If you would like to have borders everywhere, you can use the option `nodes in empty cells`.



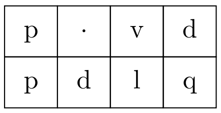
```
\begin{tikzpicture}
  \matrix[
    matrix of nodes,
    nodes in empty cells,
  nodes={draw,
      text depth=.14cm,
      text height=.3cm,
      minimum width=.7cm}
    ]{
    p&&v&d\\
    p&d&l&q\\
    };
\end{tikzpicture}
```

You can also fill the empty cell with a default value, using `execute at empty cell=`⟨*code*⟩.

However, there is still a little problem; the inner borders are drawn "twice", so they are thicker than the contour.

There is a little trick to solve this situation, using `-\pgflinewidth` as column/row separation.

Indeed, the border's width is stored in the PGF macro `\pgflinewidth`; if you decrease the column and row separators by it, you will have only one line drawn, not two.



```
\begin{tikzpicture}
  \matrix[
    matrix of nodes,
    column sep=-\pgflinewidth,
    row sep=-\pgflinewidth,
    execute at empty cell={\node{$\cdot$};},
    nodes={draw,
      text depth=.14cm,
      text height=.3cm,
      minimum width=.7cm}
    ]{
    p&&v&d\\
    p&d&l&q\\
    };
\end{tikzpicture}
```

2.5 Mysterious errors

When you put a Ti*k*Z matrix in a `\newcommand`, an error appears which leaves newbies a bit astonished:

```
! Package pgf Error: Single ampersand used
with wrong catcode.
```

You may also get something like:

```
! Extra alignment tab has been changed to \cr.
```

when the text in a matrix node is a `tabular`.

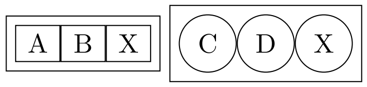
These problems happen because Ti*k*Z does not actually use `&` to separate cells, even if it seems so, but the command `\pgfmatrixnextcell`.

I will not explain (it is too difficult for me!) how Ti*k*Z substitutes the macro `\pgfmatrixnextcell` with `&`, but sometimes it fails. If you are interested, see Section 20.5 *Considerations Concerning Active Characters* of [2] for further details.

Well, don't panic, quack! There is a simple solution for it, just use the option
`ampersand replacement=`⟨*macro name*⟩.

For instance, you can set
`ampersand replacement=\&`
and then use `\&` instead of `&` as column separator for the Ti*k*Z matrix, or `\pgfmatrixnextcell` directly.

An example of a `\matrix` in a macro follows. Note that if you put the `ampersand replacement` as an option of the `tikzpicture`, it is valid for any matrix in the picture.[2]

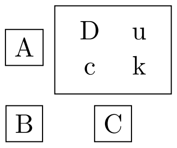


```
% The % signs present in the following macro
% definition are mandatory to avoid
% spurious spaces
\newcommand{\mymatrix}[3][]{%
  \begin{tikzpicture}[
    ampersand replacement=\&,
    nodes={draw},
    baseline=0pt
    ]
    \matrix[matrix of nodes, nodes={#1},
        text width=1em, text centered,
        ] {
        {#2} \& {#3} \& X\\
        };
  \end{tikzpicture}%
```

[2] When you define a new command, as prof. Enrico Gregorio always says, do not forget putting the `%`'s where needed, to avoid spurious spaces!
I take the occasion to thank him for suggesting the previous example and for his proofreading. Of course, all remaining errors are mine.

```
}
...
\begin{document}
  \mymatrix{A}{B} \mymatrix[circle]{C}{D}
\end{document}
```

And here is an example of a `tabular` in a cell:



```
\begin{tikzpicture}
  \matrix [
    nodes=draw,
    matrix of nodes,
    ampersand replacement=\&,
    row sep=4pt,
    column sep=4pt,
    ]{
    A \& \begin{tabular}{cc}
            D & u \\
            c & k
         \end{tabular}\\
    B \pgfmatrixnextcell C \\
    };
\end{tikzpicture}
```

2.6 Matrices of math nodes

A peculiar case of matrices of nodes are matrices of *math* nodes; we could compare them to `array` environments.

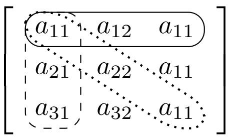
Indeed, the option `matrix of math nodes` allows you to avoid putting `$...$` or `\(...\)` in every cell. As you know, LaTeX hates repetitive boring actions, quack!

If you would like to have some delimiters around your matrix (or, in general, around any node), you can use the
`left/right/above/below delimiter =` ⟨*delimiter*⟩ option.

The ⟨*delimiter*⟩ can be anything acceptable to the ordinary `\left` command.

Of course, we do not need a Ti*k*Z matrix for a simple math matrix; there are already many math environments to create it without Ti*k*Z.

On the other hand, a Ti*k*Z matrix is useful if you would like to add some graphical accessories to it. In the following example, the first row and column are highlighted using `fit`, and the main diagonal with a `\draw`:


a_{11} a_{12} a_{11}
a_{21} a_{22} a_{11}
a_{31} a_{32} a_{11}

```
\documentclass{standalone}
\usepackage{tikz}
\usetikzlibrary{matrix, fit}
\begin{document}
\begin{tikzpicture}[
  every matrix/.style={
    matrix of math nodes,
    column sep =1mm,
    row sep =1mm,
    left delimiter={[},
    right delimiter={]},
    }
  ]
  \matrix (mymat) {
     a_{11} & a_{12} & a_{11}\\
     a_{21} & a_{22} & a_{11}\\
     a_{31} & a_{32} & a_{11}\\
     };
  \node[fit=(mymat-1-1)(mymat-1-3), draw,
    inner sep=0pt, rounded corners=6pt] {};
  \node[fit=(mymat-1-1)(mymat-3-1), draw,
    inner sep=0pt, rounded corners=6pt,
    dashed] {};
  \draw[rounded corners=5pt, dotted, thick]
    ([yshift=-2pt]mymat-1-1.west) |-
    ([xshift=2pt]mymat-1-1.north) --
    ([yshift=2pt]mymat-3-3.east) |-
    ([xshift=-2pt]mymat-3-3.south) --
    cycle;
\end{tikzpicture}
\end{document}
```

3 Conclusion

I hope you enjoyed my explanation, and if you are searching for The One, remember:

A duck can fly!
(Humans cannot)

References

[1] C. Maggi. The DuckBoat — news from TeX.SE: The Morse code of TikZ. *TUGboat* 39(1):21–26, 2018. `https://tug.org/TUGboat/tb39-1/tb121duck-tikz.pdf`

[2] T. 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`

Creating accessible pdfs with LaTeX

Ulrike Fischer

Abstract

This article describes the current state and planned actions to improve accessibility of pdfs created with LaTeX, as currently undertaken by the LaTeX Team.

1 Accessibility of pdf

The pdf language is at its core a *page description* programming language. It describes very accurately how text and graphical elements look and where they are placed on a page. But it doesn't describe the semantic meaning of the elements and the reading order; by looking at the code there is no way to know if a text is a section heading or some watermark or a footnote or if it belongs to a tabular. You can't know where a sentence has its continuation on another page or how many words a text contains, and sometimes it is even impossible to identify the characters; you only see some glyph index number and copy & paste can give gibberish.

All this is not a problem as long as the pdf is merely meant for printing or viewing but it restricts its use for digital processing like copy & paste, automatic extraction of billing data, reflowing, or using the pdf with a screen reader. For such uses you need accessible, structured, extractable content.

"Accessibility" as a standard is specified in PDF/UA. It is also included in other standards, notably PDF/A-1a and PDF/A-2a (the "a" stands for accessible). The standards contain a number of requirements to help in retrieving the content for further processing; for example, that every character has a Unicode representation (no gibberish when copy & pasting), that word spaces are correctly marked up (so that reflowing works), that the language of the document and the text is declared (so a screen reader can guess the pronunciation), that pictures have sensible alternative descriptions. And most importantly: that the document is *tagged*. This last requirement is responsible for adding structure information to the content. It marks up content as section or tabular cell or list item. This improves navigation in the document with, for example, a screen reader, but also exporting to other formats like XML.

TUG maintains a web page with links to relevant standards, articles and packages [3].

First published in *ArsTeXnica* 28 (Oct 2019), pp. 135–139. Reprinted with permission.

2 Creating a tagged pdf

Tagging consists of two main tasks. First, in the *stream object* of a page every bit of content must be marked and labelled with a number `MCID` n.

The following listing shows a small example. The `BDC` and the `EMC` lines are the start and end markers needed for tagging. The `/H1` indicates that the content is part of a sectioning element.

```
stream
/H1 <<MCID 0>> BDC
BT
/F17 14.3462 Tf 124.802 706.129
 Td [(0.1)-1100(Section)]TJ
ET
EMC
```

In the second step a number of pdf objects must be created to describe the *structure tree*. Every object contains references to the parent `/P` and to one or more kid elements `/K`. The leaf nodes are the `MCID` n created in the first step. A typical object looks roughly like this:

```
5 0 obj
<<
   /Type /StructElem
   /S /H1
   /P 4 0 R
   /K <</Type /MCR /MCID 0>>
>>
endobj
```

This is a structure element of the type `/H1` (and so a sectioning element) and it has one kid element, the text of the section marked above.

Beside this a number of additional settings and objects must be added to the pdf for cross-referencing and "administration".

3 Changing LaTeX

Measured in computer time, LaTeX is quite old. LaTeX is not only a format; it was always meant to be extended by packages and classes, and over time, many people have contributed to LaTeX. It has a quite large user base with *very* varied demands regarding stability, features and development. LaTeX is still used with a variety of engines: pdfTeX, XeTeX, LuaTeX, (u)pTeX and backends (dvips, dvipdfmx). One could compare LaTeX to an old city; lots of houses built at different times in different styles by various people, some modern, some older, some are in a good state, others are falling apart but nevertheless home to someone.

This means that changing LaTeX is not easy; we can't break lots of packages and old documents even if the reward is accessible pdfs. And we have to consider that documents must be compilable in TeX systems of varying age, for example when uploading them to a journal.

Thus, a very important aspect of the project is to develop a long term change strategy and manage integration of core support across the LaTeX universe.

4 First steps towards tagging

Tagging pdf with LaTeX has been on the agenda for quite some time. Babett Schalitz wrote a thesis about it in 2007, and Ross Moore has given a number of talks and articles at TUG conferences since then. When I considered working on the topic some time ago I got code from both, and decided rather quickly that first some work on the basics was needed.

Tagging should, in my opinion, not be done by creating a package that patches all sorts of commands in other packages; this is much too fragile. It needs proper support in the LaTeX kernel and proper support in the main classes and packages. I also thought that to identify the needed support and to test implementations and interfaces, concrete code was needed. So I wrote the package `tagpdf`. The package offers core commands to tag a document and to activate some of the other requirements needed to make a pdf accessible. The low-level code to mark up a text as a section looks roughly like this:

```
\tagstructbegin{tag=H1}
  \tagmcbegin{tag=H1}
    Section
  \tagmcend
\tagstructend
```

The `\tagstructXX` commands create the structure, while the `\tagmcXX` commands add the `MCID` marks to the page stream.

At present, the `tagpdf` package works with pdfLaTeX and LuaLaTeX, and LuaLaTeX gives the best results, as one doesn't have to worry about the behaviour at page breaks. However, with the help of the work on the `pdfresources` project described below, it should be possible to extend it to other engines and backends.

5 LaTeX-dev

Another important step towards accessible pdfs was the implementation of the latex-dev format by the LaTeX team and the maintainers of TeX Live and MiKTeX; latex-dev is a pre-release of LaTeX from the development branch and made available through CTAN. It allows users of a current TeX distribution to test their documents and code against an upcoming LaTeX release by simply using their preferred `latex` with the suffix `-dev`; `pdflatex-dev`, `lualatex-dev`, `xelatex-dev`, etc. (For more information, see [1].)

latex-dev has not been created solely with tagging in mind, but it will help us to coordinate and test changes with package and class authors, so it is an important part of the project.

6 PDF resource management

When tagging a pdf one has to add a number of settings to pdf dictionaries which can be described as "global resources". As already mentioned in an answer [2], LaTeX has no interfaces for this:

> Unhappily, the LaTeX format has overslept PDF development quite entirely. Managing global resources is the prime task for an OS, format in TeX speak. Because of the missing resource manager, both [`tikz` and `transparent`] packages do what most packages do, they think they are alone and add their stuff to the resource, . . .

With tagging entering the scene it was clear that something needed to be done to remedy this problem and so the `pdfresources` project in the LaTeX github was created; it contains a (still quite experimental) `expl3`-style interface which offers commands to add contents to pdf resources in a controlled way. It also offers backend-independent interfaces to a number of core commands needed when writing objects to a pdf. The package works with the main engines (pdfTeX, LuaTeX and XeTeX) and backends (dvipdfmx and — more or less — dvips).

The main task for the next months is to test the code, to integrate it into the kernel and to adapt existing packages to use it. The number of packages which should use the pdf resource manager is not very large but includes important packages such as `hyperref`, `tikz`, `media9`, `pdfx`.

7 Adapting the engines

Another open issue that emerged during the last year was missing functionality in engines and backends. For example pdfTeX was not ready for pdf 2.0; it has no command to set a major pdf version. (This will be remedied in TeX Live 2020.) As pdf 2.0 adds important features needed for accessibility (the concept of associated files) this is clearly something that should be changed. It would be also useful if pdfTeX could execute code at shipout time as can be done with luatex with `\latelua`. The dvipdfmx backend and dvips are missing additional color stacks.

8 Adding hooks

As already shown in sections 2 and 4, tagging a pdf requires adding quite a number of commands. Obviously, all the standard structures should if possible add the needed code automatically. For this, hooks are needed at the right places. The "right place" has firstly a technical meaning; with the exception of LuaTEX, the tagging code inserts *whatsits*; this means it can change the output if used in the wrong place (as sometimes anchors set by `hyperref` do).

But more importantly the "right place" means that we need to identify the *owner* of the code which should insert the tagging code. For example, sections are generally created with `\@startsection`. So this kernel command looks like a natural place to insert hooks for tagging commands. On the other hand, chapters and parts have special commands created by the classes. Does it make sense if the kernel handles the one part and the classes the other? Other examples are bibliographies and glossaries; packages like `biblatex` and `glossaries` look like the natural owner here — and both packages already have lots of hooks which make it easy to implement tagging — but both also use standard structures like lists or tabulars and additions to these generic environments could clash with their needs.

This means that besides a pdf resource manager we also need a hook management. And we need lots of real use cases and examples to be able to investigate the various dependencies.

9 Mathematics

How to tag maths is still an open problem. There are quite a number of possibilities to make it accessible.

- One is to attach the LATEX source code, either as file or verbatim, with `/ActualText` to the math structure. For a number of environments this can be automated quite well as the `axessibility` package demonstrates (but it is difficult for inline math input with `$...$`). The usability with a screen reader is not bad — even if not every word was correctly read aloud in my tests — but it requires that the user understands LATEX input syntax, and with large equations and complicated grouping it can be quite difficult to follow and to navigate through subequations. Usability can be improved if one invests the time to manually split the math and add explaining words.
- Another possibility is to mark all the maths bits with MathML structure names. At least with LuaTEX this can probably be done more or less automatically — proofs of concept are the ConTEXt format and TEX4ht. But it is unknown whether screen readers or other applications can actually use the information.
- A third possibility is to convert the equation to MathML, for example with MathJax, and attach it as an associated file to the structure. But here too it is unclear how such MathML can be processed by the pdf consumer. It is also unknown whether the presentation or content flavour of MathML should be used in this case.

The pdf standard requires that glyphs and symbols are mapped to Unicode. Here too variants are possible; for example, a could be mapped to U+1D44E (Mathematical Italic Small A) or U+0061 (Latin Small Letter A); $\int$ could mapped to U+222B (Integral) or to `\int` (as is done by the package `mmap`). The first alternative sounds more Unicode-like but actually the screen readers don't seem to know what to do with the symbols.

The main task here is to get more information to be able to decide about which route to follow.

10 Contacts

Quite a number of questions and projects circle around the pdf specification, the needs of users and of pdf consumer applications. To get tagging working it is not enough to know how TEX works. So one important part of the tagging project is to get in contact with people having inside knowledge about pdf and pdf consumer applications in various pdf related organizations and to promote the project in the TEX world to get user feedback.

11 Summary

Adding tagging facilities to LATEX is a large project with many aspects. Happily it doesn't have to be done in one large jump; with the `tagpdf` package it is already possible for adventurous users with a bit of knowledge in TEX programming to tag quite large documents. Despite the clear warning in the documentation that it isn't meant for production, I have already received feedback about several successful uses. This gives hope that it can evolve to a stable and usable system.

References

[1] LATEX. LATEX development formats are now available, 2019. `https://latex-project.org/news/2019/09/01/LaTeX-dev-format/`

[2] H. Oberdiek. Ti*k*Z and `transparent` incompatibility, 2015. `https://tex.stackexchange.com/a/253417/2388`

[3] TUG. PDF accessibility and PDF standards, 2019. `https://tug.org/twg/accessibility/`

◇ Ulrike Fischer
LATEX Project
Mönchengladbach, Germany
`ulrike.fischer (at) latex-project.org`

LaTeX3 News

Issue 11, February 2018

Contents

Move of sources from Subversion to Git

The LaTeX team have used a variety of version control systems over the life of the LaTeX3 sources. For a long time we maintained the LaTeX3 sources in Subversion (`svn`) but also provided a read-only clone of them on GitHub using SubGit from TMate Software [1] to synchronize the two repositories—a solution that worked very well.

We have now retired the Subversion repository and completely moved over to Git, with the master LaTeX3 repository hosted on GitHub: `https://github.com/latex3/latex3`. This new approach means we are (slowly) adopting some new approaches to development, for example branches and accepting pull requests.

Version identifiers

Following this change, we have removed Subversion `$Id` lines from the LaTeX3 sources. At present, we will be retaining `\GetIdInfo` as there are several possible use cases. The LaTeX3 sources now have only release date strings as identifiers. However, the team recommend that package authors include version information directly in `\ProvidesExplPackage` (or similar) lines.

expl3 updates and extensions

Work has continued on the codebase over the last year, with both small changes/fixes and more substantial changes taking place. The following sections summarise some of the more notable changes.

l3sort moves to the kernel

Sorting is an important ability, and for some time the team have provided a stand-alone l3sort to support this. The functionality has seen wide take up, and so has now been integrated directly into the kernel. This took place in parallel with some interface changes to "round out" the code.

Boolean functions

For some time, the team have been aware that boolean expressions can fail in certain circumstances, leading to low-level errors. This is linked to two features of the long-standing `\bool_if:n(TF)` function: expandable operation and short-circuit evaluation.

Addressing that has meant two changes: altering `\bool_if:n(TF)` to *always* evaluate each part of the expression, and introducing new short-circuit functions without the issue. The latter are `lazy` in expl3 terms:

- `\bool_lazy_all:n(TF)`
- `\bool_lazy_and:nn(TF)`
- `\bool_lazy_any:n(TF)`
- `\bool_lazy_or:nn(TF)`

These new, stable functions are now the recommended way of handling boolean evaluations. Package authors are encouraged to employ these new functions as appropriate.

Revision of l3file

Large parts of l3file have been revised to give a better separation of path/file/extension. This has resulted in the addition of a number of new support functions and variables.

At the same time, new experimental functions have been added to utilise a number of useful primitives in pdfTeX: `\file_get_mdfive_hash:nN`, `\file_get_size:nN` and `\file_get_timestamp:nN`. Currently, XeTeX does not support getting file size/timestamp information: this is available in other engines.

Paralleling these changes, we have added (experimental) support for shell escape to the l3sys module, most notably `\sys_shell_now:n`. A range of test booleans are also available to check whether shell escape is enabled.

Detection of `\cs_generate_variant:Nn` errors

The ability to generate variants is an important feature of expl3. At the same time, there are crucial aspects of this approach that can be misunderstood by users. In particular, the requirement that variants map correctly to an underlying `N`- or `n`-type base function is sometimes misunderstood.

To help detect and correct these cases, `\cs_generate_variant:Nn` now carries out error checking on its arguments, and raises a warning where it is mis-applied. At present, the team have avoided making this an error as it is likely to be seen by end users rather than directly by package developers. In time, we are likely to revisit this and tighten up further on this key requirement.

Accessing random data

To support randomised data selection, we have introduced a family of experimental functions which use underlying engine support for random values, and provide one entry at random from the data type.

At the same time, we have addressed some issues with uniformity stemming from the random number function used by pdfTEX and inherited by other engines. This means that expl3's FPU will generate *pseudo*-random values across the range of possible outputs.

More powerful debugging

A new set of debugging functions have been added to the kernel. These allow debug code to be enabled locally using the new option `enable-debug` along with functions `\debug_on:n` and `\debug_off:n`. Accompanying this change, we have improved the handling of global/local consistency in variable setting.

Mark-up changes in l3doc

Since the introduction of the `__` syntax to mark internal functions, the need for explicit markup of internal material in sources has been negated. As such, we have now dropped the requirement to mark internal material with `[aux]` when using l3doc. Instead, the status of functions and variables is auto-detected from the presence of `__`. For cases where non-standard names are used for internal code, the mark-up `[int]` is retained, *e.g.*

```
\begin{macro}[int]{\l@expl@enable@debug@bool}
```

l3build updates

Work on l3build has continued in parallel with expl3 work, in particular continuing to develop features to allow wider use of the tool.

Paralleling the move of the LATEX3 codebase to Git, l3build now has its own separate Git repository: `https://github.com/latex3/l3build`. This will enable us to involve other developers in the Lua code required for the build system. At the same time, we have split the code into a number of small source files, again to ease development both for the team ourselves and for potential collaborators.

Another major change is that l3build can now retain the structure of source repositories when creating a CTAN archive. Whilst the team favor 'flat' source setups, other users prefer structured approaches. Most notably, this new l3build functionality means that it is now used to carry out beamer releases.

The other major new feature is a new approach to multiple test setups, which replaces the older `--testfiledir` option. In the new approach, separate configuration files are listed in the main `build.lua` script, and can be selected manually using a new `--config` switch. This new approach allows complex test setups to be run in a totally automated fashion, which is important for kernel testing.

Some changes to the normalisation routines have been carried out, some to deal with upcoming LuaTEX changes, others to address aspects which show up only in some tests. This has required `.tlg` updates in some cases: as far as possible, we strive to avoid requiring changes to the reference files.

References

[1] *SubGit*, TMate Software, `https://subgit.com`

[2] Links to various publications by members of the LATEX Project Team.
`https://www.latex-project.org/publications`

LaTeX3 News

Issue 12, January 2020

Contents

Introduction

There has been quite a gap since the last *LaTeX3 News* (Issue 11, February 2018), and so there is quite a bit to cover here. Luckily, one of the things there *is* to cover is that we are using a more formalised approach for logging changes, so writing up what has happened is a bit easier. (By mistake LaTeX3 News 11 itself did not get *published* when written, but is now available: we have kept the information it contains separate as it is a good summary of the work that had happened in 2017.)

Work has continued apace across the LaTeX3 codebase in the last (nearly) two years. A lot of this is ultimately focussed on making the core of expl3 even more stable: *squeezing* out more experimental ideas, refining ones we have and making it a serious option for core LaTeX programming.

As a result of these activities, the LaTeX3 programming layer will be available as part of the kernel of LaTeX 2_ε from 2020-02-02 onwards, i.e., can be used without explicitly loading expl3. See *LaTeX News 31* [2] for more details on this.

New features in expl3

A new argument specifier: e-type

During 2018, the team worked with the TeX Live, XeTeX and (u)pTeX developers to add the `\expanded` primitive to pdfTeX, XeTeX and (u)pTeX. This primitive was originally suggested for pdfTeX v1.50 (never released), and was present in LuaTeX from the start of that project.

Adding `\expanded` lets us create a new argument specifier: `e`-type expansion. This is *almost* the same as `x`-type, but is itself expandable. (It also doesn't need doubled `#` tokens.) That's incredibly useful for creating function-like macros: you can ensure that *everything* is expanded in an argument before you go near it, with not an `\expandafter` in sight.

New functions

New programming tools have appeared in various places across expl3. The highlights are

- Shuffling of sequences to allow randomization
- Arrays of integers and floating point values; these have constant-time access
- Functions to return values after system shell usage
- Expandable access to file information, including file size, MD5 hash and modification date

For the latter, we have revised handling of file names considerably. There is now support for finding files in expansion contexts (by using the `\(pdf)filesize` primitive). Spaces and quotes in file names are now fully normalised, in a similar manner to the approach used by the latest LaTeX 2_ε kernel.

String conversion moves to expl3

In addition to entirely new functions, the team have moved the l3str-convert module from the l3experimental bundle into the expl3 core. This module is essential for dealing with the need to produce UTF-16 and UTF-32 strings in some contexts, and also offers built-in escape for url and PDF strings.

Case changing of text

Within expl3, the team have renamed and reworked the ideas from `\tl_upper_case:n` and so on, creating a new module l3text. This is a “final” home for functions to manipulate *text*; token lists that can reasonably be expected to expand to plain text plus limited markup, for example emphasis and labels/references. Moving these functions, we have also made a small number of changes in other modules to give consistent names to functions: see the change log for full details.

Over time, we anticipate that functions for other textual manipulation will be added to this module.

Notable fixes and changes

File name parsing

The functions for parsing file names have been entirely rewritten, partly as this is required for the expandable access to file information mentioned above. The new code correctly deals with spaces and quote marks in file names and splits the path/name/extension.

Message formatting

The format of messages in expl3 was originally quite text-heavy, the idea being that they would stand out in the `.log` file. However, this made them hard to find by a regular expression search, and was very different from the LaTeX 2_ε message approach. The formatting of expl3 messages has been aligned with that from the LaTeX 2_ε kernel, such that IDE scripts and similar will be able to find and extract them directly.

Key inheritance

A number of changes have been made to the inheritance code for keys, to allow inheritance to work “as expected” in (almost) all cases.

Floating point juxtaposition

Implicit multiplication by juxtaposition, such as `2pi`, is now handled separately from parenthetic values. Thus for example `1in/1cm` is treated as equal to `(1in)/(1cm)` and thus yields `2.54`, and `1/2(pi+pi)` is equal to `pi`.

Changing box dimensions

TeX’s handling of boxes is subtly different from other registers, and this shows up in particular when you want to resize a box. To bring treatment of boxes, or rather the grouping behavior of boxes, into line with other registers, we have made some internal changes to how functions such as `\box_set_wd:N` are implemented. This will be transparent for “well-behaved” use cases of these functions.

More functions moved to stable

A large number of functions which were introduced as candidates have been evaluated and moved to stable status. The team hopes to move all functions in expl3 to stable status, or move them out of the core, over the coming months.

Deprecations

There have been two notable sets of deprecations over the past 18 months. First, we have rationalised all of the “raw” primitive names to the form `\tex_<name>:D`. This means that the older names, starting `\pdftex_...`, `\xetex_...`, etc., have been removed.

Secondly, the use of integer constants, which dates back to the earliest days of expl3, is today more likely to make the code harder to read than anything else. Speed improvements in engines mean that the tiny enhancements in reading such constants are no longer required. Thus for example `\c_two` is deprecated in favour of simply using `2`.

In parallel with this, a number of older `.sty` files have been removed. These older files provided legacy stubs for files which have now been integrated in the expl3 core. They have now had sufficient time to allow users to update their code.

Internal improvements

Cross-module functions

The team introduced the idea of internal module functions some time ago. Within the kernel, there are places where functions need to be used in multiple modules. To make the nature of the kernel interactions clearer, we have worked on several aspects

- Reducing as far as possible cross-module functions
- Making more generally-useful functions public, for example scan marks
- Creating an explicit cross-kernel naming convention for functions which are internal but are essential to use in multiple kernel modules

The backend

Creating graphics, working with color, setting up hyperlinks and so on require backend-specific code. Here, backends are for example `dvips`, `xdvipdfmx` and the direct PDF mode in pdfTeX and LuaTeX. These functions are needed across the LaTeX3 codebase and have to be updated separately from the expl3 core. To facilitate that, we have split those sources into a separate bundle, which can be updated as required.

At the same time, the code these files contain is very low-level and is best described as internal. We have re-structured how the entire set of functions are referred to such that they are now internal for the area they implement, for example image inclusion, box affine transformations, etc.

Better support for (u)pTeX

The developers behind (u)pTeX (Japanese TeX) have recently enhanced their English documentation (see `https://github.com/texjporg/ptex-manual`). Using this new information, we have been able to make internal adjustments to expl3 to better support these engines.

Options

A new option `undo-recent-deprecations` is now available for cases where a document (or package) requires some expl3 functions that have been formally removed after deprecation. This is to allow *temporary* work-arounds for documents to be compiled whilst code is begin updated.

The “classical” options for selecting backends (`dvips`, `pdftex`, etc.) are now recognised in addition to the native key–value versions. This should make it much easier to use the expl3 image and color support as it is brought up to fully-workable standards.

Engine requirements

The minimum engine versions needed to use expl3 have been incremented a little:

- pdfTEX v1.40
- XƎTEX v0.99992
- LuaTEX v0.95
- ε-(u)pTEX mid-2012

The team have also worked with the XƎTEX and (u)pTEX developers to standardise the set of post-ε-TEX utility primitives that are available: the so-called “pdfTEX utilities”. These are now available in all supported engines, and in time will all be *required*. This primarily impacts XƎTEX, which gained most of these primitives in the 2019 TEX Live cycle. (Examples are the random number primitives and expandable file data provision.) See *LATEX News 31* [2] for more.

Documentation

News

The *LATEX3 News* files were until recently only used to create PDF files on the team website [1]. We have now integrated those into the l3kernel (expl3 core) bundle. The news files cover all of LATEX3 files, as the core files are always available.

ChangeLog

Since the start of 2018, the team have commenced a comprehensive change log for each of the bundles which make up the LATEX3 code. These are simple Markdown text files, which means that they can be displayed formatted in web views.

Changes in xparse

A number of new features have been added to xparse. To allow handling of the fact that skipping spaces may be required only in some cases when searching for optional arguments, a new modifier `!` is available in argument specifiers. This causes xparse to *require* that an optional argument follows immediately with no intervening spaces.

There is a new argument type purely for environments: `b`-type for collecting a `\begin...\end` pair, i.e., collecting the body of an environment. This is similar in concept to the environ package, but is integrated directly into xparse.

Finally, it is now possible to refer to one argument as the default for another optional one, for example

```
\NewDocumentCommand{\caption}{O{#2} m} ...
```

New experimental modules

A number of new experimental modules have been added within the l3experimental bundle:

l3benchmark Performance-testing system using the timing function in modern TEX engines

l3cctab Category code tables for all engines, not just LuaTEX

l3color Color support, similar in interface to xcolor

l3draw Creation of drawings, inspired by pgf, but using the LATEX3 FPU for calculations

l3pdf Support for PDF features such as compression, hyperlinks, etc.

l3sys-shell Shell escape functions for file manipulation

l3build changes

The l3build tool for testing and releasing TEX packages has seen a number of incremental improvements. It is now available directly as a script in TEX Live and MiKTEX, meaning you can call it simply as

`l3build` *target*

Accompanying this, we have added support for installing scripts and script `man` files.

There is a new `upload` target that can take a zip file and send it to CTAN: you just have to fill in release information for *this* upload at the prompts.

Testing using PDF files rather than logs has been heavily revised: this is vital for work on PDF tagging.

There is also better support for complex directory structures, including the ability to manually specify TDS location for all installed files. This is particularly targeted at packages with both generic and format-specific files to install.

References

[1] *LATEX Project Website.* `https://latex-project.org/`

[2] *LATEX 2ε release newsletters on the LATEX Project Website.* `https://latex-project.org/news/latex2e-news/`

LaTeX News

Issue 31, February 2020

Contents

Experiences with the LaTeX `-dev` formats

As reported in the previous *LaTeX News*, we have made a pre-release version of the LaTeX kernel available as LaTeX`-dev`. Overall, the approach of having an explicit testing release has been positive: it is now readily available in TeX systems and is getting real use beyond the team.

The current release has been tested by a number of people, and we have had valuable feedback on a range of the new ideas. This has allowed us to fix issues in several of the new features, as described below.

We wish to thank all the dedicated users who have been trying out the development formats, and we encourage others to do so. Pre-testing in this way does mean that, for the vast majority of users, problems are solved before they even appear!

Concerning this release . . . (LuaLaTeX engine)

The new LuaHBTeX engine is LuaTeX with an embedded HarfBuzz library. HarfBuzz can be used by setting a suitable renderer in the font declaration. A basic interface for that is provided by fontspec. This additional font renderer will greatly improve the shaping of various scripts when using LuaLaTeX, many of which are currently handled correctly only by XeTeX, which always uses HarfBuzz.

To simplify testing of the new engine, binaries have already been added to MiKTeX and TeX Live 2019 and both distributions have already now changed the LuaLaTeX-dev format to use it.

Going forward, LuaLaTeX (and LuaLaTeX-dev) will both use the LuaHBTeX engine. The timing of the switch to the LuaHBTeX engine depends on the distribution you use (for TeX Live this will be with TeX Live 2020).

Improved load-times for expl3

The LaTeX3 programming layer, expl3, has over the past decade moved from being largely experimental to broadly stable. It is now used in a significant number of third-party packages, most notably xparse, for defining interfaces in cases where no expl3 code is "visible". In addition, most LaTeX documents compiled using XeTeX or LuaTeX load fontspec, which is written using expl3.

The expl3 layer contains a non-trivial number of macros, and when used with the XeTeX and LuaTeX engines, it also loads a large body of Unicode data. This means that even on a fast computer, there is a relatively large load time when using expl3.

For this release, the team have made adjustments in the LaTeX 2$_\varepsilon$ kernel to pre-load a significant portion of expl3 when the format is built. This is transparent to the user, other than the significant decrease in document processing time: there will be no "pause" whilst loading the Unicode data files. Loading expl3 in documents and packages can continue to be done as usual; eventually, it will be possible to omit

```
\RequirePackage{expl3}
```

entirely but, to support older formats, this is still recommended at present.

Improvements to LaTeX's font selection mechanism (NFSS)

Extending the shape management in NFSS

Over time, more and more fonts have become available for use with LaTeX. Many such font families offer additional shapes such as small caps italic (`scit`), small caps slanted (`scsl`) or swash (`sw`). By using `\fontshape` those shapes can be explicitly selected. For the swash shapes there is also `\swshape` and `\textsw` available.

In the original font selection implementation a request to select a new shape always overrode the current shape. With the 2020 release of LaTeX this has changed and `\fontshape` can now be used to combine small capitals with italic, slanted or swash letters, either by explicitly asking for `scit`, etc., or by asking for italics when typesetting already in small caps, and so forth.

Using `\upshape` will still change italics or slanted back to an upright shape but will not any longer alter the small caps setting. To change small capitals back to upper/lower case you can now use `\ulcshape` (or `\textulc`) which in turn will not change the font with respect to italics, slanted or swash. There is one exception: for compatibility reasons `\upshape` will change small capitals back to upright (`n` shape), if the current shape is `sc`. This is done so that something like `\scshape...\upshape` continues to work as before, but we suggest that you don't use that deprecated method in new documents.

Finally, if you want to reset the shape back to normal you can use `\normalshape` which is a shorthand for `\upshape\ulcshape`.

The way that shapes combine with each other is not hardwired; it is customizable and extensible if there is ever a need for this. The mappings are defined through `\DeclareFontShapeChangeRule` and the details for developers are documented in `source2e.pdf`.

The ideas for this interface extension have been pioneered in fontspec by Will Robertson for Unicode engines, and in fontaxes by Andreas Bühmann and Michael Ummels for pdfTeX; they are by now used in many font support packages.

Extending the font series management in NFSS

Many of the newer font families also come provided with additional weights (thin, semi-bold, ultra-bold, etc.) or several running widths, such as condensed or extra-condensed. In some cases the number of different values for series (weight plus width) is really impressive: for example, Noto Sans offers 36 fonts, from ultra-light extra condensed to ultra-bold medium width.

Already in its original design, NFSS supported 9 weight levels, from ultra-light (`ul`) to ultra-bold (`ub`), and also 9 width levels, from ultra-condensed (`uc`) to ultra-expanded (`ux`): more than enough, even for a font family like Noto Sans. Unfortunately, some font support packages nevertheless invented their own names, so in recent years you have been able to find all kinds of non-standard series names (`k`, `i`, `j` and others), making it impossible to combine different fonts successfully using the standard NFSS mechanisms.

Over the course of the last year a small number of individuals, notably, Bob Tennent, Michael Sharpe and Marc Penninga, have worked hard to bring this unsatisfactory situation back under control; so today we are happy to report that the internal font support files for more than a hundred font families are all back to following the standard NFSS conventions. Combining them is now again rather nice and easy, and from a technical perspective they can now be easily matched; but, of course, there is still the task of choosing combinations that visually work well together.

In the original font selection implementation, a request to select a new series always overrode the current one. This was reasonable because there were nearly no fonts available that offered anything other than a medium or a bold series. Now that this has changed and families such as Noto Sans are available, combining weight and width into a single attribute is no longer appropriate. With the 2020 release of LaTeX, the management of series therefore changed to allow independent settings of the weight and the width attributes of the series.

For most users this change will be largely transparent as LaTeX offers only `\textbf` or `\bfseries` to select a bolder face (and `\textmd` and `\mdseries` to return to a medium series): there is no high-level command for selecting a condensed face, etc. However, using the NFSS low-level interface it is now possible to ask for, say, `\fontseries{c}\selectfont` to get a condensed face (suitable for a marginal note) and that would still allow the use of `\textbf` inside the note, which would select a bold-condensed face (and not a rather odd-looking bold-extended face in the middle of condensed type).

The expectation is that this functionality will be used largely by class and package designers but, given that the low-level NFSS commands are usable on the document level and that they are not really difficult to apply, there are probably also a number of users who will enjoy using these new possibilities that bring LaTeX back into the premier league for font usage.

The ways in which the different series values combine with each other is not hardwired but is again customizable and extensible. The mappings are defined through `\DeclareFontSeriesChangeRule` and the details for developers are documented in `source2e.pdf`.

Font series defaults per document family

With additional weights and widths now being available in many font families, it is more likely that somebody will want to match, say, a medium weight serif family with a semi-light sans serif family, or that with one family one wants to use the bold-extended face when `\textbf` is used, while with another it should be bold (not extended) or semibold, etc.

In the past this kind of extension was provided by Bob Tennent's mweights package, which has been used in many font support packages. With the 2020 release of LaTeX this feature is now available out of the box. In addition we also offer a document-level interface to adjust the behavior of the high-level series commands `\textbf`, `\textmd`, and of their declaration forms `\bfseries` and `\mdseries`, so that they can have different effects for the serif, sans serif and typewriter families used in a document.

For example, specifying

```
\DeclareFontSeriesDefault[rm]{bf}{sb}
\DeclareFontSeriesDefault[tt]{md}{lc}
```

in the document preamble would result in `\textbf` producing semi-bold (`sb`) when typesetting in a roman typeface. The second line says that the typewriter default face (i.e., the medium series `md`) should be a light-condensed face. The optional argument here can be either `rm`, `sf` or `tt` to indicate one of the three main font families in a document; if omitted you will change the overall document default instead. In the first mandatory argument you specify either `md` or `bf` and the second mandatory argument then gives the desired series value in NFSS nomenclature.

Handling of nested emphasis

In previous releases of LaTeX, nested `\emph` commands automatically alternated between italics and upright. This mechanism has now been generalised so that you can now specify for arbitrary nesting levels how emphasis should be handled.

The declaration `\DeclareEmphSequence` expects a comma separated list of font declarations corresponding to increasing levels of emphasis. For example,

```
\DeclareEmphSequence{\itshape,%
              \upshape\scshape,\itshape}
```

uses italics for the first, small capitals for the second, and italic small capitals for the third level (provided you use a font that supports these shapes). If there are more nesting levels than provided, LaTeX uses the declarations stored in `\emreset` (by default `\ulcshape\upshape`) for the next level and then restarts the list.

The mechanism tries to be "smart" by verifying that the given declarations actually alter the current font. If not, it continues and tries the next level—the assumption being that there was already a manual font change in the document to the font that is now supposed to be used for emphasis. Of course, this only works if the declarations in the list's entries actually change the font and not, for example, just the color. In such a scenario one has to add `\emforce` to the entry, which directs the mechanism to use the entry, even if the font attributes appear to be unchanged.

Providing font family substitutions

Given that pdfTeX can only handle fonts with up to 256 glyphs, a single font encoding can only support a few languages. The `T1` encoding, for example, does support many Latin-based scripts, but if you want to write in Greek or Cyrillic then you will need to switch encodings to `LGR` or `T2A`. Given that not every font family offers glyphs in such encodings, you may end up with some default family (e.g., Computer Modern) that doesn't blend in well with the chosen document font. For such cases NFSS now offers `\DeclareFontFamilySubstitution`, for example:

```
\DeclareFontFamilySubstitution{LGR}
      {Montserrat-LF}{IBMPlexSans-TLF}
```

tells LaTeX that if you are typesetting in the sans serif font `Montserrat-LF` and the Greek encoding `LGR` is asked for, then LaTeX should use `IBMPlexSans-TLF` to fulfill the encoding request.

The code is based on ideas from the substitutefont package by Günter Milde, but the implementation is different.

Providing all text companion symbols by default

The text companion encoding `TS1` was originally not available by default, but only when the textcomp package was loaded. The main reason for this was limited availability of fonts with this encoding other than Computer Modern; another was the memory restrictions back in the nineties. These days neither limitation remains, so with the 2020 release all the symbols provided with the textcomp package are available out of the box.

Furthermore, an intelligent substitution mechanism has been implemented so that glyphs missing in some fonts are automatically substituted with default glyphs that are sans serif if you typeset in `\textsf` and monospaced if you typeset using `\texttt`. In the past they were always taken from Computer Modern Roman if substitution was necessary.

This is most noticeable with `\oldstylenums` which are now taken from `TS1` so that you no longer get 1234 but 1234 when typesetting in sans serif fonts and `1234 when using typewriter fonts.`

If there ever is a need to use the original (inferior) definition, then that remains available as `\legacyoldstylenums`; and to fully revert to the old behavior there is also `\UseLegacyTextSymbols`. The latter declaration reverts `\oldstylenums` and also changes the footnote symbols, such as `\textdagger`, `\textparagraph`, etc., to pick up their glyphs from the math fonts instead of the current text font (this means they always keep the same shape and do not nicely blend in with the text font).

With the text companion symbols as part of the kernel, it is normally no longer necessary to load the `textcomp` package, but for backwards compatibility this package will remain available. There is, however, one use case where it remains useful: if you load the package with the option `error` or `warn` then substitutions will change their behavior and result in a LaTeX error or a LaTeX warning (on the terminal), respectively. Without the package the substitution information only appears in the `.log` file. If you use the option `quiet`, then even the information in the transcript is suppressed (which is not really recommended).

New `alias` size function for use in `.fd` files

Most of the newer fonts supported in TeX have been set up with the `autoinst` tool by Marc Penninga. In the past, this program set up each font using the face name chosen by that font's designer, e.g., "`regular`", "`bold`", etc. These face names were then mapped by substitution to the standard NFSS series names, i.e., "`m`" or "`b`". As a result one got unnecessary substitution warnings such as "`Font T1/abc/bold/n not found, using T1/abc/b/n instead`".

We now provide a new NFSS size function, `alias`, that can and will be used by `autoinst` in the future. It provides the same functionality as the `subst` function but is less vocal about its actions, so that only significant font substitutions show up as warnings.

Suppress unnecessary font substitution warnings

Many sans serif fonts do not have real italics but usually only oblique/slanted shapes, so the substitution of slanted for italics is natural and in fact many designers talk about italic sans serif faces even if in reality they are oblique.

With nearly all sans serif font families, the LaTeX support files therefore silently substitute slanted if you ask for `\itshape` or `\textit`. This is also true for Computer Modern in `T1` encoding but in `OT1` you got a warning on the terminal even though there is nothing you can do about it. This has now been changed to an information message only, written to the `.log` file. *(github issue 172)*

Other changes to the LaTeX kernel

UTF-8 characters in package descriptions

In 2018 we made UTF-8 the default input encoding for LaTeX but we overlooked the case of non-ASCII characters in the short package descriptions used in declarations, e.g., in the optional argument to `\ProvidesPackage`. They worked (sometimes) before, but the switch to UTF-8 made them always generate an error. This has been corrected. *(github issue 52)*

Fix inconsistent hook setting when loading packages

As part of loading a package, the command `\`*package*`.sty-h@@k` gets defined. However, attempting to load a package a second time resulted in this hook becoming undefined again. Now the hook remains defined so that extra loading attempts do not change the state of LaTeX (relevant only to package developers). *(github issue 198)*

Avoid spurious warning if `LY1` is made the default encoding

Making `LY1` the default encoding, as is done by some font support packages, gave a spurious warning even if `\rmdefault` was changed first. This was corrected. *(github issue 199)*

Ensure that `\\` remains robust

In the last release we made most document-level commands robust, but `\\` became fragile again whenever `\raggedright` or similar typesetting was used. This has been fixed. *(github issue 203)*

Make math delimiters robust in a different way

Making math delimiters robust caused an issue in some situations. This has been corrected. This also involved a correction to `amsmath`. *(github issue 251)*

Allow more write streams with `filecontents` in LuaTeX

Most TeX engines only support a maximum of sixteen concurrently open write streams, and when those have been used up, then `filecontents` or any other code trying to open one will fail. In LuaTeX more write streams are available and those can also now be utilised. *(github issue 238)*

Allow spaces in `filecontents` option list

Leaving spaces or newlines in the option list prevented the options from being correctly recognized. This has been corrected. *(github issue 256)*

New `reverselist` Lua callback type

A new callback type, `reverselist`, was added: `post_mlist_to_hlist_filter` and `post_linebreak_filter` are now of this type.

Changes to packages in the graphics category

Make color & graphics user-level commands robust

Some of the user-level commands in color, graphics and graphicx, such as `\textcolor` or `\includegraphics`, were still fragile so didn't work in moving arguments. All of these are now robust. *(github issue 208)*

Changes to packages in the tools category

Fixed column depth in boxed `multicols`

The `multicols` environment was setting `\maxdepth` when splitting boxes; but, due to the way the internal interfaces of LaTeX are designed, it should have used `\@maxdepth` instead. As a result, balanced boxed multicols sometimes ended up having different heights even if they had exactly the same content. *(github issue 190)*

Ensure that `multicols` does not lose text

The `multicols` environment needs a set of consecutively numbered boxes to collect column material. The way those got allocated could result in disaster if other packages allocated most boxes below box 255 (which TeX always uses for the output page). In the original implementation that problem was avoided because one could only allocate box numbers below 255, but nowadays the LaTeX allocation routine allows allocating box numbers both below and above 255. So the assumption that when asking for, say, 20 boxes you always get a consecutive sequence of 20 box register numbers became no longer true: some of the column material could end up in box 255, where it would get overwritten. This has now been corrected by allocating all necessary boxes with numbers above 255 whenever there aren't enough lower-numbered registers available. *(github issue 237)*

Allow spaces in `\hhline` arguments

The `\hhline` command, which allows the specification of rule segments in `tabular` environments, now allows (but ignores) spaces between its tokens: so `\hhline{: = : =}` is now allowed and is equivalent to `\hhline{:=:=}`. This matches similar token arguments in LaTeX such as the `[h t p]` argument on floats. A similar change has been made to the extended `\hhline` command in the `colortbl` package. *(github issue 242)*

LaTeX requirements on engine primitives

Since the finalization of ε-TeX in 1999, a number of additional 'utility' primitives have been added to pdfTeX. Several of these are broadly useful and have been required by expl3 for some time, most notably `\pdfstrcmp`. Over time, a common set of these 'post-ε-TeX' primitives have been incorporated into XeTeX and (u)p-TeX; they were already available in LuaTeX.

A number of these additional primitives are needed to support new or improved functionality in LaTeX. This is seen for example in the improved UTF-8 handling, which uses `\ifincsname`. The following primitive functionality (which in LuaTeX may be achieved using Lua code) will therefore be *required* by the LaTeX kernel and core packages from the start of 2021:

- `\expanded`
- `\ifincsname`
- `\ifpdfprimitive`
- `\pdfcreationdate`
- `\pdfelapsedtime`
- `\pdffiledump`
- `\pdffilemoddate`
- `\pdffilesize`
- `\pdflastxpos`
- `\pdflastypos`
- `\pdfmdfivesum`
- `\pdfnormaldeviate`
- `\pdfpageheight`
- `\pdfpagewidth`
- `\pdfprimitive`
- `\pdfrandomseed`
- `\pdfresettimer`
- `\pdfsavepos`
- `\pdfsetrandomseed`
- `\pdfshellescape`
- `\pdfstrcmp`
- `\pdfuniformdeviate`

For ease of reference, these primitives will be referred to as the 'pdfTeX utilities'. With the exception of `\expanded`, these have been present in pdfTeX since the release of version 1.40.0 in 2007; `\expanded` was added for TeX Live 2019. Similarly, the full set of these utility primitives has been available in XeTeX from the 2019 TeX Live release, and has always been available in LuaTeX (some by Lua emulation). The Japanese pTeX and upTeX gained all of the above (except `\ifincsname`) for TeX Live 2019 and will both have that primitive also from the 2020 release onward.

At the same time, engines which are fully Unicode-capable must provide the following three primitives:

- `\Uchar`
- `\Ucharcat`
- `\Umathcode`

Note that it has become standard practice to check for Unicode-aware engines by using the existence of the `\Umathcode` primitive. As such, this is already a requirement: engines lacking these primitives cannot use Unicode features of the LaTeX 2ε kernel or expl3. Note also that upTeX can handle Unicode but it is not classed as a Unicode engine by the base LaTeX code.

References

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

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

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

Case changing: From TEX primitives to the Unicode algorithm

Joseph Wright

1 Introduction

The concept of letter case is well established for several alphabet-based scripts, most notably Latin, Greek and Cyrillic. Upper- and lowercase[1] are so widely used that it may not be obvious that there are several subtleties in converting case. However, those subtleties are important in supporting a wide range of users, and getting all of them right is non-trivial.

Whilst the English alphabet has simple case-changing rules, when we look beyond English and (possibly) beyond the Latin alphabet, tracking the requirements becomes more complicated. Many of these have been codified by the Unicode Consortium, and following these guidelines means that different pieces of software can give consistent outcomes.

Here, I want to look at how case changing can be set up in TEX, primarily focussing on tools that the LATEX Project have provided, but in the wider context of the TEX ecosystem.

2 Different kinds of case operation

To understand what functionality is needed for case changing, we first have to know what types of input we might be dealing with. Broadly, there are two:

- Text: material that we will want to typeset or similar, and which contains natural language content. This material might also have some formatting, and may be marked up as being in a particular language.
- Strings: material used in code, for example as identifiers, to construct control sequences or to find files. This material will never have formatting, and should always give the same outcome, irrespective of the language in which a document is written.

Unsurprisingly, case changing strings is a lot more straight-forward than case changing text: there is no context to worry about. However, neither text nor string case changing is reversible, and that leads us to the different types of case changing operations that are needed.

The Unicode Consortium describe four case operations, three case mappings and one case folding:

- Uppercasing
- Lowercasing
- Titlecasing: changing the first character to uppercase and the remaining characters to lowercase — we will see later how a small number of situations need special handling
- Case folding: removing case information from the input to allow 'caseless' operations

(I strongly recommend the Unicode Consortium's FAQ at `https://unicode.org/faq/casemap_charprop.html` for more on the concepts here.)

Upper-, lower- and titlecasing material is about *text*: material to be read by people and which can have context- and language-dependence. In contrast, case folding is about computers: doing things mechanically for comparing internal information. Commonly, programmers use all-lowercase for that, but there are places where that would be wrong: again, we'll see some examples below. We'll also see that for TEX use, there are places we need 'programmer's upper- and lowercase (a.k.a. CamelCase) for strings.

3 Changing tokens or changing output

Before we look at the methods we can use in TEX to change case, it's worth bearing in mind that for *typesetting*, there's the possibility of leaving any change in appearance to the font mechanisms. Whilst this is complicated in classical TEX, LuaTEX offers the potential to delegate the task to well after character tokens have been handed to the paragraph-builder.

However, it's quite natural in many programming languages to change the case of text (referred to in many languages as *strings*). As such, I will focus on methods that can alter the characters in the TEX input stream.

4 Built-in TEX mechanisms

In the TEX world, the primitives `\lowercase` and `\uppercase` are the obvious starting point for case changing. These primitives use data stored in TEX using the `\lccode` and `\uccode` primitives, respectively, and only ever perform context-independent conversion. Whilst there are important uses for this behaviour well beyond 'normal' case changing, I am going to focus here only on their utility (or otherwise) for application to text.

The most obvious limitation of the primitives is that they assume a single mapping for all characters. Whilst there are a large number of simple relationships, there are exceptions: see the Unicode data file `SpecialCasing.txt` for the full list! The simple approach taken by the primitives means that there is no chance of handling context-dependence: this is most obvious with Greek, where there are two forms of lowercase sigma, one used only at the end of words (ς).

[1] The spelling of these concepts is somewhat variable: 'upper case', 'upper-case' and 'uppercase' are all valid. The LATEX Project have chosen the latter form as it makes creating clearly-named code functions easier!

At the TeX level, the other issue with the primitives is that they work by execution not expansion. I think almost every trainee TeX programmer will at some stage have tried

```
\edef\foo{\lowercase{STUFF}}
```

and been very surprised that it fails, and that they need to use

```
\lowercase{\edef\foo{STUFF}}
```

instead. This shows up when you want to change case inside a `\csname` construct too: you can't use `\lowercase` within the construction, but rather have to use it around the entire thing.

```
\lowercase{\csname FooBar\endcsname}
```

There's also one other very important consideration: the primitives convert character tokens only, but do not know the meaning of these tokens. This leads to a few issues:

- They do not convert text hidden in macros: relatively easily remedied by applying `\edef` (or `\protected@edef`) prior to using the primitive.
- They cannot handle letter-like control sequences such as `\aa` or `\l`.
- They cannot handle multi-byte 'letters' in 8-bit engines (so for example `\uppercase{é}` fails).
- They provide no mechanism for excluding characters from case changing, most critically those inside math mode.

5 The LaTeX 2ε kernel mechanism

The LaTeX 2ε kernel builds on the primitives to address some of the issues above. First off, these commands include an internal `\protected@edef`, which ensures that input is expanded first, then the case change is applied. They also provide definitions for a set of letter-like commands to allow them to be case-changed, stored as `\@uclclist`. This for example allows `\aa` to be uppercased to `\AA`.

6 The textcase mechanism

Case changing is fundamentally something that applies to *text*, and never to *mathematics*. In LaTeX, it is pretty clear which content is mathematical, and David Carlisle's textcase package makes it possible to case change text containing math mode content without 'breaking' the latter. Thus for example

```
\MakeTextUppercase
  {A simple formula:
    $y = mx + c$}
```

yields

A SIMPLE FORMULA: $y = mx + c$

The package also allows text to be marked as not to be altered during case changing, using the marker command `\NoCaseChange`.

It's possible to load textcase such that it replaces the LaTeX 2ε kernel case-changing commands with its own. That gives away that it works in a fundamentally similar way: it's a more sophisticated wrapper around the TeX primitives. That means it still has the core issues of not knowing the meaning or context of its argument tokens, and not being usable in an expansion context.

7 The expl3 mechanisms

7.1 The core concepts

At the core of the expl3 case changing mechanisms is the idea that the implementation should provide, as far as possible, the full set of Unicode Consortium functionality. The code is also written to work purely by expansion, meaning that it *can* be used inside `\csname` construction or inside an `\edef`.

To support all this, the input must be examined on a token-by-token basis and converted 'manually'. It also means that `\lowercase` and `\uppercase` cannot be used. Instead, for single-token conversion, expandable functions which can convert single tokens are defined: `\char_lowercase:N`, `\char_uppercase:N`, `\char_titlecase:N` and `\char_foldcase:N`. Almost always, those are too low-level. Thus we will not look further at the 'back end', but will rather concentrate on functions which can be used for longer pieces of input.

7.2 Strings

In TeX terms, a string is a series of characters which are all treated as 'other' tokens (except spaces, which are still spaces). That's important here because it means strings won't contain any control sequences, and because with pdfTeX there can't be any (useful) accented characters.

The most obvious need to handle case in programming strings is when comparing in a caseless manner: 'removing' the case. Programmers often do that by lowercasing text, but there are places where that's not right. For example, as mentioned above, Greek has two forms of the lowercase sigma (σ and ς), and these should be treated as the same for a caseless test. Unicode defines the correct operation: case *folding*. In expl3, that's called `\str_foldcase:n`:

```
\str_foldcase:n { AbC }
```

gives

abc

whilst the slightly more challenging

```
\str_foldcase:n { ὈΔΥΣΣΈΥΣ }
```

gives

ὀδυσσεύσ

Much more rare is the need to upper- or lowercase a string. Unicode does not mention this at all, but in TEX we might want to construct a control sequence dynamically. To do that, we might want to uppercase the first character of some user input *string*, and lowercase the rest. We can do that by combining `\str_uppercase:n` and `\str_lowercase:n` with the `\str_head:n` and `\str_tail:n` functions:

```
\str_uppercase:f { \str_head:n { someThing } }
\str_lowercase:f { \str_tail:n { someThing } }
```

which produces

Something

(We could also split off the first token and use the single-character `\char_uppercase:N` here.)

7.3 Text: basics

Case changing text is much more complicated because it has to deal with control sequences, accents, math mode and context. The first step of case changing here is to expand the input as far as possible: that's done using a function called `\text_expand:n` which works very similarly to the LATEX 2$_\varepsilon$ command `\protected@edef`, but is *expandable*. We don't really need to worry too much about this: it's built into the case changing system anyway.

Upper- and lowercasing is straight-forward: the functions have the natural names `\text_uppercase:n` and `\text_lowercase:n`. These deal correctly with things like the Greek final-sigma rule and (with LuaTEX and XƎTEX) cover the full Unicode range. Thus we can have examples such as the following. (Recall that spaces are ignored in expl3 input, and ~ is used to produce a space.)

```
\text_lowercase:n { Some~simple~English }
\newline
\text_uppercase:n { Ragıp~Hulûsi~Özdem }
\newline
\text_lowercase:n { ὈΔΥΣΣΕΥΣ }
```

some simple english
RAGIP HULÛSI ÖZDEM
ὀδυσσεύς

A variety of standard LATEX accents and letter-like commands are set up for correct case changing with no user intervention required.

```
\text_uppercase:n { \aa{}ngstr\"{o}m ~ caf\'{e} }
```

produces the token list

```
\AA{}NGSTR\"{O}M CAF\'{E}
```

7.4 Case changing exceptions

There are places that case changing should not apply, most obviously to math mode material. There are a set of exceptions built-in to the case changer, and that list can be extended: it's easy to add the equivalent of `\NoCaseChange` from the textcase package. First, create and activate the command, excluding it from expansion and excluding its argument from case-changing:

```
\cs_new_protected:Npn \NoCaseChange #1 {#1}
\tl_put_right:Nn
  \l_text_case_exclude_arg_tl
  { \NoCaseChange }
```

then we can use it

```
\text_uppercase:n
  { Hello ~ $y = max + c$ }
\newline
\text_lowercase:n
  { \NoCaseChange { iPhone } ~ iPhone }
```

which gives us the desired results

HELLO $y = max + c$
iPhone iphone

without having to worry further. Note that case changing does take place within braces (in contrast to BIBTEX's approach):

```
\text_uppercase:n { { Text } ~ More }
```

gives

TEXT MORE

The reason is simple: braces are difficult to control and to remove, and can lead to undesirable impact on kerning and so on. (It is likely that a dedicated conversion function to approximate BIBTEX case protection by expl3 protection will be added; the biblatex maintainers are keen to have this ability.)

7.5 Titlecasing

Commonly, people think about uppercasing the first character of some text then lowercasing the rest, for example to use it at the start of a sentence. Unicode describes this operation as titlecasing, as there are some situations where the 'first character' is handled in a non-standard way. Perhaps the best example is IJ in Dutch: it's treated as a single 'letter', so *both* letters have to be uppercase at the start of a sentence. There are also a small set of codepoints that *look* like two letters, and have special forms when they appear as the titlecase first-character of a word.

Depending on the exact nature of the input, we might want to uppercase the first 'character' and then lowercase everything else, or we might want to uppercase the first 'character' and then leave everything else unchanged. These are called `\text_titlecase:n` and `\text_titlecase_first:n`, respectively. Most of the time, things look pretty simple:

```
\text_titlecase:n { some~text } \newline
\text_titlecase:n { SOME~TEXT } \newline
\text_titlecase_first:n { some~text } \newline
\text_titlecase_first:n { SOME~TEXT }
```

gives

Some text
Some text
Some text
SOME TEXT

To see titlecasing in action, let's stick with Dutch (we'll see how the language argument works shortly):

```
\text_titlecase:nn { nl } { IJSSELMEER }
```

should be

IJsselmeer

and not what most English speakers might expect

Ijsselmeer

As we are not simply grabbing the first token of the input, non-letters are ignored and the first real text character is case-changed. So when we say

```
\text_titlecase:n { `some~text' }
```

we of course want the result to be

'Some text'

without needing to worry about that first character.

7.6 Language-dependent functions

One important context for case changing text is the language the text is written in: there are special considerations for Dutch, Lithuanian, Turkic languages and Greek. That's all handled by using versions of the case-changing functions that take a second argument: a BCP 47 string which can determine the path taken. We've already seen Dutch, so let's examine the other special situations.

We've seen Greek for handling the final-sigma rule, but we also need to remove accents when uppercasing (but *not* when titlecasing). Sticking with the hero of myth

```
\text_uppercase:n { Ὀδυσσεύς } \newline
\text_titlecase:n { Ὀδυσσεύς } \newline
\text_uppercase:nn { el } { Ὀδυσσεύς } \newline
\text_titlecase:nn { el } { Ὀδυσσεύς }
```

gives us the expected

ὈΔΥΣΣΈΥΣ
Ὀδυσσεύς
ΟΔΥΣΣΕΥΣ
Ὀδυσσεύς

For Turkish, it's all about the dot over an `i`: that's a different letter from the dotless ı, and so

```
\text_uppercase:n { Ragıp~Hulûsi~Özdem }
\newline
\text_uppercase:nn { tr } { Ragıp~Hulûsi~Özdem }
```

produces

RAGIP HULÛSI ÖZDEM
RAGIP HULÛSİ ÖZDEM

Finally, Lithuanian needs us to retain dots on `i` and `j` when we add certain accents

```
\text_lowercase:n
  { ÌÍĨÌÍĨJ̀J́J̃Į̀Į́Į̃ }
\newline
\text_lowercase:nn { lt }
  { ÌÍĨÌÍĨJ̀J́J̃Į̀Į́Į̃ }
```

produces (in `\large` for better visibility)

ìíĩìíĩj̀j́j̃į̀į́į̃
i̇̀i̇́i̇̃ i̇̀i̇́i̇̃ j̇̀j̇́j̇̃ į̇̀į̇́į̇̃

with explicit 'more above' dot accents (and yes, it does look odd here, but that's a font shaper decision).

There's one more special case: the capital *Eszett* in German. Normally, capitalisation of `ß` (U+00DF) yields simply `SS` (two characters), but there is now a Unicode codepoint (U+1E9E) for the capital form, `ẞ` (that's a capital Eszett in the Courier Narrow font we're using in this article; it looks nearly the same as the lowercase). There is currently no official BCP 47 name for this (that I know of), so one needs to use

```
\text_uppercase:nn { de-alt } { ß }
```

8 The Lua route

There is another way to do case changing in expansion contexts: using Lua.

```
\directlua{tex.print(unicode.utf8.lower
  ("A string"))}
```

That works, but it leaves us with a few issues. For a start, anything that depends on TEX tokens, like math mode, needs more work. Secondly, it doesn't deal with things like Greek final-sigma. That's all solvable with enough Lua, but it's not (yet) an out-of-the-box solution.

9 Conclusions

Implementing all of the subtleties of case-changing following the Unicode Consortium requires effort, but is accessible and can be done by pure expansion. Case-changing functions in expl3 are now mature and stable, and ready for wider use.

◇ Joseph Wright
Northampton, United Kingdom
joseph dot wright (at) morningstar2.co.uk

TeX, LaTeX and math

Enrico Gregorio

Abstract

We discuss some aspects of mathematical typesetting: choice of symbols, code abstraction, fine details. Relationships between math typesetting and international standards are examined. A final section on typesetting of numbers and units reports on some recent developments in the field.

1 Introduction

Readers may well know that TeX was born out of Knuth's discomfort after having seen the proofs of the new edition of the first volume of his *magnum opus* "The Art of Computer Programming".

Many papers have been written by Knuth himself and by others on the topic of math typesetting. Here I'd like to present some personal ideas on the subject, coming from almost thirty years of experience in mathematical typesetting. I'll also present some recent developments and new tricks made available with expl3.

2 A very short lead-in to math in TeX

Every TeX guru knows that TeX is (almost) always in one of three *modes*:

- horizontal mode,
- vertical mode,
- math mode.

Except for circumstances when TeX is in no mode at all (when writing to external files, for the curious).

Each mode comes into two flavors, but here we are interested only in math mode. Knuth calls the two flavors 'math mode' and 'display math mode'. In order to better distinguish between them, I'll call the former 'inline math mode', so the unadorned 'math mode' will denote both.

There are subtle, well, not so much so, differences between the two flavors; beginners are most impressed by $\sum_{k=1}^{n} k^2 = \frac{1}{3}n\left(n+\frac{1}{2}\right)(n+1)$ that suddenly becomes

$$\sum_{k=1}^{n} k^2 = \frac{1}{3}n\left(n+\frac{1}{2}\right)(n+1)$$

when displayed and a very common question is 'how do I get the limits above and below the summation symbol and real fractions, not that smallish replacement symbol?'

Like all of us, I've been a beginner myself; I discovered `\limits` and abused it. *Penitenziagite*, would have said Salvatore in "Il nome della rosa" (Umberto Eco's novel; the English title is "The name of the Rose"). Now I'm no longer a beginner and *know* why `\limits` should not be used; and let's not talk about the dreaded `\displaystyle` that is sometimes suggested to newbies. The proper way is just `\sum`.

To the contrary, beginners are usually much less impressed by the wrong typesetting in

$$A\backslash B = \{x|x\in A, x\notin B\}$$

but they are likely to shrug and move on, if they ever note it. Sometimes they see something's wrong and 'fix' the vertical bar by using `\,|\,`, that's still wrong. Why is it wrong? The spacing is too small, of course, but there's more to the problem: two appearances of such a construction in the same document is a sin similar to what I describe to young basketball referees: "whoever calls a double foul during their career has called one too many". The correct answer is: first of all define a macro for the object, for instance,

```
\newcommand{\suchthat}{\,|\,}
```

(I'm talking LaTeX, plain TeX users can translate). In case one asks, if `a+b` appears twice or more in a document there's no need to make a macro out of it; the separator in the set builder notation is a single conceptual object and so it *must* be typed by a single command.

About the spacing, one should realize that the reverse bar is a binary operation symbol and the vertical bar is a relation symbol. Both are already defined in all flavors of TeX and they are, respectively, `\setminus` and `\mid`, but it's still convenient and logically sound to define `\suchthat`, because `\mid` is a 'generic' name:

```
\newcommand{\suchthat}{\mid}
...
A \setminus B=\{x \suchthat
  x\in A, x\notin B\}
```

will typeset as

$$A \setminus B = \{x \mid x\in A, x\notin B\}$$

while this is the version with the thin spaces:

$$A \setminus B = \{x \,|\, x\in A, x\notin B\}$$

Compare closely the spaces around the vertical bar.

I'm not saying the last realization should be rejected as awfully wrong: personal judgment is always welcome when typography is concerned, after having studied the alternatives and common practice. Above all, consistency throughout a document is a must. I had to edit a paper where the separator was a bar or a colon or a semicolon, depending on which

First published in *ArsTeXnica* 28 (Oct 2019), pp. 47–57. Reprinted with permission.

of the three authors had typed the formula. Defining `\suchthat` allows for delaying any decision about what symbol to use until the last minute. More on set builder notation later.

The TeXbook lists several symbol names, some have semantics attached to them, like `\setminus`, and others don't, like `\mid` or `\otimes`.[1] Why is that? Some symbols have essentially a single use case, others appear in different branches of mathematics with different meanings. Everybody loves `\lhd` and `\unlhd`, right? The symbols typeset as $\lhd$ and $\unlhd$ respectively. I believe to have seen once what the names should suggest, but I forgot it. The symbols are common in group theory, where they denote 'normal subgroup': it's heartily recommended to group theorists to define a meaningful command for them. Oh, I was almost forgetting! Those are not defined as relation symbols in LaTeX, so a savvy group theorist will type in the document preamble

```
% normal subgroup
\newcommand{\ns}{\mathrel\lhd}
\newcommand{\nseq}{\mathrel\unlhd}
% subnormal subgroup
\newcommand{\sns}{\ns\ns}
```

The symbols are not among the core ones designed by Knuth. They first appeared in a symbol font distributed along with LaTeX; possibly Lamport used them for his own papers as binary operators and the classification stuck. They were later included in amssymb (`\vartriangleleft`, which is a relation).

What should an author do? The case of normal subgroups is clear: I surely wouldn't litter my paper with `\mathrel\lhd` each time I want to mention normal subgroups. However, suppose a paper frequently uses Euler's *totient function*, which has the well established tradition of being denoted by φ (the open version of phi). Is it better to use `\varphi` or to define `\euphi`? The latter. Imagine that upon receiving proofs, the author realizes that *all* instances of `\varphi` print out ϕ, because the publisher uses a font that lacks the proper symbol. With `\euphi` it is a matter of doing a redefinition, probably borrowing the open phi from another font. We don't know when the instruction `\let\varphi=\phi` is performed, but using `\euphi` makes this irrelevant.

An important exception: in the abstract there should be *no* use of personal macros. It should be able to typeset with a 'naked' version of LaTeX: it's very common nowadays that the abstract is fed to some web page that maybe uses MathML, MathJax or similar device for handing the text to browsers.

[1] Generally LaTeX kept the same names.

Going back to the normal subgroup symbol, one should know that every math symbol belongs to a class and there are seven of them:

- class 0, ordinary symbols;
- class 1, operators;
- class 2, binary operations;
- class 3, binary relations;
- class 4, opening symbols;
- class 5, closing symbols;
- class 6, punctuation.

TeX will set the spacing between symbols according to well defined rules. This is not the place to discuss them fully, see [4]. Any object, as long as it is legal in math mode, can be defined to behave as if it belongs in one of the above classes by typing it as the argument to

```
\mathord \mathop \mathbin \mathrel
\mathopen \mathclose \mathpunct
```

For instance, the symbol for the determinant is internally carried out (in LaTeX) by something like

```
\mathop{\operator@font det}\nolimits
```

but there is a higher level interface available for declaring new symbols like this; for instance, one does

```
\DeclareMathOperator{\adj}{adj}
```

in order to introduce a symbol for the adjugate matrix. A one-shot operator can be typeset in the document by

```
\operatorname{adj}
```

The *-version of both commands makes for a symbol that carries limits above and below in display math mode, on the side when inline.

The unfortunately common perversion of denoting open intervals like $]a,b[$ needs input such as

```
\mathopen]a,b\mathclose[
```

One can easily spot that something is wrong when just using `]a,b[` by looking at the difference between the two instances below

$$x \in]a,b[\qquad x \in \mathopen]a,b\mathclose[$$

In my calculus notes I type `\interval[o]{a,b}`, so I can decide to be a perv by just changing a few lines in the definition. An open interval will be typeset as $(a\mathinner{\ldotp\ldotp}b)$, but I'm not bound in any way: I can go back to the comma again by just changing a line. Also, I like to write upper unbounded intervals like $(a\mathinner{\ldotp\ldotp}\to)$, but I use `\pinf` for the arrow, so I can make it to be typeset ∞ by acting on a single line, should I change my mind.

Upon entering math mode, TeX will construct a *math list* consisting of *math atoms*, each of which

$$
\begin{cases}
a^6+2a^3b^3+b^6=q^2\\
4a^3b^3=-\dfrac{4}{27}p^3
\end{cases}
\qquad
\left\{\begin{aligned}
a^6+2a^3b^3+b^6&=q^2\\
4a^3b^3&=-\frac{4}{27}p^3
\end{aligned}\right.
\qquad
\left\{\begin{gathered}
a^6+2a^3b^3+b^6=q^2\\
4a^3b^3=-\frac{4}{27}p^3
\end{gathered}\right.
$$

$$
\begin{cases}
a^3+b^3=-q\\
a^6-2a^3b^3+b^6=q^2+\dfrac{4}{27}p^3
\end{cases}
\qquad
\left\{\begin{aligned}
a^3+b^3&=-q\\
a^6-2a^3b^3+b^6&=q^2+\frac{4}{27}p^3
\end{aligned}\right.
\qquad
\left\{\begin{gathered}
a^3+b^3=-q\\
a^6-2a^3b^3+b^6=q^2+\frac{4}{27}p^3
\end{gathered}\right.
$$

$$
\begin{cases}
a^3+b^3=-q\\
(a^3-b^3)^2=q^2+\dfrac{4}{27}p^3
\end{cases}
\qquad
\left\{\begin{aligned}
a^3+b^3&=-q\\
(a^3-b^3)^2&=q^2+\frac{4}{27}p^3
\end{aligned}\right.
\qquad
\left\{\begin{gathered}
a^3+b^3=-q\\
(a^3-b^3)^2=q^2+\frac{4}{27}p^3
\end{gathered}\right.
$$

$$
\begin{cases}
a^3+b^3=-q\\
a^3-b^3=\sqrt{q^2+\dfrac{4}{27}p^3}
\end{cases}
\qquad
\left\{\begin{aligned}
a^3+b^3&=-q\\
a^3-b^3&=\sqrt{q^2+\frac{4}{27}p^3}
\end{aligned}\right.
\qquad
\left\{\begin{gathered}
a^3+b^3=-q\\
a^3-b^3=\sqrt{q^2+\frac{4}{27}p^3}
\end{gathered}\right.
$$

Figure 1: Three ways of laying out the derivation of Cardano's formula

$$
\left\{\begin{aligned}
a^6+2a^3b^3+b^6&=q^2\\
4a^3b^3&=-\frac{4}{27}p^3
\end{aligned}\right.
$$
$$
\left\{\begin{aligned}
a^3+b^3&=-q\\
a^6-2a^3b^3+b^6&=q^2+\frac{4}{27}p^3
\end{aligned}\right.
$$
$$
\left\{\begin{aligned}
a^3+b^3&=-q\\
(a^3-b^3)^2&=q^2+\frac{4}{27}p^3
\end{aligned}\right.
$$
$$
\begin{cases}
a^3+b^3=-q\\
a^3-b^3=\sqrt{q^2+\dfrac{4}{27}p^3}
\end{cases}
$$

Figure 2: One of the worst alignments I can conceive

has a *nucleus*, a *subscript field* and a *superscript* field. When exiting from math mode, the math list will be transformed into a horizontal list according to the (complex) rules described in Appendix G of *The TEXbook*. These rules add spaces, as said before, but also take care of the bidimensionality of math formulas: superscripts, subscripts, fractions, accents, radicals, extensible delimiters and many more aspects.

Had Knuth been into theoretical physics, he probably would have added also "prescripts" for isotopes and staggered multiple subscripts and superscripts for tensors. Unfortunately he hasn't. See later for more on this topic.

3 Fine points of mathematics typing

The title is the same as chapter 18 in *The TEXbook*. Of course I won't go through Knuth's words. Since I'm talking LATEX and math, I assume that amsmath is loaded: no serious math typesetting can be done without it.

A point that's not touched upon in the TEXbook is 'when, really, consecutive equations should be aligned and where'. Browsing TEX.StackExchange reveals several examples of bad alignments.

A prominent example is a derivation of Cardano's formula[2] which I won't give the code for, but just three realizations that you can see in figure 1.

I often use the style "the good, the bad, and the ugly". There is actually an even uglier way, which is what the questioner was asking for, see figure 2.

What's the problem? The equals signs are not really related to one another. The *pairs of formulas* are related, but the fact that they are equations is almost irrelevant. Mixing ragged right and ragged left in one and the same paragraph (or display) makes for very hard reading. I'd instead be more generous with vertical spacing between the various braces and I have no doubt whatsoever that the leftmost realization is our Clint Eastwood. Look for holes in the typeset output and remove them.

Another example can be seen in figure 3.[3] You can judge for yourself which is the best way to present the display. My opinion is that the equals signs in the second column pair are not related to each other, so they're not to be aligned.

Linear systems are an exception, because their matrix-like structure is more important than holes. I recommend the wonderful systeme package by Christian Tellechea [11]. No doubt there are other exceptions: typography, and mathematical typography in particular, is a craft that doesn't obey mechanical rules. A thin space may open up symbols and make them easier to read, adding a pair of parentheses may clear up an ambiguity, removing unnecessary

[2] `https://tex.stackexchange.com/questions/193581`
[3] `https://tex.stackexchange.com/questions/500472`

$$\begin{pmatrix}A_\mu\\ \rho^*_\mu\end{pmatrix} \to \begin{pmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{pmatrix}\begin{pmatrix}A_\mu\\ \rho^*_\mu\end{pmatrix}, \qquad \tan\theta = \frac{g_{el}}{g_*} \tag{1}$$

$$\begin{pmatrix}\psi_L\\ \chi_L\end{pmatrix} \to \begin{pmatrix}\cos\varphi_{\psi_L} & -\sin\varphi_{\psi_L}\\ \sin\varphi_{\psi_L} & \cos\varphi_{\psi_L}\end{pmatrix}\begin{pmatrix}\psi_L\\ \chi_L\end{pmatrix}, \qquad \tan\varphi_{\psi_L} = \frac{\Delta}{m} \tag{2}$$

$$\begin{pmatrix}\tilde\psi_R\\ \tilde\chi_R\end{pmatrix} \to \begin{pmatrix}\cos\varphi_{\tilde\psi_R} & -\sin\varphi_{\tilde\psi_R}\\ \sin\varphi_{\tilde\psi_R} & \cos\varphi_{\tilde\psi_R}\end{pmatrix}\begin{pmatrix}\tilde\psi_R\\ \tilde\chi_R\end{pmatrix}, \qquad \tan\varphi_{\tilde\psi_R} = \frac{\tilde\Delta}{\tilde m} \tag{3}$$

$$\begin{pmatrix}A_\mu\\ \rho^*_\mu\end{pmatrix} \to \begin{pmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{pmatrix}\begin{pmatrix}A_\mu\\ \rho^*_\mu\end{pmatrix}, \qquad \tan\theta = \frac{g_{el}}{g_*} \tag{4}$$

$$\begin{pmatrix}\psi_L\\ \chi_L\end{pmatrix} \to \begin{pmatrix}\cos\varphi_{\psi_L} & -\sin\varphi_{\psi_L}\\ \sin\varphi_{\psi_L} & \cos\varphi_{\psi_L}\end{pmatrix}\begin{pmatrix}\psi_L\\ \chi_L\end{pmatrix}, \qquad \tan\varphi_{\psi_L} = \frac{\Delta}{m} \tag{5}$$

$$\begin{pmatrix}\tilde\psi_R\\ \tilde\chi_R\end{pmatrix} \to \begin{pmatrix}\cos\varphi_{\tilde\psi_R} & -\sin\varphi_{\tilde\psi_R}\\ \sin\varphi_{\tilde\psi_R} & \cos\varphi_{\tilde\psi_R}\end{pmatrix}\begin{pmatrix}\tilde\psi_R\\ \tilde\chi_R\end{pmatrix}, \qquad \tan\varphi_{\tilde\psi_R} = \frac{\tilde\Delta}{\tilde m} \tag{6}$$

Figure 3: Two similar alignments

parentheses may improve the quality of a formula. Compare top and bottom line

$$a\frac{f(x+h)-f(x)}{h}+b\frac{g(x+h)-g(x)}{h}$$
$$a\,\frac{f(x+h)-f(x)}{h}+b\,\frac{g(x+h)-g(x)}{h}$$

and decide which one looks better. In my notes I used the bottom one when working the proof of linearity of the derivative. If I talk about "the function $g(z) = \sqrt{z-1}\,$", I add a thin space before ending inline math mode:

```
‘‘the function $g(z)=\sqrt{z-1}\,$’’
```

in order to avoid the clash between the vinculum and the quotes in "the function $g(z) = \sqrt{z-1}$". Try with a parenthesis after the radical to see another case: $(1+\sqrt{2})^{-1}$ versus $(1+\sqrt{2}\,)^{-1}$. In the latter case a thin space has been added.

Going to *very* fine details: does anybody notice the differences below? Consider the formulas

$$\log|x| \neq \log\lvert x\rvert \tag{7}$$
$$|\sin x| \neq \lvert\sin x\rvert \tag{8}$$
$$\|\operatorname{adj} A\| \neq \lVert\operatorname{adj} A\rVert \tag{9}$$

where the questionable typesetting is on the left. While the top left *could* be a typographic choice (so long as it is consistent), the other formulas in the left-hand sides are definitely wrong.

The mathtools package provides a very good facility for handling these cases, namely

```
\DeclarePairedDelimiter{\abs}{|}{|}
```

that allows to type `\abs{\sin x}` and forget about the dreaded thin space, which can also be avoided by

```
\lvert\sin x\rvert
```

Which style to choose is a matter of personal preference and habit. I recommend not to abuse the facility: reserve it for *functions* such as absolute values, norms and similar objects. Don't exploit it for parenthesized expressions: something like

```
\paren{a+b}\paren{a-b}=a^2-b^2
```

hinders input reading and would print the same as `(a+b)(a-b)=a^2-b^2`. True, one could do

```
\paren[\big]{a\paren{b+c}}
```

but is this really more legible than

```
\bigl( a (b + c) \bigr)
```

that keeps the usual mathematical structure? That is, assuming `\big` size is really necessary, which it isn't in the particular case.

Since I mentioned trigonometric functions, look at

$$\sqrt{\sin x}+\sqrt{\cos x}+\sqrt{\tan x}$$

and explain what's going wrong. Yes, the tittle makes the difference! It makes 'sin' higher than 'cos' and moves up the radical sign; similarly with 'tan'. In my trigonometry notes I have

```
\let\cos\undefined
\DeclareMathOperator{\cos}
  {cos\vphantom{i}}
\let\tan\undefined
\DeclareMathOperator{\tan}
  {tan\vphantom{i}}
```

with which the above formula would become

$$\sqrt{\sin x}+\sqrt{\cos\vphantom{i} x}+\sqrt{\tan\vphantom{i} x}$$

Radicals often need fine control in order to get them aligned with each other. Some appropriate trick involving `\vphantom` or `\smash` can fix things up:

$$\sqrt{x}+\sqrt{y} \neq \sqrt{x}+\sqrt{y}$$

Again, left is the questionable output; the formula on the right has been input as

```
\sqrt{x}+\sqrt{\smash[b]{y}}
```

The alternative

```
\sqrt{\mathstrut x}+\sqrt{\mathstrut y}
```

doesn't seem as attractive: $\sqrt{\mathstrut x}+\sqrt{\mathstrut y}$. Radicals would need a full chapter, so I'll stop here, except for one last thing: add a thin space when a radical is followed by a fence; similarly, add a thin space when a big operator (summation, product, integral) in display math mode is preceded by a fence and its limits are wide. Example:

$$\biggl(\sum_{k=1}^{n} a_k\biggr) \neq \biggl(\,\sum_{k=1}^{n} a_k\biggr)$$

4 Upright or italic?

Rivers of (electronic) ink have been spilled trying to answer the question. Actually it cannot be answered: mathematicians and engineers agree to disagree. Physicists disagree with each other.

Part of the question is: should constants be typeset in an upright font or not? The ISO 80000-2:2009 standard prescribes upright; adhering to this standard is mandatory in some technology and commercial fields. This is a good thing: people reading a technical report or manual will have no doubt about the meaning of a symbol.[4] While I strongly disagree with several decisions of the ISO standard, on mathematical grounds, I accept the underlying philosophy towards uniformity in the technical fields. Surely I appreciate its ban on the mathematically wrong $\sin^{-1}$ and similar: the standard has disputable aspects, but it's never wrong from a mathematical point of view.

On the other hand, many mathematicians are traditionalists and prefer italics for constants such as e (the Euler number) and i (the imaginary unit). Euler and Gauss used italics for the latter, and I'm among those who don't dare to challenge their authority. Of course, I know that mathematical notation has changed along time. I'd not use Cayley's original notation for matrices[5] because a better notation has developed. I follow the practice of setting standard function names in upright type (sine, cosine, logarithm and so on) even when ancient mathematicians didn't.

However, such decrees as 'symbols for vectors should be bold italic serif lowercase, for matrices should be bold italic serif uppercase, for tensors should be bold italic sans serif uppercase' make me smile: as a mathematician, I know that vectors, matrices and tensors are not different objects from a mathematical point of view. Matrices admit an easier two-dimensional representation: this is the 'big' difference.

For pedagogical reasons, I might use distinctive typesetting for vectors and matrices in a students' textbook. In a research paper or graduate level book I'd probably not make any distinction, if not mandated by clarity. In this case I'd explain the notation choices at the beginning of the paper or book.

A very fine book by J. Dieudonné [2], in the English edition by Academic Press, uses

- $\mathbf{R}$ or $\mathbf{C}$ for number sets,
- X for manifolds, E for vector bundles,
- A for vector space operators,
- $\mathrm{T}_x(\mathrm{X})$ or $\mathrm{T}_x(f)$ for the tangent space or linear mapping,
- $d_x\mathsf{f}$ or $d_x f$ for the differential at x of a mapping (vector valued or scalar valued),
- $\boldsymbol{\mathsf{Z}}$ for tensor fields,

and several other conventions that are consistently followed across the book and the series. The book starts off with a nine page long notation section. The same notation is used in the original French version.

However, it happens that book translations use different conventions from the original. It is the case for W. Rudin's 'Real and Complex Analysis' [9] where the differential 'd' is in italics, whereas it's upright in the Italian translation published by Bollati-Boringhieri [10]. I disagree with the publisher: maybe the editorial preference is for the upright 'd', but the author's style should be preserved as much as possible.

Not a big deal, one could think. No, this reflects on the *meaning* of the differential 'd'. There are several arguments in favor or against italics; my feeling is that most pure mathematicians prefer italics.

By the way, how to input the symbol in such a way that the convention can be changed at will? The simplest and more effective way is to define

```
\newcommand{\diff}{\mathop{}\!d}
```

(or `\mathrm{d}` if one *must* have the abomination).

I believe I learned this from Claudio Beccari through a `comp.text.tex` post. The code was credited to him in the paper [5],[6] but I'm not sure about the real source of this code pearl. Claudio Beccari had earlier proposed much more complicated code [1], namely

[4] A problem with ISO standards is that they have to be *bought*; the one we're talking about prices 158 CHF, about 143 € or \$160 at the current exchange rate.

[5] `https://tex.stackexchange.com/q/487643`

[6] The paper is also available in English [6].

```
\makeatletter
\providecommand*{\diff}{%
  \@ifnextchar^{\DIfF}{\DIfF^{}}%
}
\makeatother
\def\DIfF^#1{%
  \mathop{\mathrm{\mathstrut d}}%
  \nolimits^{#1}%
  \gobblespace
}
\def\gobblespace{%
  \futurelet\diffarg\opspace
}
\def\opspace{%
  \let\DiffSpace\!%
  \ifx\diffarg(%
    \let\DiffSpace\relax
  \else
    \ifx\diffarg[%
      \let\DiffSpace\relax
    \else
      \ifx\diffarg\{%
        \let\DiffSpace\relax
      \fi
    \fi
  \fi
  \DiffSpace
}
```

What's the idea in the complicated definition? Look whether a superscript follows; if it doesn't, add a dummy one. Well, this is already wrong, because it adds `\scriptspace` unconditionally. After that, the next token is examined: if it is a fence, then don't add `\!`, because a `\mathop` is followed by a fence with no thin space; in case an ordinary symbol follows, the `\mathop` would add a thin space, which is removed by `\!`. Well, try it with `\diff\bigl(x+y\bigr)`. Next try the simpler definition and see! Where's the trick? The *empty* `\mathop` is followed by an ordinary symbol, the 'd'; we just need to remove the excess thin space! The thin space preceding the empty `\mathop` is inserted automatically by TeX following the rules. Thus we can define

```
\newcommand{\tder}[2]
  {\frac{\diff #1}{\diff #2}}
```

without worrying that spurious spaces may creep in. Instead

```
\iint\limits_{D} f(x,y) \diff x \diff y
```

will typeset as needed

$$\iint\limits_{D} f(x,y)\,dx\,dy$$

For differential forms

$$f(x,y)\,dx\wedge dy$$

the spacing will be automatically right.

The same paper by Claudio [1] proposes commands for the constants, namely

```
% The number 'e'
\providecommand*{\eu}
  {\ensuremath{\mathrm{e}}}
% The imaginary unit
\providecommand*{\iu}
  {\ensuremath{\mathrm{j}}}
```

I strongly disagree with proposing `\ensuremath`; referring in the text to the Euler's number by

```
We use \eu\ to denote...
```

is by no means easier and clearer than

```
We use $\eu$ to denote...
```

One keystroke more? So what? That's a mathematical symbol so it ought to be typed in math mode, just like when we talk about the variable x. *My* definition would be

```
\newcommand{\eu}{\mathord{e}}
```

so typing `\eu` outside of math mode would raise an error. Change to `\mathrm` if you prefer upright type.

During the preparation of this paper, I examined the toptesi bundle, to find

```
\providecommand{\eu}{%
  \ensuremath{%
    {\mathop{\mathrm{e}}\nolimits}%
  }%
}
```

This is disputable in several respects:

- `\ensuremath` serves no real purpose;
- `\nolimits` can be safely omitted, because the `\mathop{...}` bit is followed by `}`, so surely there are no limits to take into account;
- `\mathop` itself is redundant, because the whole thing is braced, so it is treated as an ordinary symbol.

Oh, wait! No, `\mathop` is actually wrong! Consider the following code:

```
\documentclass{standalone}
\usepackage{amsmath}
\newcommand{\euA}{\mathrm{e}}
\newcommand{\euB}{%
  \ensuremath{%
    {\mathop{\mathrm{e}}\nolimits}%
  }%
}
\begin{document}
$2\euA\euB$
\end{document}
```

The output is shown in figure 4. Do you see the problem? A single character in the argument to `\mathop` is raised or lowered so that it extends the same distance above and below the math axis.

2ee

Figure 4: Magnified output for the Euler's constant problem

Now that we're on the spot, how to define a better `\tder` macro also supporting higher order derivatives? The first attempt,

```
\newcommand{\tder}[3][]{%
  \frac{\diff^{#1}#2}{\diff #3^{#1}}%
}
```

has a flaw: it unconditionally adds `\scriptspace` to both the numerator and denominator. If I measure the width of `\tder{f}{t}` in display math mode, with the standard fonts and document class, I get `14.07712pt`; the version without the dummy exponents has width `11.91045pt`. More than two points! With the upright 'd', the difference would be half a point. And the visual result shows more:

$$\frac{d^{}f}{dt^{}} \neq \frac{df}{dt} \qquad \frac{\mathrm{d}^{}f}{\mathrm{d}t^{}} \neq \frac{\mathrm{d}f}{\mathrm{d}t}$$

Yes, we need to avoid the dummy superscript, also with the upright 'd', although the difference is less noticeable: we want perfect output, don't we? And we want macros that allow users to choose their own preferred 'd'. One could test whether the argument is empty, but there's a better way with xparse:

```
\NewDocumentCommand{\tder}{s o m m}{%
  \IfBooleanTF{#1}{\dfrac}{\frac}%
    {\diff\IfValueT{#2}{^{#2}}#3}% num
    {\diff #4\IfValueT{#2}{^{#2}}}% den
}
```

The *-version delivers `\dfrac` (just in case one needs it), otherwise `\frac` is used. The numerator and the denominator add the exponent only if the optional argument is specifically used. Thus `\tder{f}{t}` will not add a dummy exponent.

5 Sets, bras and kets

A short note on the title. Physicists have a sense of humor: a well-established notation for inner products is $\langle x \mid y\rangle$, called a "bracket". A mathematician would denote the linear or semilinear forms induced by the bracket as $\langle x \mid -\rangle$ and $\langle - \mid y\rangle$. Physicists, instead, use $\langle x|$ for the former and $|y\rangle$ for the latter, calling them "bra" and "ket".

For several years, LaTeX has been requiring e-TeX extensions, among which `\middle` is a very useful one. For instance, we can typeset

$$\left\{x\in\mathbf{R} \;\middle|\; -\frac{1}{2}\le x\le \frac{8}{5}\right\}$$

with no phantom and no null delimiter. On the other hand, the code

```
\left\{x\in\mathbf{R} \;\middle|\;
  -\frac{1}{2}\le x\le \frac{8}{5}\right\}
```

is still really ugly and something like

```
\set*{x\in\mathbf{R}\suchthat
  -\frac{1}{2}\le x\le \frac{8}{5}}
```

would be much nicer. We call xparse and expl3 to the rescue!

```
\documentclass[varwidth]{standalone}
\usepackage{amsmath}
\usepackage{xparse}

\ExplSyntaxOn
\NewDocumentCommand{\set}{som}
 {
  % limit the scope for \suchthat
  \group_begin:
  \cs_set_protected:Npn \suchthat
   {
    \tl_use:N \l__egreg_set_st_tl
   }
  \IfBooleanTF{#1}
   {
    \egreg_set_auto:n { #3 }
   }
   {
    \egreg_set_fixed:nn { #2 } { #3 }
   }
  \group_end:
 }

\tl_new:N \l__egreg_set_st_tl

\cs_new_protected:Nn \__egreg_set_st:n
 {
  \tl_set:Nn \l__egreg_set_st_tl { #1 }
 }

\cs_new_protected:Nn \egreg_set_auto:n
 {
  \__egreg_set_st:n
   {
    \nonscript\;
    \middle\vert
    \nonscript\;
   }
  \left\{ #1 \right\}
 }

\cs_new_protected:Nn \egreg_set_fixed:nn
 {
  \tl_if_novalue:nTF { #1 }
```

```
   {
    \__egreg_set_st:n { \mid }
    \lbrace #2 \rbrace
   }
   {
    \__egreg_set_st:n
     { \mathrel{#1\vert} }
    \mathopen{#1\lbrace}
    #2
    \mathclose{#1\rbrace}
   }
 }
\ExplSyntaxOff

\begin{document}

$\set{a,b,c}\cup\set[\big]{a,b,c}$

$\set{x\suchthat a<x<b}$

$\set[\Big]{x\suchthat a<x<b}$

$\set*{x\suchthat \dfrac{1}{2}<x<3}$
\end{document}
```

The idea is to use a syntax familiar from mathtools' `\DeclarePairedDelimiter`. The output is in figure 5.

$$\{a,b,c\}\cup\big\{a,b,c\big\}$$
$$\{x\mid a<x<b\}$$
$$\Big\{x\;\Big|\;a<x<b\Big\}$$
$$\left\{x\;\middle|\;\frac{1}{2}<x<3\right\}$$

Figure 5: Examples of set notation

In *The TeXbook*, Knuth recommends to add thin spaces when the set builder notation contains a bar, that is, it is not just a list of elements. I disagree, but how could it be implemented? It's possible to look for the presence of `\suchthat` at the outer level and, in this case, to add the thin spaces at either end; nested sets would examine their own contents for the presence at the outer level.

A full implementation would also feature the choice for the delimiter as a preamble setting. I leave this as an exercise for whoever wants to make a package out of this code.

There is some duplication in the code below, but it's unavoidable. The reason is that using an `O{}` specifier for the optional argument would allow `\mathclose{#2\rbrace}` and no case distinction. However, one can see the difference if a subscript is added

`\rbrace_{1}`	`\mathclose{\rbrace}_{1}`
$\}_1$	$\mathclose{\}}_1$

Different coding is possible, though. It would not be difficult to allow `|` instead of `\suchthat`. Look at how the macros for bras and kets can be defined.

```
\documentclass[varwidth]{standalone}
\usepackage{amsmath}
\usepackage{xparse}

\NewDocumentCommand{\bra}{som}{%
  \IfBooleanTF{#1}
    {\left\langle #3 \right|}
    {%
     \IfNoValueTF{#2}
      {\langle#3\mathclose|}
      {\mathopen{#2\langle}#3\mathclose{#2|}}%
    }
}
\NewDocumentCommand{\ket}{som}{%
  \IfBooleanTF{#1}
    {\left| #3 \right\rangle}
   ,  {%
     \IfNoValueTF{#2}
      {\mathopen|#3\rangle}
      {\mathopen{#2|}#3\mathclose{#2\rangle}}%
    }
}

\NewDocumentCommand{\braket}{som}{%
  \IfBooleanTF{#1}
   {\extensiblebraket{#3}}
   {\fixedbraket{#2}{#3}}%
}

\ExplSyntaxOn
\NewDocumentCommand{\extensiblebraket}{m}
 {
  \group_begin:
  \char_set_active_eq:nN { `| } \egreg_bar_auto:
  \mathcode`|="8000 \scan_stop:
  \left\langle
  #1
  \right\rangle
  \group_end:
 }

\NewDocumentCommand{\fixedbraket}{mm}
 {
  \group_begin:
  \char_set_active_eq:nN
    { `| }    % active char is |
    \egreg_bar_fixed: % equal to
  \mathcode`|="8000 \scan_stop:
  \IfNoValueTF{#1}
   { \egreg_braket:n { #2 } }
   { \egreg_braket:nn { #1 } { #2 } }
  \group_end:
 }

\cs_new_protected:Nn \egreg_bar_auto:
 {
  \nonscript\,\middle\vert\nonscript\,
 }
\cs_new_protected:Nn \egreg_bar_fixed:
 {
  \mathinner{\egreg_size: \vert}
 }
\cs_new_protected:Nn \egreg_braket:n
 {
  \cs_set_protected:Nn \egreg_size: { }
  \langle #1 \rangle
 }
\cs_new_protected:Nn \egreg_braket:nn
 {
  \cs_set_protected:Nn \egreg_size: { #1 }
  \mathopen{\egreg_size: \langle}
  #2
  \mathclose{\egreg_size: \rangle}
 }
\ExplSyntaxOff

\begin{document}

$\bra{x}\quad\ket{x}$

$\braket{x|y}$

$\braket[\Big]{x|y|z}$

$\braket*{a|b}$

$\braket[\Big]{a|b|\dfrac{c}{d}}$

$\braket*{a|b|\dfrac{c}{d}}$
\end{document}
```

I'll not comment the code, except for mentioning how easy is to define the value of a character when it will be made active (math active, in this case).

$$\langle x| \quad |x\rangle$$
$$\langle x \,|\, y\rangle$$
$$\Big\langle x \,\Big|\, y \,\Big|\, z \Big\rangle$$
$$\langle a \,|\, b\rangle$$
$$\left\langle a \,\middle|\, b \,\middle|\, \dfrac{c}{d} \right\rangle$$
$$\Big\langle a \,\Big|\, b \,\Big|\, \dfrac{c}{d} \Big\rangle$$

Figure 6: Examples of bras and kets

6 Numbers and units

How should numbers be typed in the LaTeX document? Knuth himself once acknowledged that his usual practice is not very good and realized it when writing "Concrete Mathematics" [3], were numbers are typeset with the Euler font when they're used in their mathematical meaning (and not, say, as page markers).

When a number appears in text and is mentioned as a mathematical object is should be input inside a math formula:

```
a vector space of dimension~$5$
```

But what about large numbers that need to be split in smaller units for readability? For instance, can you spell out 7400043022221 without first counting how many digits the number has? Isn't 7 400 043 022 221 easier to parse? Possibly not for an American who's more accustomed to 7,400,043,022,221 (and probably would be at stake when people talks about meters and liters).

Now let's suppose your scientific paper has several tables with numeric data and you're not sure about the editorial policy of the journal to which you're submitting it. Will the journal require American style or prefer thin spaces for grouping digits?

Table 1: Tables with different formatting options for numbers (Source: Mr Leporello, private communication)

Nation	Number	Nation	Number
Italy	640 375	Italy	640,375
Germany	231 803	Germany	231,803
France	100 002	France	100,002
Turkey	91 329	Turkey	91,329
Spain	1 003 000	Spain	1,003,000

Let's consider the two tables in table 1. They are typeset with exactly the same input, namely

```
\begin{tabular}{
  @{}
  l
  S[table-format=7.0]
  @{}
}
\toprule
Nation  & {Number} \\
\midrule
Italy   &  640375 \\
Germany &  231803 \\
France  &  100002 \\
Turkey  &   91329 \\
Spain   & 1003000 \\
\bottomrule
\end{tabular}
```

and it's siunitx [12] doing all the magic. Of course there is a catch: just before the second copy of the table I added

```
\sisetup{group-separator={,}}
```

I could have added the option also in the bracketed argument to the `S` column, which is one of the facilities made available by the package. Similarly, the big number above has been typeset first with `\num{7400043022221}` and then with

```
\num[group-separator={,}]{7400043022221}
```

The default for the package is to use a thin space as a group separator between digits. An `S` column basically applies `\num` to every entry, but also aligns them at the decimal separator. In the case of our Leporello table, all entries are integers, so they're right aligned.

If an entry belonging to an `S` column is braced, it will be ignored as far as number alignment is concerned and centered on the total width of the column (options are available for left or right alignment). This is obviously needed in the header.

With another option we can easily scale down the figures:

Nation	Number
Italy	640×10^3
Germany	232×10^3
France	100×10^3
Turkey	91.3×10^3
Spain	1.00×10^6

This is achieved with the options

```
\sisetup{
  round-mode=figures,
  round-precision=3,
  scientific-notation=engineering
}
```

and by changing the column specifier to

```
S[table-format=3.2e1]
```

which directs to reserve space for three digits in the integer part, two in the mantissa and one in the exponent. The table body in the input has not changed in any way.

One might write an entire large chapter of *The LaTeX Companion* about siunitx. Some time ago, Joseph Wright took up the job of making a successor package to SIunit adding some features along the way. For version 2 he had the idea of exploiting expl3 which not only allowed for *many* more features and facilities, but made him enter the LaTeX team.[7] He's into chemistry, and tables with numeric data are his staple food.

The main purpose of the package is of course typesetting numbers with their SI unit according to the guidelines of the Bureau International des Poids et Mesures (BIPM). This is also part of the ISO standard mentioned before:

```
\SI{1}{\newton} is defined as
\SI{1}{\kilogram\meter\per\second\squared}
```

will typeset "1 N is defined as $1\,\mathrm{kg\,m\,s^{-2}}$". However, if we prefer slashes instead of negative exponents, we can add to the preamble

```
\sisetup{per-mode=symbol}
```

and the same text will now typeset as "1 N is defined as $1\,\mathrm{kg\,m/s^2}$". The mode can also be changed on a local basis with an optional argument to `\SI`.

All SI units and prefixes are supported:

```
\SI{5}{\tera\meter} \SI{2}{\pico\farad}
```

yields 5 Tm and 2 pF. One can also print just a unit with `\si`: the unit for energy is the J which is the same as $\mathrm{kg\,m^2\,s^{-2}}$.

Going on with our fictional scientist who's uncertain where her breakthrough paper will be published, decimal numbers might require the period as separator, or the comma; in scientific notation, there could be the $\times 10^n$ part or En might be asked for. How to do it? Not to mention uncertainty! Let's take as an example the rest mass of the electron

$$9.109\,383\,701\,5(28) \times 10^{-31}\,\mathrm{kg}$$
$$9.1093837015(28) \times 10^{-31}\,\mathrm{kg}$$
$$9{,}109\,383\,701\,5(28) \times 10^{-31}\,\mathrm{kg}$$
$$(9.109\,383\,701\,5 \pm 0.000\,000\,002\,8) \times 10^{-31}\,\mathrm{kg}$$
$$9.109\,383\,701\,5 \times 10^{-31}\,\mathrm{kg}$$
$$9.109\,383\,701\,5(28)\mathrm{E}{-31}\,\mathrm{kg}$$

The first line has been input with

```
\SI{9.1093837015(28)E-31}{\kilogram}
```

and the following lines by adding an option

```
\SI[⟨option⟩]{9.1093837015(28)E-31}
  {\kilogram}
```

The used options are, in order,

```
output-decimal-marker={,}
group-digits=integer
separate-uncertainty
omit-uncertainty
output-exponent-marker=\mathrm{E}
```

and they can be combined to get the desired effect *without changing the code in the document* if the settings are done with `\sisetup` in the document preamble. When inputting numbers, one can use spaces and either a decimal period or decimal comma. The first mandatory argument to `\SI` behaves the same as the mandatory argument to `\num`, so I'll use the latter:

[7] It seems that understanding and propagating expl3 opens a straight way to the team.

```
\num{12345.678}
\num{12345,678}
\num{12 345.678}
```

will print the same

12 345.678   12 345.678   12 345.678

Very few things are hardwired in siunitx: one can instruct it to ignore something, for instance. Suppose you have a set of numbers with comma separators for groups: the big number already used might occur in the source file as `7,400,043,022,221`. Set (globally or locally) the `input-ignore` option and we can remove the comma from the possible decimal separators:

```
\num[
  input-ignore={,},
  input-decimal-markers={.}
]{7,400,043,022,221}
```

and you'll get

7 400 043 022 221

The package is not limited to 'standard numbers': it also copes with angles, time and complex numbers. For instance, we can type

```
\ang{30.24}
\ang{30;12;44.375}
\num{3-4i}
```

to get

$$30.24°,\ 30°12'44.375'',\ 3-4\mathrm{i}$$

Oh, dear! An upright 'i'! Let's fix it with

```
\sisetup{
  output-complex-root=\mathnormal{i}
}
```

(one could also tell it to use 'j', of course) and get

$$3-4i$$

Phew! Yes, the package obviously adheres to the ISO standard, but it's very customizable.

7 Further reading

Twenty-two years have passed from the seminal paper by Claudio Beccari: we have seen great progress in the field of math typesetting, in particular towards the uniformity that's necessary in technical and commercial reports.

There are other packages that can be tried for the purpose of compliance to the ISO 80000-2:2009 standard. I would mention isomath by Günter Milde [7] and also unicode-math by Will Robertson [8] that provided facilities to the purpose; the former is for legacy `pdflatex`, the latter for XƎLaTeX and LuaLaTeX.

References

[1] C. Beccari. Typesetting mathematics for science and technology according to ISO 31/XI. *TUGboat* 18(1), 1997. `https://tug.org/TUGboat/tb18-1/tb54becc.pdf`

[2] J. Dieudonné. *Treatise on Analysis. Vol. III.* Academic Press, New York-London, 1972. Translated from the French by I. G. MacDonald, Pure and Applied Mathematics, Vol. 10-III.

[3] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics.* Addison-Wesley Publishing Company, Advanced Book Program, Reading, MA, 1989. A foundation for computer science.

[4] E. Gregorio. Simboli matematici in TeX e LaTeX. *ArsTeXnica* 8:7–24, Ottobre 2009. `https://www.guitex.org/home/numero-8`

[5] M. Guiggiani and L. F. Mori. Consigli su come *non* maltrattare le formule matematiche. *ArsTeXnica* 5:5–14, Aprile 2008. `https://www.guitex.org/home/numero-5`

[6] M. Guiggiani and L. F. Mori. Suggestions on how *not* to mishandle mathematical formulae. *TUGboat* 29(2), 2008. `https://tug.org/TUGboat/tb29-2/tb92guiggiani.pdf`

[7] G. Milde. isomath — mathematical style for science and technology, 2012. Version 0.6.1. `https://ctan.org/pkg/isomath`

[8] W. Robertson. Experimental Unicode mathematical typesetting: The unicode-math package, 2019. Version 0.8o. `https://ctan.org/pkg/unicode-math`

[9] W. Rudin. *Real and Complex Analysis.* McGraw-Hill Book Co., New York–Toronto, Ont.–London, 1966.

[10] W. Rudin. *Analisi Reale e Complessa.* Bollati Boringhieri, 1974.

[11] C. Tellechea. L'extension pour TeX et LaTeX systeme, 2019. Version 0.32. `https://ctan.org/pkg/systeme`

[12] J. Wright. siunitx — A comprehensive (SI) units package, 2018. Version 2.7. `https://ctan.org/pkg/siunitx`

◇ Enrico Gregorio
Dipartimento di Informatica,
Università di Verona
`enrico dot gregorio at univr dot it`

The `fewerfloatpages` package[*]

Frank Mittelbach

Abstract

LaTeX's float algorithm has the tendency to produce fairly empty float pages, i.e., pages containing only floats but with a lot of free space remaining that could easily be filled with nearby text. There are good reasons for this behavior; nevertheless, the results look unappealing and in many cases documents are unnecessarily enlarged.

The fewerfloatpages package provides an extended algorithm that improves on this behavior without the need for manual intervention by the user.

Contents

1 Introduction

We start by giving a quick overview of LaTeX's float algorithm and the problems that result from the approach used.

We then look in some detail into possible alterations and improvements to that algorithm and discuss possible issues that need to be resolved. In this section we also describe all configuration possibilities of the extended algorithm.

The final section then documents the code changes that are necessary to LaTeX kernel macros to implement the extension.

1.1 A quick overview of LaTeX's float algorithm

LaTeX's output routine uses a greedy algorithm to place floats near to their call-outs in the source document. The decision of how to place a float is made when the float is first encountered. If possible it is placed onto the current page, either in mid-text, on top or into the bottom area, depending on what is allowed for the float and how many floats are already placed into those areas.

If the float can't be placed immediately, it goes into a defer list, and in order to not accumulate too many unplaced floats LaTeX tries to empty that list whenever there is a chance. This chance comes after the next page break: LaTeX then starts a special "float page" algorithm in which it examines the defer list and from it forms float pages (i.e., pages that contain only floats). If necessary, it generates several float pages and only stops if there are no floats waiting to be placed, or there are too few floats to

[*] The current package version is `v1.0a` dated 2020/02/14.

form a float page, or there are only floats left that are for one or another reason not allowed to be placed in this way.

Finally LaTeX looks at the remaining floats and tries to place as many of them as possible into the top and bottom area of the next page. Then it continues to process further text to fill the text part of that page. Details on the exact behavior of the algorithm are discussed in [1].

1.2 The typical float page and its problems

LaTeX considers a float page to be successfully built if its floats take up more than `\floatpagefraction` of the whole page. By default this parameter is set to `.5` which means that such float pages may end up being half empty.

Many users think that this is not a good value and try to improve on it by enforcing a higher percentage (such as 80%) only to find that this prevents LaTeX in many cases from successfully generating any float page, with the effect that all floats are suddenly piling up at the end of the document.

Why is this the case? In a nutshell, because a higher percentage makes it much more likely that a float can't be placed, because it is not big enough to be used on its own and no other nearby floats can be combined with it, because their combination violates some other restriction, e.g., together they are bigger than a page, not all of them are allowed to go on float pages, etc. The moment that happens this float prevents the placement of all later floats of the same class too (i.e., all figures) and disaster is ensured. In most cases these floats will then never get placed, because they need a float of the right size from a different class to appear, which may in theory happen but is, unfortunately, unlikely.

Tinkering with the parameter settings will usually produce unwanted effects

Thus, while tempting, tinkering with this parameter by making it larger is usually not a good idea, unless you are prepared to place most if not all of your floats manually, by overwriting the placement algorithm on the level of individual floats (e.g., using `!` syntax and/or shifting its position in the source document).

Why does the current algorithm have these problems? To some extent, because it offers only global parameters that need to fit different scenarios and thus settings that are suitable when many floats need to be placed result in sub-optimal paginations in document parts that contain only a few floats, and vice versa. To overcome this problem, either one can try to develop algorithms with many more configurable parameters that act differently in different scenarios or one can let the algorithm follow a main strategy, configurable with only a few parameters (like today), but monitor the process and make more local adjustments and corrections depending on the actual outcome of that base strategy and additional knowledge of the actual situation in a given document part. This is the approach taken by the extension implemented in this package.

2 Improvements to the float page algorithm

A simple way to improve on the existing algorithm, without compromising its main goal of placing the floats as fast as possible and as close as possible to their call-outs, is the following: as long as there are many floats waiting to be placed, generate float pages as necessary to get them placed (using the current algorithm and its parameters).

Once we are unable to build further float pages, do some level of backtracking by checking if we have actually succeeded in placing all floats. If there are still floats waiting to be placed then assume that what has been done so far is the best possible way to place as many floats as possible (which it probably is). However, if we have been able to place all floats onto float pages then check if the last float page is sufficiently full; if not, undo that float page and instead redistribute its floats into the top and bottom area of the next upcoming page. This way the floats will be combined with further text and we avoid a possible half-empty float page.

This approach will not resolve all the problematic scenarios where we find that LaTeX has decided to favor fairly empty float pages over some tighter type of placement. It will, however, help to improve typical cases that do not involve too many floats. For a example, if a single (larger) float appears near the end of a page, then using the standard algorithm it can't be immediately placed (because there isn't enough free space on the current page). It is therefore moved to the defer list and at the page break it is then placed onto a float page (possibly by itself, if it is large enough to allow for that) even though it could perfectly well go into the top or bottom area of the next page and thus be combined with textual material on that page.

A typical case where we don't really want LaTeX to make a float page

With the new algorithm this float page is reexamined and unless it is pretty much filled up already, it is unraveled and its floats are redistributed into the top and bottom areas of the next page. If, however, we have many floats waiting on the defer list, the normal float page algorithm will first place as many of them as possible into float pages and only the last of these pages will be subject to a closer inspection and a possible unraveling.

An extension of this idea (and the one that we actually implement) is to monitor the whole float page generation process and instead of just considering the last float page in the sequence for unraveling, we look at each prospective float page in turn and based on the current situation (e.g., number of floats still being unplaced, free space on the float page, etc.) decide whether this float page should be produced or whether we should stop making float pages and instead place the pending floats into top and bottom areas of the upcoming page.

2.1 Details on the extended algorithm

The main idea of the extended algorithm is to avoid unnecessary cases of float pages especially if those float pages are fairly empty. Natural candidates are single float pages, but even in cases where the current LaTeX algorithm produces several sequential float pages the extended algorithm may decide to replace them by normal pages under certain conditions. However, the main goal is and should remain to place as many floats as soon as possible and so generating float pages when many floats are waiting is usually essential.

Don't unravel a float page if there are too many floats on the defer list

`floatpagedeferlimit`

`\setcounter{floatpagedeferlimit}{⟨number⟩}`

Whether or not unraveling for a float page is considered at all is guided by the counter `floatpagedeferlimit`. As long as there are more floats waiting on the defer list than this number, float pages are not considered for unraveling. The default is `3` which corresponds to the default value for `totalnumber`, i.e., with that setting the unraveling of a floating page has a fighting chance to place all floats into the top and bottom areas on the current page. It would also resolve cases for up to three floats, each larger than `\floatpagefraction`, where the standard LaTeX algorithm would produce three individual float pages.

If you set the counter to `1` then only the last float page in a sequence is considered, and only if it contains only a single float and if there are no other floats that are still waiting to be placed. If you set it to `0`, then the extension is disabled, because float pages are produced only if there was at least one float on the defer list.

Don't unravel if the float page contains many floats

Even if we set `floatpagedeferlimit` to a fairly high value, we may not want to unravel float pages that contain many floats. To support this case there is a second counter that guides the algorithm in this respect.

`floatpagekeeplimit` `\setcounter{floatpagekeeplimit}{⟨number⟩}`

Whenever the float page contains at least `floatpagekeeplimit` floats it will not be unraveled. The default is also `3` so that float pages with three or more floats are not touched. Obviously the counter can have any effect only if it has a value less than or equal to `floatpagedeferlimit` because this is tested first.

Don't unravel if the float page contains at least one `[p]` *float*

There are, however, a number of other situations in which we shouldn't unravel a float page even if the above checks for the size of the defer list were passed successfully. The most important one is the case when the float page contains at least one float that is allowed *only* on float pages (i.e., has a `[p]` argument). Such a float would not be placeable in a top/bottom area on any page and thus would be repeatedly sent back to the defer list (possibly forever).

Don't unravel if the float page is nearly filled

The other case where unraveling would normally be counterproductive is when the particular float page is nearly or completely filled up with floats. If we unravel it, then it is certain that we can place only some of the floats into the top or bottom area of the next page, while some would end up on the defer list. That in turn means that these deferred floats float even further away from their call-out positions than need be.

`\floatpagekeepfraction` `\renewcommand\floatpagekeepfraction{⟨decimal⟩}`

So what is a good way to determine if a float page is "full enough"? A possible answer is that if the remaining free space on that page is less than `\textfraction` we consider it full enough to stay. `\textfraction` defines the minimum amount of space that has to be occupied by text on a normal page, thus if all floats together need so much space that this amount of text could not fit, then trying to place all floats onto a normal page can't succeed and some of them would get deferred for sure. To allow for further flexibility the algorithm uses the variable `\floatpagekeepfraction` (defaulting to `\textfraction`) so if desired a lower (or even a higher) boundary can be set.

The above parameters give some reasonable configuration possibilities to guide the algorithm as to when and when not to unravel a possible float page and instead produce further normal pages. It should be noted, however, that except for the case of setting `floatpagedeferlimit` to `1`, there is always a chance that floats drift further away from their call-outs, because they may not be immediately placeable due to other parameter settings of the float algorithm. For example, the counter `topnumber` (default value 2) limits the number of floats that can be placed in the top area on a normal page and if more remain after unraveling only two can immediately go in this area.

2.2 Possible pitfalls and how to avoid them

The algorithm detects if a float is allowed only on float pages (i.e., is given in the source as `[p]`) and it will ensure that float pages containing such floats are not unraveled.

However, if you have a float with the default specifier `[tbp]` whose size is larger than the allowed size of the top or bottom area (e.g., larger than `\topfraction` × `\textheight`), then this effectively means it can only be placed on a float page.

However, according to the specifier the float is allowed to go into the top or bottom area, so the algorithm, as explained so far, would be allowed to unravel and when that float later is considered for top or bottom placement it will get again deferred and thus move from one page to the next, most likely messing up the whole float placement.

`checktb` *(option)*

There are two possible ways to improve the algorithm to avoid this disaster. One way would be to check the float size when it is initially encountered and remove any specifier that is technically not possible because of the parameter settings and the float size. A possible disadvantage is that this determination will be done once and any later (temporary) change to the float parameters will have no effect. This is currently the package default. It can be explicitly selected by specifying the option `checktb`. In this case you might see warnings like

```
LaTeX Warning: Float too large for top area: t changed to p on line ...
```

`addbang` *(option)* Another possibility is that we automatically add a `!` specifier to all floats during unraveling, i.e., when we send them back for reevaluation. This way such floats become placeable into top and bottom areas regardless of their size. This may result in fewer pages at the cost of violating the area size restrictions once in a while. It is specified with the option `addbang`.

`nocheck` *(option)* If you prefer no automatic adjustment of the specifiers, add the option `nocheck`. In this case you might find that floats of certain sizes are unplaceable and thus get delayed to the end of the document. If that happens, the remedy is either to explicitly specify `[p]` or `[hp]` for such a float (to ensure that they aren't subject to unraveling) or to manually add an exclamation specifier, e.g., `[!tp]` so that LaTeX doesn't use the size restrictions in its algorithm.

2.3 Tracing the algorithm

`trace` *(option)* The package offers the option `trace`, which if used, will result in messages such as

```
[1]
fewerfloatpages: PAGE: trying to make a float page
fewerfloatpages: ----- \@deferlist: \bx@B \bx@D
fewerfloatpages:  starting with \bx@B
fewerfloatpages:  --> success:  \bx@B \bx@D
fewerfloatpages: ----- current float page unraveled
                (free space 192.50336pt > 109.99832pt)
[2]
```

which means that the algorithm is trying to make a float page from the defer list which at that point contained two floats (the float boxes `\bx@B` and `\bx@D`), that it was able to produce a float page containing just `\bx@B` and `\bx@D`, and that the extended algorithm then decided to unravel that float page, because it has an unused space of `192.5pt`, i.e., roughly 16 text lines. With the current `\floatpagekeepfraction` that is too much empty space on the page.

Or it might say

```
fewerfloatpages: PAGE: trying to make a float page
fewerfloatpages: ----- \@deferlist: \bx@D \bx@F \bx@G \bx@H \bx@I
fewerfloatpages:  starting with \bx@D
fewerfloatpages:  --> success: \bx@D \bx@F
fewerfloatpages: ----- too many deferred floats for unraveling (5 > 3)
[3]
```

which means that the algorithm made a float page out of the first two floats from the defer list (i.e., 3 remained). That page was kept regardless of the amount of free space it contained because we have a total of 5 floats on the defer list and the counter `floatpagedeferlimit` has its default value of 3.

The above tracing messages are both from the same test document. What they also (implicitly) show is that the unraveling that happened after page 1 resulted in only one float (`\bx@B`) being placed on page 2, because we see the second one (`\bx@D`) reappearing in the defer list after page 2 got finished. In other words it was moved one page further away from its call-out: the price for getting a nicely filled page 2 instead of a fairly empty float page with roughly 200 points left empty. The final part of that test document then exhibits another type of message:

```
fewerfloatpages: PAGE: trying to make a float page
fewerfloatpages: ----- \@deferlist: \bx@G \bx@H \bx@I
fewerfloatpages:  starting with \bx@G
fewerfloatpages:  --> success: \bx@G \bx@H \bx@I
```

```
fewerfloatpages: ----- all floats placed on float page(s)
fewerfloatpages: ----- current float page kept, full enough
                      (free space 38.99496pt < 109.99832pt)
[4]
```

This means that the remaining floats (that were left unplaced after float page 3 got constructed) formed a float page and that float page was the last in sequence (i.e., all floats have been placed). However, this time the algorithm decided not to unravel it, because it is nicely full: there are only 39 points of free space left on that page.

Three other possible messages are shown in this sequence of tracing lines from a second test document (which is using some uncommon settings: `floatpagedeferlimit` is 10 and `floatpagekeeplimit` is 5):

```
[1]
fewerfloatpages: PAGE: trying to make a float page
fewerfloatpages: ----- \@deferlist: \bx@B \bx@C \bx@D \bx@E \bx@F \bx@G \bx@H
fewerfloatpages:  starting with \bx@B
fewerfloatpages:  --> success: \bx@B \bx@C \bx@D \bx@E \bx@F \bx@G \bx@H
fewerfloatpages: ----- current float page kept (contains at least 5 floats)
[2] [3]
```

In this case 7 floats have been waiting on the defer list and the algorithm was able to construct a float page using all of them. The algorithm then keeps that page because it has 5 or more floats in it (the value of the `floatpagekeeplimit` counter).

The next message in that test document shows what happens when there are not enough floats waiting or they are simply too small (to even get past the `\floatpagefraction` limit):

```
fewerfloatpages: PAGE: trying to make a float page
fewerfloatpages: ----- \@deferlist: \bx@I \bx@J
fewerfloatpages:  starting with \bx@I
fewerfloatpages:  --> fail
fewerfloatpages:  starting with \bx@J
fewerfloatpages:  --> fail
fewerfloatpages:  --> fail: no float page made
[4]
```

So no float page was made, but for some reason (that becomes clear later) the two floats also didn't got distributed into the top or bottom area of the next page. Instead they remained on the defer list and during processing of page 4 one more float was found so that after that page the defer list had grown to length 3:

```
fewerfloatpages: PAGE: trying to make a float page
fewerfloatpages: ----- \@deferlist: \bx@I \bx@J \bx@K
fewerfloatpages:  starting with \bx@I
fewerfloatpages:  --> success: \bx@I \bx@J \bx@K
fewerfloatpages: ----- current float page kept, contains a float
fewerfloatpages:       with p but no t or b specifier
[5]
```

This time all floats could be placed, but again the float page wasn't unraveled (even though in the test document it contained a lot of white space) because of the fact that one of its floats (in fact the first though that can only be deduced implicitly) was specified as a "float page only" float. This explains why on page 4 `\bx@I` couldn't be placed into the top or bottom area and then all following floats of the same class (the test document contained only `figure` floats) couldn't be placed either.

Detailed tracing of the complete algorithm If you want detailed tracing of the complete algorithm, also load the fltrace package and enable the tracing with `\tracefloats` anywhere in your document. Note, however, that the resulting output is very detailed but rather low-level and unpolished.

2.4 Local (manual) adjustments

If the extended algorithm is used you will get fewer float pages that contain a noticeable amount of white space. By adjusting `\floatpagekeepfraction` and the counters `floatpagekeeplimit` and `floatpagedeferlimit` you can direct the algorithm to unravel more or fewer of the otherwise generated float pages. However, in some cases it might happen that redistribution of the floats into the top and bottom areas of the next page(s) may result in some of them drifting too far away from their call-outs. If that happens, you can either try to change the general parameters or you could help the algorithm along by using the optional argument of individual float environments. The two main tools at your disposal are

- using the `[!..]` notation to allow the float to go into the top or bottom area even if it would be normally prevented by other restrictions;
- using `[p]` to force a float into a float page as that prevents the algorithm from unravelling the float page which contains that float.

As an alternative you can, of course, temporarily alter the definition of the command `\floatpagekeepfraction` or the values of two counters in mid-document, but remember that they are not looked at when a float is encountered in the source but when we are at a page break and LaTeX attempts to empty the defer list, which is usually later and unfortunately somewhat asynchronous, i.e., not easy to predict.

3 The implementation

We start off with the package announcement. Requiring a fairly new LaTeX kernel is not absolutely necessary but it will help to ensure that we patch what we think we patch and in the future it means that will can be assured that the rollback functionality of the kernel is available in case will need to support several releases of the package.

```
⟨*package⟩
\NeedsTeXFormat{LaTeX2e}[2018-04-01]

\ProvidesPackage{fewerfloatpages}
               [\fewerfloatpagesdate\space \fewerfloatpagesversion\space
                improve float page generation (FMi)]
```

3.1 Option handling

This release of the package has four options: `trace` for tracing the algorithm, `addbang` and `checktb` to handle cases where the float size in combination with the float specifiers makes it difficult if not impossible to place the floats, and `nocheck` to not make adjustments for that case.

The option `trace` enables tracing of the algorithm and is implemented by giving the command `\fl@trace` (which is also used by the fltrace package) a suitable definition.

To handle the case that the fltrace package is loaded first, we use `\providecommand`, so that its definition is not overwritten, but used if it is already available. If the package is loaded later everything works fine because it unconditionally defines `\fl@trace`, i.e., overwrites whatever fewerfloatpages has defined.

```
\DeclareOption{trace}
             {\providecommand\fl@trace[1]%
               {{\let\@elt\@empty\typeout{fewerfloatpages: #1}}}}
```

The other three options are mutually exclusive so we number them 0 to 2 in the command `\fp@strategy` to ensure that only one is ever active. Option `nocheck` does nothing, with the cost that some floats may float to the end of the document. Option `addbang` adds a `!` to floats that are sent back for reevaluation when a float page gets unraveled. Option `checktb` implements a different approach to handling problematic floats: the vertical size of a float is checked, and if it is too large to be allowed into the top or the bottom area, any t or b specifier is replaced by p (or dropped if p is already specified).

```
\def\fp@strategy{0}%
\DeclareOption{nocheck}{\def\fp@strategy{0}}  % better name?
\DeclareOption{addbang}{\def\fp@strategy{1}}
\DeclareOption{checktb}{\def\fp@strategy{2}}
```

The actual implementation is done later. The default is currently `checktb` but this may change to `addbang` based on user feedback.

```
\ExecuteOptions{checktb}
\ProcessOptions
```

3.2 Tracing code

`\fl@trace` The command `\fl@trace` is used to output tracing information. By default the tracing of the algorithm is turned off, so `\fl@trace` will simply swallow its argument. But if fltrace is loaded or the option `trace` is given then the command already has a definition so we don't change it here.

```
\providecommand\fl@trace[1]{}
```

(*End definition for* `\fl@trace`.)

3.3 User-level interfaces

For the most part the packages provides internal code that extends the float algorithm of LaTeX. There are, however, also three new parameters that guide this algorithm; they are defined in this section.

`\floatpagekeepfraction` The fraction that the algorithm uses to decide whether a given float page is so full that it would be pointless to unravel it for the reasons outlined above. The default is whatever fraction has been chosen as the minimum amount of text that needs to be on a normal page (i.e., `\textfraction`).

```
\newcommand\floatpagekeepfraction{\textfraction}
```

(*End definition for* `\floatpagekeepfraction`. *This variable is documented on page 57.*)

`floatpagedeferlimit`
`\c@floatpagedeferlimit` The algorithm will only consider unraveling float pages if there are not too many floats on the defer list. The definition of "too many" is provided through the counter `floatpagedeferlimit` if there are more floats waiting to be placed; float pages are generated until their number falls below this level. Thus, a value of 0 will disable the whole algorithm and a value of 1 means that only float pages with a single float might get unraveled and only if there aren't others still waiting to be placed.

```
\newcounter{floatpagedeferlimit}  \setcounter{floatpagedeferlimit}{3}
```

(*End definition for* `floatpagedeferlimit` *and* `\c@floatpagedeferlimit`. *These variables are documented on page 56.*)

`floatpagekeeplimit`
`\c@floatpagekeeplimit` A float page that contains at least this number of floats will also be kept. The default is 3 but if you have a lot of small floats it might be better to set this to a higher value.

```
\newcounter{floatpagekeeplimit}   \setcounter{floatpagekeeplimit}{3}
```

(*End definition for* `floatpagekeeplimit` *and* `\c@floatpagekeeplimit`. *These variables are documented on page 57.*)

3.4 Patching the LaTeX kernel commands

\@tryfcolumn The main macro we have to patch to extend LaTeX's algorithm is `\@tryfcolumn`. That command is changed when fltrace gets loaded, so we make our definition as late as possible to ensure that it will survive.

```
\AtBeginDocument{%
```

```
\def \@tryfcolumn #1{%
  \global \@fcolmadefalse
  \ifx #1\@empty
  \else
    \fl@trace{PAGE: trying to make a float
                   \if@twocolumn column/page\else page\fi}%
    \fl@trace{----- \string #1: #1}%
    \xdef\@trylist{#1}%
    \global \let \@failedlist \@empty
    \begingroup
      \let \@elt \@xtryfc \@trylist
    \endgroup
```

Up to this point the definition is the same as in the original algorithm. At this point the switch `\if@fcolmade` tells us if making a float page was successful and the original algorithm then called `\@vtryfc` and removed the floats used for this float page from the defer list.

In the extended algorithm this is the place where things start to differ as we may not want that float page to actually come into existence.

```
    \if@fcolmade
```

As a first step we count the number of floats in the defer list and save the result in `\fp@candidates`.

```
      \fp@candidates\z@
      \def\@elt##1{\advance\fp@candidates\@ne}%
        #1%
      \let\@elt\relax
```

Now we compare this number with the values of the counter `floatpagedeferlimit` and if it is higher we definitely want to keep the float page. The rationale is that if we unravel now, then all floats from the defer list need to go into the top/bottom areas (or get deferred again but to a later page) and so a high number means the defer list will not get shortened very much and too many floats will get delayed further.

```
      \ifnum \fp@candidates >\c@floatpagedeferlimit
        \fl@trace{----- too many deferred floats for unraveling
                    (\the\fp@candidates\space> \the\c@floatpagedeferlimit)}%
      \else
```

Otherwise we do a bit more testing. First we set `\if@fcolmade` back to false; after all our goal is to not keep the float page. If during the tests we decide otherwise we set it back to true, which then signals that it should stay.

We also count the floats on the float page, reusing `\fp@candidates` for that, which is why we initialize it to zero.

```
        \global\@fcolmadefalse
        \fp@candidates\z@
```

The actual checking is done with `\fp@analyse@floats@for@unraveling` and it loops over `\@flsucceed`, i.e., the floats for that float page. This checks if any float for that page has only a `[p]` specifier and if so it sets `\if@fcolmade` back to `true` and as a side effect it also does the counting for us. Furthermore, it also changes the switch to true if it finds at least `floatpagekeeplimit` floats on that page.

```
      \let\@elt\fp@analyse@floats@for@unraveling
        \@flsucceed
      \let\@elt\relax
```

Now we recheck the state of the switch and if it still says `false`, all tests so far indicate that we don't want the float page.

```
      \if@fcolmade  \else
```

But we aren't done yet: the float page might be nicely filled, in which case it would be a shame to unravel it. During the above loop we also measured the free space on the float page and stored it in `\fp@unused@space` (see `\@xtryfc` below). We now compare that to the maximum free space that we consider to be still okay and if there is more we finally do the unraveling.

```
        \@tempdima\floatpagekeepfraction\@colht
        \ifdim \fp@unused@space >\@tempdima
          \fl@trace{----- current float page unraveled^^J%
                    \@spaces\@spaces\@spaces\space\space\space
                    (free space \fp@unused@space\space > \the\@tempdima)}%
```

For this we basically return all floats back to the defer list. The switch is still `false` so it doesn't need changing.

```
          \xdef #1{\@failedlist\@flsucceed\@flfail}%
```

However, we may also want to add a `!` specifier to each of the floats (if the `addbang` option was given) so we loop over all the floats once more to get this done.[1]

```
          \let\@elt\fp@maybe@add@bang
            \@flsucceed
          \let\@elt\relax
        \else
```

But if we want to keep the float page after all, we have to set the switch back to `true` so that the rest of the algorithm proceeds correctly.

```
          \global \@fcolmadetrue
          \fl@trace{----- current float page kept, full enough^^J%
                    \@spaces\@spaces\@spaces\space\space\space
                    (free space \fp@unused@space\space < \the\@tempdima)}%
        \fi
      \fi
    \fi
```

The next `\else` matches the first `\if@fcolmade`, i.e., the case that the algorithm wasn't able to make any float page. If we are tracing the algorithm, we want to tell the user about this.

```
    \else
      \fl@trace{ --> fail: no float page made}%
    \fi
```

Finally, at this point we are back in the original algorithm. Now the switch tells the truth about whether or not we want to make a float page, and if so, we go ahead and produce it.

```
    \if@fcolmade
        \@vtryfc #1%
    \fi
  \fi}%
}% -- END of \AtBeginDocument
```

(*End definition for* `\@tryfcolumn`.)

[1] This could have been integrated with `\fp@analyse@floats@for@unraveling` but there is not much gain if any and by keeping it separate the processing logic seems clearer to me.

`\@makefcolumn` In contrast to `\@tryfcolumn` this macro will always make float pages out of the deferred floats. It is used by `\clearpage` when we really need the floats to get out because there is no further text coming up. Thus, in that case we should not unravel the float pages. That would happen with the kernel definition of `\@makefcolumn` as that calls `\@tryfcolumn` which we just changed above. We therefore modify its definition to include the original code for `\@tryfcolumn` instead of calling our updated version.

Again this change is made at `\begin{document}` so that it is not overwritten in case fltrace is loaded afterwards.

```
\AtBeginDocument{%
\def\@makefcolumn #1{%
  \begingroup
    \@fpmin -\maxdimen
    \let \@testfp \@gobble
```

At this point the original definition called `\@tryfcolumn` and the lines above ensured that it was always succeeding in making a float page. However, since we have changed that command to do unraveling we had better not use it any more. Instead we replace it by its original definition (with the addition of two tracing lines).

```
    \global \@fcolmadefalse
    \ifx #1\@empty
    \else
      \fl@trace{PAGE: trying to make a float
                      \if@twocolumn column/page\else page\fi}%
      \fl@trace{----- \string #1: #1}%
      \xdef\@trylist{#1}%
      \global \let \@failedlist \@empty
      \begingroup
        \let \@elt \@xtryfc \@trylist
      \endgroup
      \if@fcolmade
        \@vtryfc #1%
      \fi
    \fi
  \endgroup
}%
}% -- END of \AtBeginDocument
```

(*End definition for* `\@makefcolumn`.)

`\@xtryfc` The only change to `\@xtryfc` is the addition of the `\fl@trace` calls. But this extra tracing info is generally useful and should also be done in the fltrace package.

The macro initiates a float page trial starting with the first float in `\@trylist`. More detailed explanations can be found in the documented sources of the LaTeX kernel [2].

```
\def\@xtryfc #1{%
  \fl@trace{ starting with \string#1}%
  \@next\reserved@a\@trylist{}{}%
  \@currtype \count #1%
  \divide\@currtype\@xxxii
  \multiply\@currtype\@xxxii
  \@bitor \@currtype \@failedlist
  \@testfp #1%
  \@testwrongwidth #1%
  \ifdim \ht #1>\@colht
     \@testtrue
  \fi
  \if@test
    \@cons\@failedlist #1%
    \fl@trace{ --> fail}%
  \else
    \@ytryfc #1%
  \fi
}%
```

(*End definition for* `\@xtryfc`.)

`\@ytryfc` The command `\@ytryfc`, which is also part of the code in the kernel, loops through the defer list and tries to build a float page starting with the float passed to it in `#1`. If it succeeds, the floats that are part of the float page are listed in `\@flsucceed` and the switch `\if@fcolmade` is set to `true`. Also of interest to us is that inside the code `\@tempdima` holds the size taken up by the floats, so we can use this to calculate the unused space on the float page and store it in `\fp@unused@space` for use in our extended algorithm.

```
\def\@ytryfc #1{%
  \begingroup
    \gdef\@flsucceed{\@elt #1}%
    \global\let\@flfail\@empty
    \@tempdima\ht #1%
    \let\@elt\@ztryfc
    \@trylist
    \ifdim \@tempdima >\@fpmin
      \global\@fcolmadetrue
```

This branch is executed when the floats together are big enough to form a float page. Thus, this is the right place to calculate the free space by subtracting the used space from the column height (which may not be the full height if there are spanning floats in two column mode).

```
      \@tempdimb\@colht
      \advance\@tempdimb-\@tempdima
      \xdef\fp@unused@space{\the\@tempdimb}%
```

The remaining code is again unchanged except that we added two additional tracing lines (though those should be added to the fltrace package too one of these days).

```
    \else
      \@cons\@failedlist #1%
      \fl@trace{ --> fail}%
    \fi
  \endgroup
  \if@fcolmade
    \let\@elt\@gobble
    \fl@trace{ --> success: \@flsucceed}%
  \fi}
```

(*End definition for* `\@ytryfc`.)

`\@largefloatcheck` The final kernel macro we need to patch is `\@largefloatcheck`. This is called when a float box is constructed and it checks if that box is larger than the available `\textheight`, which would mean it could never be placed anywhere, not even on a float page. The code therefore reduces the box size as necessary and issues a warning.

This macro is therefore a natural candidate to also check if the float size is too large for the float to go into top or bottom areas (if the option `checktb` is used).

```
\def \@largefloatcheck{%
  \ifdim \ht\@currbox>\textheight
    \@tempdima -\textheight
    \advance \@tempdima \ht\@currbox
```

```
      \@latex@warning {Float too large for page by \the\@tempdima}%
      \ht\@currbox \textheight
    \fi
```

The `\fp@maybe@check@tb` does the checking (or nothing if the option is not given).

```
    \fp@maybe@check@tb
  }
```

(*End definition for* `\@largefloatcheck`.)

3.5 Internal helper commands and parameters

`\fp@candidates` We use an internal counter to count the number of floats in the defer list and on a float page under construction.

```
  \newcount\fp@candidates
```

(*End definition for* `\fp@candidates`.)

`\fp@unused@space` In `\fp@unused@space` we store the amount of free space on the current float page.

```
  \def\fp@unused@space{}
```

(*End definition for* `\fp@unused@space`.)

`\fp@analyse@floats@for@unraveling` With `\fp@analyse@floats@for@unraveling` we loop over the floats on the float page, i.e., `#1` will be one such float.

One of its tasks is to count the floats (in `\fp@candidates`) and check if there are at least `floatpagekeeplimit` of them (which means the float page should definitely be kept).

Its most important task, however, is to check if one of the floats has only a `p` specifier but no other. In that case it is essential that we not unravel the float page because such a float would then only go back onto the defer list as it has no place to go except a float page.

```
  \def\fp@analyse@floats@for@unraveling#1{%
    \advance\fp@candidates\@ne
    \ifnum \fp@candidates <\c@floatpagekeeplimit
```

So far we haven't got enough floats to know that this float page should be kept so we check the given float specifiers.

The test may look a little weird,[2] but what we want to know is this: is there a `p` (third bit) but neither a `b` (second bit) nor a `t` (first bit). We don't care about `h` or `!` which are the next two bits in the float counter nor any of its higher bits (which encode the type of float). So we divide the integer number by 8, which drops the two least significant bits (think of the integer represented in binary format), and then multiply it again by 8. As a result the first two bits are zeroed out. We then compare the result with the original value and if the two values are the same then the `b` and `t` bits must both have been zero from the start. And since the float was on a float page we also know that it had a `p` specifier.

```
      \@tempcntb\count#1%
      \divide\@tempcntb 8\relax
      \multiply\@tempcntb 8\relax
      \ifnum \count#1=\@tempcntb
```

[2] "Little" might be an understatement. Encoding a lot of information in individual bits of the counter value associated with a float was a great way in the early days of LaTeX to preserve macro space (and absolutely essential back then), but these days ... Anyway, it is the way it is and that part can't really be changed without breaking a lot of packages.

In that case we set `\if@fcolmade` to `true` to signal that this float page should be kept, generate a tracing message and change `\@elt` to become `\@gobble` to quickly jump over any remaining floats in the loop without doing further tests or generate further tracing messages.

```
      \global \@fcolmadetrue
      \fl@trace{----- current float page kept, contains a float}%
      \fl@trace{\@spaces\space\space with p but no t or b specifier}%
      \let\@elt\@gobble
    \fi
```

On the other hand, if we have seen enough floats we also know that the float page should be kept, so change the switch, give some tracing info and stop checking:

```
  \else
    \global \@fcolmadetrue
    \fl@trace{----- current float page kept
              (contains at least \the\fp@candidates\space floats)}%
    \let\@elt\@gobble
  \fi
}
```

(*End definition for* `\fp@analyse@floats@for@unraveling`.)

`\fp@maybe@add@bang` The helper `\fp@maybe@add@bang` is used to loop through all of the floats of a float page (receiving each as `#1` in turn) and add a `!` specifier if there wasn't one before. However, we only define it if we implement strategy 1 which is option `addbang`.

```
\ifnum\fp@strategy=1
  \def\fp@maybe@add@bang#1{%
```

Find out if the fourth bit is set (which means no `!`) and if so subtract 16 from the float counter which means setting it to zero.

```
    \@boxfpsbit #1\sixt@@n
    \ifodd \@tempcnta
      \global\advance\count#1-\sixt@@n
    \fi
  }
\else
  \let\fp@maybe@add@bang\@gobble
\fi
```

(*End definition for* `\fp@maybe@add@bang`.)

`\fp@maybe@check@tb` The code in `\fp@maybe@check@tb` is used in `\@largefloatcheck` to test if the float has a `t` or `b` specifier but is too large to fit into the respective area. This test is not made by default but only if the option `checktb` is used, i.e., strategy 2.

```
\ifnum\fp@strategy=2
  \def\fp@maybe@check@tb{%
```

Again this is a case of looking at various bits in the float counter value in binary notation. If the specifier contained a `!` we are ok and it would be wrong to change the specifier, because in that case size restrictions for areas do not apply. For this we have to test the fourth bit which means dividing by 16 and then checking if the result is odd or even (odd means there was no `!`).[3] The kernel `\@getfpsbit` does this for us and stores the result in `\@tempcnta` so we can test this with `\ifodd` to see if the bit was set.

[3] I'm sure we had good reasons to implement it this way in 1992 — we probably saved a few bytes which was important back then. But it is certainly odd that for `!` a value of zero means that it was specified on the float while for all other specifiers a value of `1` indicates that the specifier was given.

```
    \@getfpsbit \sixt@@n
    \ifodd \@tempcnta
```

If there was no `!` we check if the height of the float is too large to fit into the top area.

```
      \ifdim \ht\@currbox>\topfraction\textheight
```

If that is the case we also check the first bit of the float counter to see if a `t` was specified. For this we use `\@getfpsbit` again but this time with `2` as the argument since we test the first bit.

```
        \@getfpsbit \tw@
        \ifodd \@tempcnta
```

If `t` was specified we need to remove it (next line) and add (if not already present) a `p` instead. This is done by `\fp@add@p@bit`. Finally we add a warning for the user about the change.

```
          \global\advance\count\@currbox -\tw@
          \fp@add@p@bit
          \@latex@warning {Float too large for top area: t changed to p}%
        \fi
      \fi
```

A similar test and action is needed for bottom floats; here we need to look at and zero out the second bit (i.e., using 4 as a value).

```
      \ifdim \ht\@currbox>\bottomfraction\textheight
        \@getfpsbit 4\relax
        \ifodd \@tempcnta
          \global\advance\count\@currbox -4\relax
          \fp@add@p@bit
          \@latex@warning {Float too large for bottom area:
                           b changed to p}%
        \fi
      \fi
    \fi
  }
```

In all other cases `\fp@maybe@check@tb` does nothing.

```
\else \let\fp@maybe@check@tb\relax \fi
```

(*End definition for* `\fp@maybe@check@tb`.)

`\fp@add@p@bit` The command `\fp@add@p@bit` adds the `p` specifier which means checking the third bit and if not set, adding 8 to the float counter.

```
\def\fp@add@p@bit{%
   \@getfpsbit 8\relax
   \ifodd \@tempcnta \else \global\advance\count\@currbox 8\relax \fi}
```

(*End definition for* `\fp@add@p@bit`.)

```
⟨*package⟩
```

References

[1] Frank Mittelbach. How to influence the position of float environments like figure and table in LaTeX? *TUG*boat 35:3, 2014.
https://www.latex-project.org/publications/indexbytopic/2e-floats/

[2] LaTeX Project Team. The LaTeX 2_ε Sources (660+ pages), 2020.
https://www.latex-project.org/help/documentation

◇ Frank Mittelbach
Mainz, Germany
https://www.latex-project.org
https://ctan.org/pkg/fewerfloatpages

Frank Mittelbach

Typesetting Bangla script with LuaLaTeX

Ulrike Fischer, Marcel Krüger

In [2], Md Qutub Uddin Sajib describes some experiences and insights concerning typesetting Bangla.

While we don't know Bangla, we want to share some additional information about the LuaLaTeX side, especially about how to use the possibilities of the new `HarfBuzz` library.

1 `HarfBuzz` in LuaLaTeX — the engine choice

Typesetting a script is more than placing glyphs side by side. Even in the rather simple western scripts there are already ligatures, kerning and accents to handle and many scripts have much more complex rules. Up to now LuaLaTeX wasn't very good with such scripts. This wasn't due to a fundamental deficiency, but a lack of manpower: To implement the shaping rules one needs people knowing the script, knowing Lua and having the time and the will to put both together. The experience with `xetex` was much better as this engine relied on an external library from the start — these days on `HarfBuzz` [3]. For quite some time there has been a wish for `luatex` to also support use of `HarfBuzz`.

In 2019, this became possible: with `harftex` [1] and `luahbtex`, two engines with built-in `HarfBuzz` support were available. After some discussion the LaTeX team decided to base LuaLaTeX in TeX Live 2020 on `luahbtex` and integrated in fall 2019 the necessary Lua code into `luaotfload`. As an engine change is a major step the two TeX distributions TeX Live and MiKTeX added the new engine already in November 2019 and the LuaLaTeX-dev format has been mapped to it. This allowed beginning "real world" tests.

So, from the various LuaLaTeX variants mentioned by Sajib, only the following should be considered if `HarfBuzz` is wanted:

TeX Live 2019 / before April 2020

`lualatex-dev`	= `luahbtex`	+ LaTeX-dev

TeX Live 2020 / after April 2020

`lualatex`	= `luahbtex`	+ LaTeX
`lualatex-dev`	= `luahbtex`	+ LaTeX-dev

To make the best use of the new `HarfBuzz` integration it is important to keep one's TeX distribution, notably including `luaotfload`, up to date to benefit from the development and corrections of bugs.

2 Using `HarfBuzz` in LuaLaTeX

With XeLaTeX `HarfBuzz` is always used to shape a font (with the exception of legacy TeX fonts). This doesn't hold for LuaLaTeX. Here one can choose on a font-by-font basis between the `base mode` (mostly used for math fonts), `node mode` (used for text) and the new `harf mode` (which has a number of sub-modes). `HarfBuzz` is an *addition* to, not a replacement for, the existing font shapers. If the library isn't used the new engine behaves like `luatex`.

Table 1: Example rendering with the various modes

`base`	কণ্‌যা এখন কি করবিে?
`node`	কণ্‌যা এখন কি করিবে?
`harf`	কণ্যা এখন কি করিবে?

Table 1 shows the rendering of an example text in the three modes. In comparing the output one can see — even if one doesn't understand the script — that `base mode` doesn't know much, `node mode` a bit and `harf mode` a lot about the script. E.g., the third word is `U+0995 U+09BF` (corresponding to the Bengali letters "KA": ক and "I": ি). While the simpler `base mode` just prints these letters next to each other, the more advanced modes `node` and `harf` reverse the order. The first word in the examples shows that `node mode` is still missing some shaping rules that `harf mode` applies.

When using the `fontspec` package the mode can be chosen with the `Renderer` option. It is also important to specify the correct script. The `harf mode` in table 1, for example, can be done with this font declaration:

```
\setmainfont{Noto Sans Bengali}
    [Renderer=HarfBuzz,Script=Bengali]
```

3 The "dotted circle" mystery

In [2], Sajib also discusses how to typeset a glyph without getting an unwanted dotted circle as a base character. With XeLaTeX it isn't easy to overrule the rendering that the `HarfBuzz` library considers to be correct (though `\XeTeXglyph` can be used). But in LuaLaTeX it can be done by switching to another mode of typesetting. Compare the output of e.g. `\char"90BE`:

`harf mode`: ◌া   `base/node mode`: া

4 Coloring glyphs

The standard LaTeX color commands insert specials or literals. This interrupts the input and so can disturb the font shaping. Figure 1 shows an example where the color commands inhibit the reordering of the glyphs.

With LuaLaTeX one can use attributes instead by loading the `luacolor` package. With it, the code in figure 1 gives this output: কি কি

At first glance this looks okay but the coloring is actually odd. The code asks to color the KA red

```
\char"0995\char"09BF \quad
\color{red}\char"0995
\color{green}\char"09BF
```

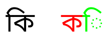


Figure 1: Wrong shaping due to color commands

(ক) and the I green (ি) but in the output the colors are reversed. This shows a general problem with this approach: the attributes are attached to the *input* chars. When getting back the shaped output from `HarfBuzz` (in this case in reversed order) `luaotfload` doesn't always know how the input and the output relate and has to guess how to reattach the attributes. Switching to `node mode` solves the problem for this specific case as in this mode `luaotfload` has full control over the shaping and so can make a better decision how to set the color.

But the core of the problem lies deeper: font shaping takes a cluster of n input chars and outputs m glyphs and there is not always a clear mapping between attributes of an input char to the attributes of the output glyphs. As an example, take the input `\color{red}f\color{green}f\color{blue}i`: what color should the output ffi-ligature glyph be?

5 Coloring the output glyphs

We've seen that adding color commands to the input does not always lead to satisfactory results. What about color instructions that target the *output* glyphs? The code in figure 2 (which requires `luaotfload 3.12`) shows that this is possible. The code defines a color scheme that maps colors to output glyphs and uses it in the font declaration.

The difficulty with this method is to correctly reference the desired output glyphs. The code shows two different methods: by name ("ivowelsignbeng") and by index, the GID, of the glyph. Both methods have drawbacks.

Glyph names can fail as not every font uses the same names for the same glyphs, and some fonts don't contain them at all. The name "`t`" for example works fine with TEX Gyre Heros but not with Arial. The handling of glyph names also differs between the modes: With the `HarfBuzz` renderer glyph names currently work only with `ttf` fonts (this will probably change with the next `HarfBuzz` version).

Index numbers are more reliable but they are different for every font (and can change if a font is updated). The GID of "`t`" is 87 in Arial and 106 in TEX Gyre Heros.

A third possibility, more difficult to implement, is to use the ToUnicode mapping of the glyph. The problems here are several: different glyphs can have the same ToUnicode, some glyphs (e.g., accented

```
\directlua{
  luaotfload.add_colorscheme("my_scheme",
  { ["FF0000"] = {"kabeng"},
    ["00FF00"] = {"ivowelsignbeng"},
    ["0000FF"] = {369} % GID of "nadarabeng"
  })}
\newfontface\colorbengali{Noto Sans Bengali}
  [Renderer=Harfbuzz,
   Script=Bengali,
   RawFeature={color=my_scheme}]

{\colorbengali
\char"0995 \char"09BF
\char"09A8 \char"09CD \char"09A6
\char"09CD \char"09B0}
```

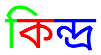


Figure 2: Coloring glyphs in a font

characters) can have more than one ToUnicode value (and which one is used in a document cannot always be easily predicted), some glyphs are clusters and so have quite long values, and finally, some have no ToUnicode mapping at all.

So while the results of coloring output glyphs can be quite good, this clearly requires some skill and the right fonts.

6 Coloring parts of a glyph

The methods mentioned above don't help to color the *parts* of a single glyph, e.g., ন্দ্র. While the *input* consists of five chars (ন্‌দ্‌র) the *output* is *one* glyph in the font and coloring parts of a glyph can only be done if the font has special support for it. Here the only option is to enhance the font with color support or to draw the glyph with Ti*k*Z or some graphic application and color it manually.

References

[1] K. Hosny. Bringing world scripts to LuaTEX: The HarfBuzz experiment. *TUGboat* 40(1):38–43, 2019. `https://tug.org/TUGboat/tb40-1/tb124hosny-harfbuzz.pdf`

[2] M. Q. U. Sajib. Typesetting the Bangla script in Unicode TEX engines — experiences and insights. *TUGboat* 40(3):263–269, 2019. `https://tug.org/TUGboat/tb40-3/tb126sajib-bangla.pdf`

[3] The FreeType Project. HarfBuzz, a text shaping library. `https://harfbuzz.org`

◇ Ulrike Fischer
Mönchengladbach
`ulrike.fischer (at) latex-project.org`

◇ Marcel Krüger
Hamburg
`marcel.krueger (at) latex-project.org`

Typographers' Inn

Peter Flynn

No time for copyright

For several years I have used a locally-developed *LaTeX Tips* document as an extension to the *Very Short Guide* which I maintain as the veryshortguide package. The document lists 10 tips distilled from my support time which I suppose constitute a personal FAQ. It's one of the very few documents which I actually wrote from scratch in LaTeX, because I wanted it to work as an example of how you can produce a 4pp folder FAQ by hand (normally I write in another system entirely, and transform to LaTeX only for formatting).

All of which is a long way round to saying that the reason it's not included in the veryshortguide package is that the first thing it says is not to reinvent the wheel, for which I (illegally) use a cartoon without permission. It shows a caveman offering a round wheel to two friends hauling a cart with square wheels, only to have his offer refused because the friends were 'too busy' trying to make the square-wheeled cart work. In extenuation, I did try to find who the artist was, in order to get permission, but a Google image search shows it used on hundreds of sites without permission, and no evidence of whose it originally was, and searches on several image-library sites we subscribe to also turned up nothing.

I was therefore delighted when I saw a live-tweet from a Project Jupyter session at Lund University of a guest lecture on Python tools by Ben Krikler of Bristol University, showing a drawing making exactly the same point. Dr Krikler has kindly given me permission to use the drawing, so I will now be able to add the document to the package, and use it here.

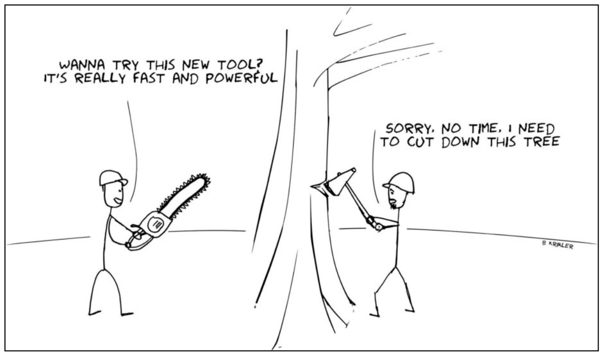

WANNA TRY THIS NEW TOOL? IT'S REALLY FAST AND POWERFUL
SORRY. NO TIME. I NEED TO CUT DOWN THIS TREE
© KRIKLER

Figure 1: No time [Ben Krikler, University of Bristol]

If you're setting type for someone, it's conventional to assume that they have permission to use all the images that are not their own. In a formal contractual situation, you simply make it a condition, so that if there is a problem later, it's their problem, not yours. However, when you're under time pressure as an author or editor, especially if you don't have an amanuensis to take care of administrivia, it's easy to forget, or to assume it'll be OK 'because it's on the web and everyone does it'. Please don't.

No time to learn

Back to where I left off: the principal problem with documentation is that no-one wants to read it, no matter how prettily formatted or useful (☞ the item 'Be useful.' in the list on p. 72). This applies just as much to the documentation for LaTeX packages as to the articles in *TUGboat* and all the books on TeX. Tobias Oetiker's comprehensive and wonderful *The [Not So] Short Guide to LaTeX 2ε* [3] and other guides on the TeX Users Group's list at `https://tug.org/begin.html` should be on every new user's reading list, as of course should my own *Formatting Information* [1] but if the users believe there's no time to read them, they risk ending up 'learning' about LaTeX from word of mouth and obsolete web sites, along with all the bad practices inherited from decades of 'sitting by Nellie'.[1]

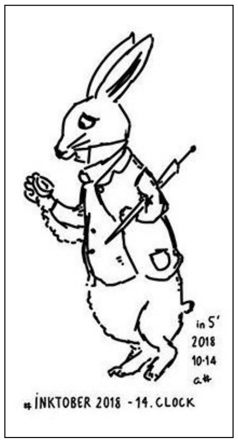


Figure 2: No time [@fiveminutedoodles]

So what can we do, typographically, to help make our documentation more attractive? As it turns out, quite a lot, if you have the freedom to influence the layout.

Avoid Computer Modern. There isn't anything wrong with it; in fact I quite like it for classical book or journal work, but it helps to perpetuate the myth that LaTeX only has one font.

[1] The term comes from the learning-by-watching training methodology used in mid-20th century British industry, and adopted by *The PracTeX Journal*, `https://tug.org/pracjourn`, for their Q&A column.

Use XƎLaTeX and the fontspec package to make use of all your typefaces, or any of the myriad fonts that come in TeX distributions, many supported by all engines (`https://tug.org/fontcatalogue`).

Reset the margins. The defaults in the standard document classes are designed for printing on smaller paper sizes than Letter or A4, and don't let you make use of the space effectively. Use the geometry package to change this.

Look at the line-spacing. If you use lines of full width with margins less than 3cm ($2\frac{1}{4}$") on A4 or Letter paper, you may need to widen the line-spacing a little (unless you use 12pt or larger) otherwise it becomes harder for the eye to flip back to the start of the next line. Alternatively, use a two-column layout with multicol.

Use color. Unless you are going to print big quantities, for which color becomes expensive, look at introducing spot color like colored section heads with the xcolor and sectsty packages, or add a light background color on figures or tables.

Make code interesting. If you're documenting a program or language or markup with blocks of code examples, use the listings package to provide syntactically-colored or fontified listings.

"...it just makes you look better to have a little color."

Lady Bird Johnson (out of context, but nevertheless...)

Break up the text. Try not to have whole pages of text without a break. You may be able to introduce pull quotes (as above) or add marginal highlights. Talk to the author or editor about using deeper sectioning to deal with subtopics instead of lists.

Be useful. Indexes, glossaries, and heralded cross-references all add to the usefulness of a document, but sometimes fail to add to its usability by being non-obvious or too complex.

Alas, if you're presented with a fixed layout, you have to work within it, but there is usually some scope for adding visual interest without changing the style.

Afterthought: proofing the unproofable

When you get asked to typeset something for somebody, you are in a sense making yourself partly responsible for the content. Most of the time this isn't a problem, but it's not just what they say, it's how they say it. Quite apart from the 'greengrocer's apostrophe', as in 'Fresh Orange's' and the cognitive problems I have referred to before of line-breaks in centered displays, there's a perennial difficulty with dangling misplacement.

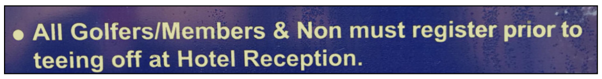

• All Golfers/Members & Non must register prior to teeing off at Hotel Reception.

Figure 3: Misplaced referent [Author's collection]

In the case of Figure 3 it's the awkward expression of whom it applies to, and the fact that it's the registration which has to take place at the hotel reception, not the teeing-off (I checked: the first tee is a long way from reception).

Here, it's a trivial case which causes a smile and harms no-one. If you typeset it, no-one is going to come running with complaints, although you might get some good-natured flak from pedants.

- **All golfers, members or not, must register at reception before teeing off.**

In more serious circumstances, though (thinking of signs, posters, forms, or other critical documents relating to disease control), it could direct people to the wrong place, or exclude people for whom it is intended, or *vice versa*.

If you're asked to typeset material during a period of difficulty and pressure, it's important to take the time to read and understand what you're being asked to set, and suggest changes if you think they're likely to be misunderstood. Charles Fyffe, in *Basic Copyfitting*, has an important comment:

> It is true, of course, that the copy should be correct when it comes to you, but it rarely is: and few people have the head for detail of a good typographer. In general, if *you* don't, no one else will correct the copy, and probably no one else has the ability. [2, pp. 59–60]

References

[1] P. Flynn. *Formatting Information: A beginner's guide to formatting documents with LaTeX*. Silmaril, Cork, Ireland, Nov 2018. `ctan.org/pkg/beginlatex`

[2] C. Fyffe. *Basic Copyfitting*. Studio Vista, London, 1969.

[3] T. Oetiker, H. Partl, et al. The Not So Short Guide to LaTeX 2_ε: Or LaTeX 2_ε in 139 minutes, Feb 2018. `ctan.org/pkg/lshort`

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

An attempt at ragged-right typesetting

Udo Wermuth

Abstract

Plain TeX (as well as LaTeX) provides the command `\raggedright` to typeset text *unjustified*, that is, only the left margin is straight, the right margin varies in its length and looks ragged. The command gives good results for narrow columns, for example, when it is applied in marginal notes and short quotations. However, its output addresses only the English tradition. For wider measures it produces the ragged-right output that is called “Rauhsatz” in German and not the more demanding “Flattersatz”. In this article we examine what recommendations are formulated by a German typographer to typeset text in “Flattersatz” and then make an attempt to capture these recommendations in TeX macros.

1 Introduction

By default, plain TeX produces paragraphs that are justified. In most cases it is able to produce decent output if the measure allows that at least 60 characters, punctuation marks, and blank spaces fit into one line. With some care the length of the line might be reduced to only 40 symbols and blank spaces and acceptable output can still be created. Smaller line widths should use *unjustified text* [3, p. 27]. (Most of the time this article uses in its experiments a measure of 150 pt which allows an average of ≈ 32 characters per line in `cmr10` whose lowercase alphabet length is 127.5837 pt; see [3, p. 28].)

For languages that are written from left-to-right *unjustified* text usually means that the left-hand side has a straight margin as for justified text and the right-hand margin is ragged as line breaks occur at the end of the line when the line is sufficiently filled. There are several phrases to describe this situation: *flush-left typesetting*, *ragged-right typesetting*, *unjustified*; all have the same meaning.

In the world of TeX, unjustified is often called ragged-right typesetting as plain TeX defines the control word `\raggedright` to typeset paragraphs unjustified [8, p. 356]. LaTeX provides a *flushleft* environment and a `\raggedright` declaration; see [11, pp. 111–112]. The latter control word has a different implementation than the plain TeX version; the next section will discuss this further.

For centuries, book printing has applied justification to nearly all paragraphs with one important exception: epics or poetry in general. These texts were often printed unjustified. But in the modern interpretation of traditional typesetting, book designers and typographers started to apply ragged-right margins to all kinds of books, occasionally even novels, at least in the 20th century [16, p. 20]. As in the case of justification, certain best practices and guidelines to avoid bad looking results have been formulated to typeset such texts unjustified.

Motivation. I am aware that ragged-right typesetting is a difficult topic. Once I needed ragged-right typesetting for a German text and the TeX macro `\raggedright` didn’t gave a result that I liked (too many hyphens) and I fixed it manually. Much later I looked up unjustified typesetting in my books about typography, and tried to develop macros that implement ragged right for a larger variety of measures. I describe some of them in this article, but I have to admit that my experience with them is still limited. And note, this is the first time that I apply them to an English text. Thus, I look at this article as a start to the discussion of how to approach the problem of automated ragged-right typesetting with TeX.

This paper discusses neither the readability of unjustified margins nor the situations when they shall be applied. A lot of factors are involved but here the assumption is made that the designer has determined that the needs of the text request or at least allow ragged-right typesetting.

Two words in German. German typographers translate “ragged-right margins” with two German words: *Rauhsatz* and *Flattersatz*. (“Satz” is typesetting; “rauh” has here the meaning “raw” or “untreated”; “flattern” means “to flutter”.) The two German words are not synonyms but rather show two different sets of rules — or better, *recommendations* — formulated to distinguish layouts by the significance of the visual effect of the right margin. Their difference is very important as the former is considered low quality typesetting compared to the latter. (“Rauhsatz damages the language, it is only acceptable for cheap, short-lived works” [16, p. 90]; my translation.) Rauhsatz is used with small line widths; Flattersatz is applied for wider measures or as an alternative for justified text. It requests that the right margin forms a rhythmical pattern: Lines shall alternate in length. Precise descriptions follow in section 4.

I don’t know why English typographers use only one word instead of the two that are considered necessary by German typographers. It might be that the concept of “Flattersatz” is so distinct from the English ragged-right typesetting as the line lengths vary so much — probably due to the longer average word length in German — that no separate English word is needed because the effect doesn't independently stand out in English typesetting. But as the traditions for different languages have different recommendations it might be useful to have a more complicated ragged-right typesetting macro that is flexible enough to reflect these diverse traditions.

Contents. First, section 2 discusses some available implementations in TeX to typeset text unjustified and section 3 presents a few examples of bad ragged-right output. Section 4 describes some recommendations, and section 5 tries to capture them in TeX macros. Modern TeX installations run fast and we can throw more code at the problem than in earlier days. A general approach to define a ragged-right typesetting macro is introduced in section 6. Section 7 applies the results to English and German examples. Control sequences to set line breaks manually in this context are the topic of section 8. A short summary in section 9 concludes the article.

2 What do TeX and LaTeX implement?

First, the definition for ragged-right typesetting in plain TeX is discussed. Actually, there are two definitions in `plain.tex` [8, pp. 355–356]. I change the format and replace a constant-value register by its content to enhance readability. Plain TeX does not have a command that switches back to justification so that the command is normally used in a group, else it is active for the rest of the document.

```
\def\raggedright{\rightskip=0pt plus 2em
   \spaceskip=0.3333em \xspaceskip=0.5em\relax}
\def\ttraggedright{\tt
   \rightskip=0pt plus 2em\relax}
```

How does the first definition work? The line-breaking algorithm of TeX includes two globs of glue, the `\leftskip` and the `\rightskip`, which are placed at the beginning and the end of every created line [8, p. 100]. The default value in plain TeX for both skips is 0 pt, so they have no effect. Here the `\rightskip` keeps the natural width of 0 pt but adds stretchability. This allows a line to end before it covers the entire width of `\hsize`.

The next two assignments change how TeX determines the amount of white space that it puts between words. Usually these *interword spaces* are based on the values of certain `\fontdimen` parameters defined by the current font. But the above assignments make interword spaces a fixed width, with a larger value after colons, periods, exclamation marks, and question marks. More precisely, the `\spaceskip` defines the interword space while the `\xspaceskip` sets the space after characters with a space factor $\geq$ 2000 [8, p. 76 and p. 351]. The values are given in the unit "em" to set them with respect to the font that is active when the macro is called. Only this font should be used in the text. (If more fonts are involved, then all `\fontdimen3` and `\fontdimen4` values for interword stretch and shrink should be cleared and `\spaceskip` shouldn't be used.) For `cmr10`, `\spaceskip`'s natural width equals the value of the font's interword space, i.e., `\fontdimen2`, but the `\xspaceskip` is larger than the sum of `\fontdimen2` and `\fontdimen7`, the extra space, which is 0.44444 em; see [8, p. 433].

It is a specialty of American English typesetting to have more white space after certain punctuation marks; in [3, pp. 29–30], this practice is classified deprecated as space itself is a kind of punctuation. The article [13] represents a longer example of ragged-right typesetting by an American typographer and type designer; it applies extra space after end-of-sentence periods. For German texts, the book [4, p. 183] requests the normal interword space after the period for the end of the sentence. But an old rule is mentioned that recommends setting a larger white space manually in certain circumstances to support the identification of a sentence's end: a small interword space and an abbreviation that ends with a period and that occurs at the end of a sentence.

The second definition does essentially the same for monospaced fonts as the first does for proportional fonts. The former has by default a fixed-width interword space, so the assignments to the parameters `\spaceskip` and `\xspaceskip` can be omitted. Note however that the font is explicitly selected before the assignment to `\rightskip` is made to get the correct value for the unit "em".

On page 338 of *The METAFONTbook* [9] code from the file `testfont.tex` is shown that "improves on plain TeX's `\raggedright`":

```
\ifdim\fontdimen6\testfont<10pt
   \rightskip=0pt plus 20pt
\else
   \rightskip=0pt plus 2em
\fi
\spaceskip=\fontdimen2\testfont
\xspaceskip=\fontdimen2\testfont
   \advance\xspaceskip by \fontdimen7\testfont
```

The improvements are: `\rightskip`'s stretchability is set to min(20 pt, 2 em) and interword spaces use the relevant `\fontdimen` parameters of the font that is tested.

LaTeX 2_ε's `\raggedright` uses a similar setup. Again I format the code for the column width and replace constant-valued registers by their contents.

```
% see ltmiscen.dtx
\newskip\@rightskip
  \@rightskip=0pt plus 0pt minus 0pt
\def\raggedright{\let\\=\@centercr
  \@rightskip=0pt plus 1fil
  \rightskip=\@rightskip
  \leftskip=0pt plus 0pt minus 0pt
  \parindent=0pt}
```

Two backslashes are used to implement a line break; see [11, p. 112]. This aspect of the macro is not analyzed further in this article.

As in the plain TEX macro, `\rightskip` gets a new value. Here it is done in two steps involving the skip register `\@rightskip`. The first difference is that infinite stretchability is assigned, which makes very short lines acceptable. It creates “very ragged-looking paragraphs” [12, p. 103]. The second difference is that nothing is done to the interword spaces, they keep their stretch- and shrinkability. Moreover, the macro suppresses indentation.

3 What is bad ragged-right typesetting?

Of course, we are interested in good ragged-right typesetting. But the word “good” is sometimes difficult to define whereas it is quite easy to show some aspects that should not happen when a paragraph is typeset ragged right. So let’s learn first what should be avoided, at least according to German tradition.

To start, the right margin shouldn’t create a contour like an arrowhead, i.e., lines get longer in small steps and after the longest line the next lines decrease their length in small steps. Such structures are easily spotted by a reader and distract him from the text. (In the case that such a shape is wanted, use TEX’s `\parshape` [8, p. 101] or a package like `shapepar` from CTAN.)

Example 1: Description

A short plain TEX example of ragged-right typesetting with several criticizable aspects.

TEX input

```
\ninepoint\raggedright\noindent
The text of this example is typeset ragged
right with the macro of plain \TeX; it is
designed to demonstrate some bad things that
happen if the line breaks are not manually
improved. For example, the first three lines
and the last three lines build a pair of
inverse slopes.
```

TEX output

The text of this example is typeset ragged right with
the macro of plain TEX; it is designed to demonstrate
some bad things that happen if the line breaks are not
manually improved. For example, the first three lines
and the last three lines build a pair of inverse slopes. ▯

The macro `\ninepoint` is defined in `tugboat.sty`.

In the next two paragraphs the line lengths vary and don’t build a contour; but other problems occur.

Example 1 continued: TEX input

```
\noindent
Another kind of problem is shown in the next
line: The last word sticks out as the other
two lines are a little bit shorter; it looks
similar to the first lines.

\noindent
Surprise! This is a new paragraph. But as it
is not indented it is very hard to see that in
the output.
```

TEX output

Another kind of problem is shown in the next line:
The last word sticks out as the other two lines are a
little bit shorter; it looks similar to the first lines.
Surprise! This is a new paragraph. But as it is not
indented it is very hard to see that in the output. ▯

Of course, the “a” at the end of one line and the identification of the last line of the paragraph should be improved. In [7], Fig. 5 [10, p. 83], the first situation is called the “sticking-out problem”.

The problem that the start of a paragraph is not clearly signaled occurs with justified typesetting too. And in justified texts it can happen that one word is repeated at the start or the end of several consecutive lines. Such *stacks* can appear too if the text is typeset ragged right. But it can also happen that the text itself forms an unintended shape.

Example 1 continued: TEX input

```
\noindent
This text creates a known pattern at its right
end. A twist of fate\Dash or it happens to be bad
luck with \TUB’s measure. No, not at all, I
designed the text. I {\sl placed\/} the
uppercase letters at the {\sl end\/} of the
lines; I wanted to make them build this pattern.
```

TEX output

This text creates a known pattern at its right end. A
twist of fate — or it happens to be bad luck with *TUG-
boat*’s measure. No, not at all, I designed the text. I
placed the uppercase letters at the *end* of the lines; I
wanted to make them build this pattern. ▯

Okay, we have seen a few problems which might occur with ragged-right typesetting but not with justified text and we have seen that some problems of justified typesetting can also appear with ragged right. The three problems of the first group that we saw are: Either the right margin or the words at the line ends build a shape and a single word might stick out into the margin. To the common problems belong the identification of the start of a paragraph if it has no indentation and the occurrence of stacks.

Of course the output is so bad that the whole text of the example must be reformatted. How to fix the situation? In the first paragraph a `\break` between "are not" avoids any contour. A tie after the colon fixes the second paragraph. A break between "not indented" in the third paragraph and one between "text. I" together with a tie in "I wanted" in the fourth improves the rest of the output.

Example 1 continued: TeX output improved

The text of this example is typeset ragged right with
the macro of plain TeX; it is designed to demonstrate
some bad things that happen if the line breaks are
not manually improved. For example, the first three
lines and the last three lines build a pair of inverse
slopes.

Another kind of problem is shown in the next line: The
last word sticks out as the other two lines are a little
bit shorter; it looks similar to the first lines.

Surprise! This is a new paragraph. But as it is not
indented it is very hard to see that in the output.

This text creates a known pattern at its right end. A
twist of fate — or it happens to be bad luck with *TUG-
boat*'s measure. No, not at all, I designed the text.
I *placed* the uppercase letters at the *end* of the lines;
I wanted to make them build this pattern. ▯

The output is not perfect yet but much better than before. This little example shows what typographers state about the manual rework of line breaks: The processing of ragged-right texts is laborious and the time expenditure is higher than with justified text [4, p. 152].

A closer look at the sticking-out problem. We saw that plain TeX's `\raggedright` sets fixed-width interword spaces and assigns stretchability to `\rightskip`. With this setup the sticking-out problem cannot occur very often.

First, let's define what the phrase "sticking-out problem" means for us, exactly. Let's assume that a one- or two-letter word (or word part) is placed by TeX at the end of a line. This word *sticks out* if the line without the word and its preceding interword space is longer than both neighboring lines and the line is recognizable as neither the last nor the next to last.

This definition declares a word part with two letters and a hyphen at its end as short. And the word doesn't stick out if instead of an interword space a hyphen or a dash connects it to the previous word in the line. The last two lines are excluded but that doesn't mean that a short word at their end looks good or is in any way acceptable if it sticks out.

For later use: Longer words that are captured in a rhythmical pattern can be tolerated. That is, we look at a five-line block if the neighboring lines are shorter than the line with the floating word. And this scenario is acceptable if the first and/or last line of this block have at least nearly the same length as the middle line.

Let's try to find out under which circumstances a short word isn't moved to the next line in a normal text. To keep it simple let's assume that neither penalties nor additional demerits are involved. Then the demerits for a line are computed by

$$(\texttt{\textbackslash linepenalty} + \texttt{\textbackslash badness})^2\;; \qquad (1)$$

see [8, pp. 97–98]. With fixed-width interword spaces the badness of a line is represented by the used amount of the stretchability of the `\rightskip` [8, p. 30]. In other words: The badness can be read off from the length of a line. Different line lengths usually have different badness values if the stretchability of `\rightskip` isn't extremely large: The longer a line the smaller the badness, except for the last line whose badness is always 0 if its width is less than `\hsize` and `\parfillskip` has its default setting.

So we have a line of badness β if the word that sticks out (let's call it `w`) and the space in front of it are deleted. The second line is shorter than this line. Thus its badness is larger: $\beta + \kappa$. And the first line with `w` has a smaller badness: $\beta - \mu$. If `w` is moved to the second line the badness of this line gets smaller too: $\beta + \kappa - \nu$.

We can assume $\beta > \mu > 0$ and we know $\kappa \ge 0$ as the second line without `w` is shorter than the first line without `w` and that $\mu \ge \nu - \kappa$ as the first line with `w` is longer than the second with `w`. And we have $\nu \ge \mu$ as the badness is based on a cubic function (see [8, p. 97] or equation (3) in section 5) and a longer line reduces its badness by a smaller amount than a shorter if the same amount of text is added.

Let's name the `\linepenalty` λ; then (1) gives two sums for the demerits of the two lines. If `w` sticks out the demerits of the two lines are

$$(\lambda + \beta - \mu)^2 + (\lambda + \beta + \kappa)^2$$

and when `w` is moved to the second line we get

$$(\lambda + \beta)^2 + (\lambda + \beta + \kappa - \nu)^2\,.$$

The sticking-out problem is avoided if the second sum is smaller than the first.

$$\begin{aligned}
&(\lambda{+}\beta{-}\mu)^2 + (\lambda{+}\beta{+}\kappa)^2 > (\lambda{+}\beta)^2 + (\lambda{+}\beta{+}\kappa{-}\nu)^2\\
\iff\quad &\mu^2 - 2(\lambda+\beta)\mu > \nu^2 - 2(\lambda+\beta+\kappa)\nu\\
\iff\quad &(\mu - 2\lambda - 2\beta)\mu > (\nu - 2\lambda - 2\beta - 2\kappa)\nu.
\end{aligned}$$

The negative value of the right-hand side does not get smaller if the factor ν is replaced by $\mu \le \nu$. Thus

$$\begin{aligned}
&\mu > \nu - 2\kappa\\
\iff\quad &\mu + \kappa > \nu - \kappa
\end{aligned}$$

which is always true if $\kappa > 0$ as $\mu \geq \nu - \kappa$. In other words: A word cannot stick out if the following line has a larger badness than the line without the word.

But badness is computed by a heuristic. So a line might be shorter than another although both have the same badness value.

Example 2: Description
Show that a word in the third-last line can stick out.

TeX input

```
\tenpoint\raggedright\noindent
Look: Never odd or even OR: neve ro ddo reveN, I
mean: Never odd or even OR: never odd or even.
It’s a palindrome!
```

TeX output

Look: Never odd or even OR: neve ro ddo reveN, I
mean: Never odd or even OR: never odd or even.
It’s a palindrome! ▯

The width of the second line is 216.63643 pt; the first line up to and including the comma measures 216.77533 pt. Thus the “I” sticks out. This happens as the badness values are small and they give identical demerits independent of a line break in front of “I”. (In older versions of TeX with, for example, a different formula to compute the demerits this was probably not the case. See [10, p. 94].)

How to avoid this unwanted sticking-out problem without manual intervention? The only way I know for this case is to change the sequence of how TeX selects the breakpoints; see [14, pp. 372–373]. So either use `\looseness` or a three-line specification for `\parshape` without effect.

Example 2 continued: TeX definitions

```
\parshape 3 0pt \hsize 0pt \hsize 0pt \hsize
```

TeX output

Look: Never odd or even OR: neve ro ddo reveN,
I mean: Never odd or even OR: never odd or even.
It’s a palindrome! ▯

On the other hand if the second line has badness 0 independent of the contents then the calculation to avoid the sticking-out problem requires that $(\lambda + \beta - \mu)^2 + \lambda^2 > (\lambda + \beta)^2 + \lambda^2$ which is only possible for $\mu > 2\lambda + 2\beta$. This implies a negative λ as $\beta > \mu > 0$. Thus, if the last line of a paragraph in our ragged-right scenario has badness 0, a sticking-out look alike can easily occur in the second-last line as shown in example 1 that uses $\lambda = 10$.

Therefore, the `\parshape` technique doesn’t remove the ‘a’ that sticks out in example 1. But it can be avoided by changing the value of `\linepenalty`.

Example 3: Description
Avoid the sticking out of ‘a’ in a paragraph of example 1.

TeX input

```
\linepenalty=-154
```

TeX output

Another kind of problem is shown in the next line:
The last word sticks out as the other two lines are
a little bit shorter; it looks similar to the first lines. ▯

But such a value of `\linepenalty` means that TeX likes lines with a high badness, i.e., short lines for ragged-right paragraphs. This generates not very good-looking output (see also [15, pp. 410–411]). As (1) excludes penalties and additional demerits, other ways to avoid this sticking-out scenario exist in this case as, for example, the parameter `\adjdemerits` plays a rôle. The assignment of a negative value to this parameter helps: It must be set to ≤ -24296 to get the output of example 3. It isn’t always advantageous to avoid a sticking-out word in the penultimate line by setting parameters to extreme values.

And there are other effects: Even if the second line is a little bit longer and has therefore a smaller badness value than the first, the visual impression that a word sticks out might occur.

Example 4: Description
Show that a word in the third-last line might appear to stick out.

TeX input

```
\ninepoint\raggedright\noindent
See this: Never odd or even OR neve ro ddo
reveN, I mean: Never odd or even. AND Never
odd or even. It’s a palindrome!
```

TeX output

See this: Never odd or even OR neve ro ddo reveN, I
mean: Never odd or even. AND Never odd or even.
It’s a palindrome! ▯

Thus one consequence of the above is that an author who uses TeX’s `\raggedright` has to check the right margin of the typeset text and to fix unwanted effects like the sticking-out problem.

4 What is good ragged-right typesetting?

What are the recommendations of typographers for ragged-right texts? They do not seem to be universally established in a way that they can be written down as “rules”. Different people describing what they understand as *good typography* for the same language might list different sets of recommendations. Is it best to find the intersection of several sets — the universal recommendations accepted by several typographers — or should a consistent set of a widely recognized typographer be used?

I decided to follow one typographer, Friedrich Forssman. For the German language I use a text that discusses many details and gives several examples of good and bad ragged-right outputs; it is the book [4] by Forssman and R. de Jong. The book [16], co-authored by Forssman, contains additional examples. Of course, the recommendations are tailored for German texts and German readers. Therefore I compare the recommendations with the statements in R. Bringhurst's book [3], a much shorter text which aims for brevity [3, p. 9], to identify significant differences for English texts. An example text by R. Southall is taken from *TUGboat* [13].

Ten recommendations. Here is the list of recommendations that I extracted from [4]. These recommendations are not copied verbatim from the book. They are my interpretation of the information given in the text and the examples.

R1 "zone". A *zone* at the right margin is defined that must be entered by a line before it can be broken. That is, except for the last lines of the paragraphs the lines cannot get arbitrarily short; at least they must reach the left boundary of the defined zone. Its width is determined by the hyphenation parameters (see R9 "hyphens") and manual reworking. In a narrow column it should not be too wide or the margin looks frayed out. [4, p. 152, p. 158, p. 159]

R2 "flutter". Ideally short and long lines should occur in turn; the back and forth of the line ends should form a rhythmical pattern. Notably, the right margin inside the zone should not look like an unsuccessful try to reach justification. [4, p. 124, p. 152]

R3 "exceed". The zone must not end at `\hsize` if the line lengths are varying considerably. Since the right margin is not evident in this case, the length of the longest line is allowed to be wider than `\hsize`. In two-sided printing, the longest lines of the reverse side might show through from its straight left margin. [4, p. 152]

R4 "no shape". The end of the lines should not form a contour or a shape, to avoid distracting the reader from the text. With restricted hyphenation (see R9 "hyphens", case b) even four consecutive lines with line lengths that increase in small steps are objectionable if the measure is wide enough and the line lengths can vary properly. [4, p. 154, p. 158]

R5 "end of line". A short word should not occur at the end of a line if the neighboring lines are shorter, i.e., the sticking-out problem should be avoided; especially one-letter words must be eliminated. Longer words at the end of a long line that stick out should be supported at least by words from two lines above or below, i.e., from one neighboring long line (see R2 "flutter"), if the zone is not narrow. [4, pp. 152–154, p. 155, p. 157]

R6 "last line". If the paragraphs have no indentation then the last line of each paragraph must leave enough white space between its end and the left boundary of the zone if the start of paragraphs isn't signaled in another way. That is, the end of a paragraph must be identifiable and it should not look like a line that has the minimal length. The distance from the left boundary of the zone should be between 2 and 4 em. [4, p. 143, p. 154]

R7 "indent". In small columns an indentation of the first line is often unwelcome. [4, p. 143, p. 154]

R8 "spaces". The interword spaces are all of equal width. Only their natural widths are used, the settings for their stretch- and their shrinkability are ignored. [4, p. 124, p. 146, p. 152, p. 154]

R9 "hyphens". Hyphenation occurs in two forms.
a) The settings of justified text for the length of the parts can be kept. Larger stacks of hyphens, i.e., consecutive lines ending with a hyphen or a dash, are tolerated than in justified texts.
b) Long syllables and *good* hyphenation points are preferred, i.e., the break shall respect the semantic structure of the word. In this case it is recommended to have at least four characters (and the hyphen) on the first line and at least four on the second; in this article, this structure is named $(4, 4)$. Even better are five letters in both cases, i.e., $(5, 5)$. That is, words that are broken must have at least eight or better ten letters. [4, p. 124, p. 155, p. 158]

R10 "exceptions". In an emergency R8 "spaces" and R9 "hyphens" might be violated but only in a very limited way. Interword spaces might not be fixed: A very small amount, at most 0.015 em, can be used to *shrink* the interword spaces but only if no other solution is found. In R9b the pair $(3, 4)$ instead of $(4, 4)$ might be used. [4, p. 124, p. 155]

Measurements in the examples of [4] with line widths between 10.5 pc and 33 pc show that the zone widths vary from 23 pt to 57 pt; 48 pt for the widest measure. The ratio of zone width to `\hsize` lies between 12 and 20%; 27% creates a frayed out margin.

The two cases in R9 "hyphens" mark the major distinction why in German two words are in use for ragged-right typesetting as each case implies the settings for other recommendations. With case R9a the right margin becomes more compact and the line widths vary not much; this implements Rauhsatz. It is the method of choice if the `\hsize` is small. The case R9b is considered to be the *artistic* ragged-right typesetting, the Flattersatz. In this setup R2 "flutter" should be obeyed as strictly as possible.

What does "as strictly as possible" mean? The recommendations describe an *ideal* that an author cannot achieve perfectly in all cases. In two examples on pages 152 and 154 of [4] three to five lines appear in blocks of nearly the same line length, page 20 of [16] has five lines with increasing lengths, and page 91 contains a 5-line block in which the lengths of first three lines increase in small steps, the lengths of the fourth and fifth decrease by a small and a large width, respectively. So it builds a shape but isn't as eye-catching as the contour in example 1.

As indicated above, other typographers recommend more or other criteria. For example, [2, p. 45] adds to R2 "flutter" that at least one third of the lines shall be "sufficiently" filled.

The comparison. The book [3], p. 29, gives a self-involved example of ragged-right typesetting. It distinguishes between a ragged-right margin looking like a "neatly pinched piecrust" that is generated with the default typesetting parameters for hyphenation, spaces, etc., and a "hard rag" which is applied in the book. Let's check whether the ten recommendations above are covered.

R1 "zone": no (and yes). A zone isn't mentioned and the text contains the statement that "no minimum line" is defined. But of course, the shortest non-last line and the `\hsize` form a zone.

R2 "flutter": no. This topic isn't described in the text and the example doesn't obey it.

R3 "exceed": unknown. Again not mentioned in the text but as the paragraph with the ragged-right margin appears between justified paragraphs no line exceeds the measure.

R4 "no shape": unknown. The aspect isn't mentioned in the text. But the example has once four lines whose lengths increase in small steps. Thus the trend goes toward: no.

R5 "end of line": unknown. There are no long lines in the sense of R2 "flutter" so the concept doesn't seem to be applicable. But the example has no short words that stick out at the end of a line.

R6 "last line": probably yes. Again not explicitly mentioned in the text but the last line of the example is much shorter than the other lines.

R7 "indent": unknown. Indentation is in general recommended on page 39 without mentioning ragged right. The example uses a wide measure and starts with an ornament to clearly distinguish it from the indented justified paragraphs.

R8 "spaces": yes. The text states to use "fixed word spaces".

R9b "hyphens": yes. The text requires that hyphenation can occur only at explicit hyphens or at manually inserted hyphenation points.

R10 "exceptions": unknown: not mentioned in the text and not obviously applied in the example.

Half of the recommendations cannot be decided because of lack of information; three times there seems to be an agreement, and two have received a "no". These two topics aren't explained in the book. That there is no minimum line has mainly to do with the unwillingness to hyphenate words but there is no short line without need.

In total I cannot say that the ideal of [3] for ragged-right typesetting is reflected by the ten recommendations that I extracted from [4]. The concept of alternate line length with its consequences isn't found in [3]. The text [13] doesn't help. Learning by example is in this case suboptimal: The first paragraph has alternating line lengths but two paragraphs later a contour is built; the last paragraph in section 5 has five strictly increasing line lengths and two paragraphs later a two-letter word sticks out.

Recommendations vs. `\raggedright`. Let's look at plain TeX first. The assignment `\rightskip=0pt plus 2em` implements R1 "zone". This defines the zone width before the text is manually adjusted; in a strict sense it's violating R1. The zone measures 2 em in the first pass and $\sqrt[3]{2} \times 2\,\mathrm{em} \approx 2.52\,\mathrm{em}$ in the second and third pass; see formula (2) below.

The other assignments `\spaceskip=0.3333em` and `\xspaceskip=0.5em` address R8 "spaces", although, of course, two widths are used instead of a single interword space, this being for American English and not German.

Hyphenation is not affected by the macro, so it is the case R9a in which the settings of justified text are used; i.e., for German texts it implements Rauhsatz. All other recommendations are not covered.

As shown above, LaTeX2ε's `\raggedright` assigns infinite stretchability to `\rightskip`. This violates R1 "zone" because there is no controlled zone: All lines have badness 0 ([8, p. 101]). TeX's main criteria to judge about the quality of a line is lost. And as the interword spaces aren't fixed width the LaTeX implementation violates R8 "spaces" too. To switch off indentation completely doesn't obey R7 "indent" which looks at the `\hsize`.

For better ragged-right typesetting — according to the ten recommendations — an author should use the (currently unmaintained) package `ragged2e`; see [12, pp. 105–106]. It reassigns plain TeX's value to `\rightskip` and provides new parameters to change not only this glue but also `\leftskip`, `\parindent`, and `\parfillskip`. And it sets fixed-width interword spaces so that it obeys R8 "spaces".

Out of scope. The recommendations are for paragraphs but, of course, the page layout must be considered too. A page that contains paragraphs that are typeset ragged right with a wide zone needs a different optical alignment for a centered page number at the bottom of the page than a page with justified text. On the other hand, in a two-column layout the white space between the columns (the gutter) might be smaller if the text is unjustified; it must only be larger than the interword space [4, p. 153]. A line that separates columns should be placed close to the straight margin [16, p. 93].

The opacity of the paper in double-sided printing has already been mentioned. Substantial transparency might let the straight left margin of the reverse page soothe the ragged-right margin [4, p. 152].

5 Implementation of the recommendations

As shown in the previous section plain TeX provides a macro that puts its focus only on the zone and the interword spaces. The user has to decide how to handle aspects like indentation of the first line, hyphenation, etc., and thus the user must invoke other commands or change parameter values.

Before we decide what a plain TeX command for ragged-right typesetting should do, techniques of TeX are analyzed to consider how each recommendation can be implemented. In this analysis more than the obvious solution will be discussed.

Figure 1a) on the next page shows how plain TeX's `\raggedright` typesets the first paragraph of a fairy tale with a measure of 150 pt. The complete English text is available in [10, pp. 84–85, Fig. 6].

R1 "zone". Above, an imperfect implementation of the zone was stated by using `\rightskip`. With a natural width of 0 pt and a finite stretchability the zone is defined up front. Even if the glue has to stretch more than its plain stretchability, i.e., the badness of the line is > 100, the `\tolerance` guarantees that there is a zone of limited width. Of course, this width must be chosen carefully with respect to the available line length. If `\pretolerance` equals `\tolerance`, the width of the zone does not depend on TeX's pass. If the stretchability of the `\rightskip` is named r^+ and the current tolerance τ, then the width of the zone in the pass is given by

$$\sqrt[3]{\frac{\tau}{100}} \times r^+. \qquad (2)$$

With $\tau = 100$, only the specified amount r^+ is available to fill the line. But we can make use of a little bit wider zone in a second pass. For example, if the width should be 20 pt in the first pass and 22 pt in the second pass, then we set `\pretolerance` $= 100$ and `\tolerance` $= 133$ as $100(22/20)^3 = 133.1$ so that $\sqrt[3]{1.33} \times 20\,\text{pt} \approx 1.0997 \times 20\,\text{pt} \approx 22\,\text{pt}$.

And there are other ways to define a zone. One implementation is given in [7] ([10, pp. 93–94]). In this setup `\rightskip` isn't used; instead the spaces are made active and become a sequence of three items: a horizontal skip with the stretchability for the zone and 0 pt natural width, a penalty of value 0, and a second horizontal skip, which has the natural width of an interword space and the negative stretchability of the first skip. It creates identical output to the version implemented in plain TeX with `\raggedright`. (At the time [7] was written, `\rightskip` had not yet been invented [10, p. 154].)

A zone can also be defined using natural width and shrinkability of the same size for `\rightskip`. However, because glue cannot shrink more than the stated value, a `\tolerance` greater than 100 is without effect; thus there is only one zone width. Another disadvantage is that lines with less content have smaller badness values: TeX prefers lines with lengths near the left boundary of the zone.

TeX prefers filled lines if the stretchability of the glue `\rightskip` defines the zone. The length of the line represents its badness: A line that reaches the left boundary of the zone has a higher badness than a line that needs all of `\hsize`. Is it good to treat them differently as both lines have made it into the zone? Let's assume that the tolerance is 0 so that all lines reaching the zone have equal badness.

If the heuristic to calculate the badness β for a line that stretches is interpreted as an equation

$$\beta = 100\left(\frac{\text{used stretchability}}{\text{available stretchability}}\right)^3 \qquad (3)$$

then we want to have β at most 0.49 so that it rounds down to 0. The available stretchability, that is, the stretchability of `\rightskip`, is then the maximal used stretchability, i.e., the zone width, divided by $\sqrt[3]{0.49/100} \approx 0.17$. Thus $1/0.17 = 5.8875$ times the intended zone width has to be assigned to the stretchability of `\rightskip`. For example, a zone width of 20 pt needs a stretchability for `\rightskip` of 117.75 pt. It's hard to see any advantage compared to simply setting `\pretolerance` $= 100$ and a direct assignment of the zone width to the stretchability of `\rightskip`. Several other parameters like penalties and additional demerits should be changed too but it isn't clear how to assign values to them to fix the situation. And if we think of the theory about the sticking-out problem then the scenario of example 2 becomes common.

My implementation defines the zone via the stretchability of `\rightskip` with a `\pretolerance` of 100 as it seems to be the best idea. Let's assume that the (temporary) dimen registers `\rrTd` and `\rrTdd` contain the width of the zone for the first pass and the width of the zone for the second

In olden times when wishing
still helped one, there lived a
king whose daughters were all
beautiful, but the youngest was
so beautiful that the sun itself,
which has seen so much, was as-
tonished whenever it shone in her
face. Close by the king's castle
lay a great dark forest, and un-
der an old lime-tree in the forest
was a well, and when the day was
very warm, the king's child went
out into the forest and sat down
by the side of the cool fountain,
and when she was bored she took
a golden ball, and threw it up on
high and caught it, and this ball
was her favorite plaything.

In olden times when wishing
still helped one, there lived a
king whose daughters were all
beautiful, but the youngest was
so beautiful that the sun itself,
which has seen so much, was as-
tonished whenever it shone in her
face. Close by the king's castle
lay a great dark forest, and under
an old lime-tree in the forest was
a well, and when the day was
very warm, the king's child went
out into the forest and sat down
by the side of the cool fountain,
and when she was bored she took
a golden ball, and threw it up on
high and caught it, and this ball
was her favorite plaything.

In olden times when wishing
still helped one, there lived a
king whose daughters were all
beautiful, but the youngest was
so beautiful that the sun itself,
which has seen so much, was as-
tonished whenever it shone in her
face. Close by the king's castle
lay a great dark forest, and un-
der an old lime-tree in the forest
was a well, and when the day
was very warm, the king's child
went out into the forest and sat
down by the side of the cool foun-
tain, and when she was bored she
took a golden ball, and threw
it up on high and caught it, and
this ball was her favorite play-
thing.

Figure 1: a) `\raggedright` & `\noindent`; b) and `\adjdemerits` = −10000; c) and `\adjdemerits` = −40000.

pass or 0 pt, respectively. The second case is used if the default value 200 is used for the `\tolerance`.

```
\pretolerance=100 % define zone width of 1st pass
\advance\rightskip by 0pt plus \rrTd minus 0pt
\ifdim\rrTdd=0pt % use default \tolerance
  \tolerance=200
\else
  \ifdim\rrTd>\rrTdd \errhelp=\rrEbadtolerance
    \errmessage{Width for 2nd pass must not be
                smaller than for 1st}%
    \tolerance=100
  \else % set \tolerance for 2nd zone width
    \rrHcompbad\rrTd=100(\rrTdd/\rrTd)^3
    \rrHassign\tolerance=[\rrTd]
\fi\fi
```

The code uses two support macros directly and then two more are called. Two count registers `\rrTc` and `\rrTcc` appear in the code snippet.

```
\def\rrHassign#1=[#2]{% #1: count register
  % #1 gets integer part of (dimen #2/pt)
  \expandafter#1\expandafter
  \rrHinteger\the#2\end\ignorespaces}
\def\rrHcompbad#1=100(#2/#3)^3{%
  % #1: dimen register; #2, #3: dimens
  #1=#3\relax \rrHassign\rrTc=[#1]
  #1=#2\relax \rrHassign\rrTcc=[#1]
  % \rrTc=int(#3/pt) and \rrTcc=int(#2/pt)
  #1=\rrTcc pt % #1=int(#2/pt)pt, short: #1=#2
  \rrHmuldiv#1*\rrTcc/\rrTc\end % #1=#2^2/#3
  \rrHmuldiv#1*\rrTcc/\rrTc\end % #1=#2^3/#3^2
  \rrHmuldiv#1*100/\rrTc\end % #1=100*#2^3/#3^3
  \advance#1 by 0.5pt \ignorespaces}% round up
```

The next two support macros have simple implementations.

```
\def\rrHinteger#1.#2\end{#1\ignorespaces}
\def\rrHmuldiv#1*#2/#3\end{%
  % #1: dimen register; #2, #3: numbers
  \divide #1 by #3\relax
  \multiply #1 by #2\ignorespaces}
```

R2 "flutter". If we want to have lines of different length then we want lines of different badness. A short line has a large badness, a long line gets a badness near 0. So we want very loose lines to be followed by decent lines and vice versa. The parameter `\adjdemerits` with value 10000 tries to guide TeX's line-breaking algorithm to avoid such combinations, that is, we need to change its value. Although the value of −10000 makes TeX work harder to find the best line breaks it seems to be worth assigning a negative value in order to support this recommendation. The paragraph in Fig. 1b) has one less hyphenated line end. Don't overreact: smaller values might give worse results for this aspect as Fig. 1c) shows.

An assignment to `\rightskip` for R1 "zone" is applied to all lines of a paragraph. There is no way to change it for odd and even line numbers. If the zone is implemented via active spaces something can be implemented to change the stretchability. Unfortunately, it is impossible to know where the line breaks occur, so whatever scheme is implemented for the switch it is guesswork.

Another possibility is to change `\hsize`. Well, this parameter is fixed for all lines of a paragraph like the `\rightskip`. But TeX has the command `\parshape` which can be used to define independent line lengths. Of course, the maximal number of lines

In olden times when wishing
still helped one, there lived a
king whose daughters were all
beautiful, but the youngest was
so beautiful that the sun itself,
which has seen so much, was as-
tonished whenever it shone in her
face. Close by the king's castle
lay a great dark forest, and un-
der an old lime-tree in the forest
was a well, and when the day was
very warm, the king's child went
out into the forest and sat down
by the side of the cool fountain,
and when she was bored she took
a golden ball, and threw it up
on high and caught it, and this
ball was her favorite plaything.

In olden times when wishing
still helped one, there lived a
king whose daughters were all
beautiful, but the youngest was
so beautiful that the sun itself,
which has seen so much, was as-
tonished whenever it shone in
her face. Close by the king's cas-
tle lay a great dark forest, and
under an old lime-tree in the for-
est was a well, and when the
day was very warm, the king's
child went out into the forest
and sat down by the side of the
cool fountain, and when she was
bored she took a golden ball, and
threw it up on high and caught
it, and this ball was her favorite
plaything.

In olden times when wishing still
helped one, there lived a king
whose daughters were all beau-
tiful, but the youngest was
so beautiful that the sun itself,
which has seen so much, was
astonished whenever it shone in
her face. Close by the king's
castle lay a great dark forest, and
under an old lime-tree in the
forest was a well, and when the
day was very warm, the king's
child went out into the forest and
sat down by the side of the cool
fountain, and when she was bored
she took a golden ball, and
threw it up on high and caught
it, and this ball was her favorite
plaything.

Figure 2: a) short is `\hsize` $-$ 6 pt b) and start short; c) short is `\hsize` $-$ 9 pt.

that the paragraphs of the text will reach must be set in advance to have different line lengths in every paragraph. Therefore a limit is declared that a user can increase to cover all number of lines in the paragraphs. As the `\parshape` command is reset after each paragraph [8, p. 103], its specification must be activated for each paragraph by `\everypar`.

But if we have a fixed parshape that, for example, indents the first line there is no chance to start a paragraph without indentation. Therefore the parshape specification must be built anew for each paragraph. Besides the indentation two other aspects should be made flexible. First, it should be possible to switch off the specification and to use `\hsize` for all lines. This helps to avoid the sticking-out problem as shown in example 2. Second, the sequence of alternate line lengths should be able to start with a short line instead of a long line. This might give more choices for the first line break.

Figure 2 shows three outputs with different line lengths for odd and even numbered lines. It uses plain TeX's `\raggedright`. The figure adds only in each case a parshape specification. Thus the observed zone width is increased by the amount that short lines are shorter than long lines.

My implementation consists of two parts. As mentioned above the parshape specification must be built anew for every paragraph because several aspects should be covered: (1) indent or not, (2) begin with a long or a short line, and (3) use only long lines, i.e., create a neutral `\parshape`.

Let `\rrCmax` be a constant that is larger than the number of lines of any paragraph in the text. We need some parts to build the lines for `\parshape`: `\rrPl` is a single long line without indent, i.e., it is `0pt \hsize`, `\rrPil` is an indented long line, `\rrPs` is a short line without indent, and `\rrPis` is short and indented. Two blocks of line pairs, each of them with `\rrCmax`/2−2 pairs, are built for long and short line specifications and for long lines only. The first block is named `\rrPpair`, the second `\rrPlong`.

The creation of the parshape parts is not complicated. Assume that `\rrTd` contains the amount by which the short lines are shorter than `\hsize` and `\rrTdd` contains the value for the indentation. `\rrTddd` is a third dimen register.

```
\rrTddd=\hsize              % create long lines
\edef\rrPl{0pt \the\rrTddd}%         no indent
\advance\rrTddd by -\rrTdd
\edef\rrPil{\the\rrTdd\space\the\rrTddd}% indent
\rrTddd=\hsize \advance\rrTddd by -\rrTd
\edef\rrPs{0pt \the\rrTddd}%  short, no indent
\advance\rrTddd by -\rrTdd
\edef\rrPis{\the\rrTdd\space\the\rrTddd}% indent
```

The two blocks of long/short and long/long line pairs are created via a loop.

```
\edef\rrPpair{\rrPl\space\rrPs}% pair long/short
\edef\rrPlong{\rrPl\space\rrPl}%  two long lines
\rrTc=\rrCmax \advance\rrTc by -4
\loop
\ifnum\rrTc>0 \advance\rrTc by -2
  \edef\rrPpair{\rrPpair\space
                \rrPl\space\rrPs}% add a pair
  \edef\rrPlong{\rrPlong\space
                \rrPl\space\rrPl}% here too
\repeat
```

The macro that builds the parshape specification makes use of three flags for the above mentioned cases: `\ifrrInoi` which is true if no indentation is wanted, `\ifrrIsl` which is true if the first line is short, and finally `\ifrrIlongonly` which is true if the alternate line lengths shall be ignored.

```
\def\rrMparshape{%
\begingroup              % needed for \aftergroup
  \aftergroup\parshape % start \parshape
  \aftergroup\rrCmax   % number of lines
  \aftergroup\space
  \ifrrIlongonly       % check for one length
    \ifrrInoi          % check for indent
      \aftergroup\rrPl % no indent
    \else
      \aftergroup\rrPil% indent
    \fi
    \aftergroup\rrPl   % the other lines
    \aftergroup\rrPlong
  \else
    \ifrrInoi          % check for indent
      \ifrrIsl         % check start short
      \else
        \aftergroup\rrPl
      \fi
      \aftergroup\rrPs % pair if started long
    \else
      \ifrrIsl         % indent short
        \aftergroup\rrPis
      \else            % indent long, a pair
        \aftergroup\rrPil
        \aftergroup\rrPs
    \fi\fi
    \aftergroup\rrPpair% add a lot of lines
    \ifrrIsl
      \aftergroup\rrPl % complete a short start
  \fi\fi
\endgroup}
```

This macro is invoked for each paragraph. (A box from `\indent` is removed using `\lastbox`.)

```
\everypar={{\setbox0=\lastbox}\rrMparshape}%
```

R3: extended zone. The parameter `\hfuzz` might be used to allow TeX to go beyond `\hsize` [8, p. 30]. The command does not increase the line length; it merely suppresses an overfull-line warning as long as the line sticks out less than the width given by `\hfuzz`. Thus this parameter is only used if TeX isn't able to typeset the text without error; its value is not available for TeX's decisions and that is not exactly what is recommended. The dimen `\hfuzz` should be kept as an emergency tool for the user and not be used in a setup.

If the `\parshape` command is used then the length of the long lines can be changed independently from the line length of the shorter lines. But then TeX tries to fill all long lines beyond `\hsize` as this gives the lowest badness. Again this is not exactly what is recommended.

The third idea avoids these disadvantages: Add shrinkability to `\rightskip`. If a line uses its natural width or stretches then nothing is changed. But now a line can shrink too. The shrinkability r^- of the `\rightskip` adds to the total line length if the line shrinks. Long lines can therefore reach the length `\hsize` $+ r^-$. Of course, r^- is added for long and short lines if a parshape specification is used.

Figure 3 on the next page applies $r^- = 2\,\text{pt}$ and $r^- = 4\,\text{pt}$ to previously shown examples.

My implementation changes, i.e., advances, `\rightskip` with shrinkability. If the dimen register `\rrTd` contains the amount by which lines shall be extended then the following assignment does the job:

```
\advance\rightskip by 0pt plus 0pt minus \rrTd
```

Such simple assignments to TeX parameters are not shown anymore in the following subsections; the description in plain English should be sufficient.

R4: no shape. The first three recommendations are technical, thus a translation into TeX macros and parameters was not too difficult, although a robust proof of their usefulness is lacking. However, the fourth recommendation is highly subjective.

Using long and short lines seems to avoid most shapes if the line lengths are sufficiently different. But, for example, the sequence of a short short, a short long, a long short, and a long long line might create the unwanted case of four lines with increasing length. More complicated designs for `\parshape` might avoid this effect. Or a negative `\adjdemerits` can trade the effect for words that stick out.

R5: short words at the end of a line. This situation cannot be avoided in all cases. It was shown that plain TeX's `\raggedright` doesn't like words that stick out. And if it happens, a neutral parshape specification might help; such a specification is provided in the implementation of R2 "flutter".

But with lines of different length the conditions are changed and short words might appear at the end of long lines. This sometimes gets fixed if the sequence of long/short pairs is changed and the paragraph is started with a short line. The above stated implementation of R2 "flutter" allows this.

Of course, one can tie short words to their successors. But that should only be done to fix bad breaks.

R6: last line. It is well known that the last line receives TeX's `\parfillskip`; by default it is `0pt plus 1fil` [8, p. 30]. But that is not acceptable if

In olden times when wishing still
helped one, there lived a king
whose daughters were all beau-
tiful, but the youngest was so
beautiful that the sun itself, which
has seen so much, was astonished
whenever it shone in her face.
Close by the king's castle lay a
great dark forest, and under an
old lime-tree in the forest was a
well, and when the day was very
warm, the king's child went out
into the forest and sat down by
the side of the cool fountain, and
when she was bored she took a
golden ball, and threw it up on
high and caught it, and this ball
was her favorite plaything.

In olden times when wishing still
helped one, there lived a king
whose daughters were all beau-
tiful, but the youngest was so
beautiful that the sun itself, which
has seen so much, was astonished
whenever it shone in her face.
Close by the king's castle lay a
great dark forest, and under an
old lime-tree in the forest was a
well, and when the day was very
warm, the king's child went out
into the forest and sat down by
the side of the cool fountain, and
when she was bored she took
a golden ball, and threw it up on
high and caught it, and this ball
was her favorite plaything.

In olden times when wishing still
helped one, there lived a king
whose daughters were all beau-
tiful, but the youngest was
so beautiful that the sun itself,
which has seen so much, was as-
tonished whenever it shone in her
face. Close by the king's castle
lay a great dark forest, and under
an old lime-tree in the forest
was a well, and when the day was
very warm, the king's child went
out into the forest and sat down
by the side of the cool fountain,
and when she was bored she took
a golden ball, and threw it up
on high and caught it, and this
ball was her favorite plaything.

Figure 3: a) Fig. 1a) with $r^- = 2\,\text{pt}$; b) Fig. 1b) with $r^- = 4\,\text{pt}$; c) Fig. 2c) with $r^- = 2\,\text{pt}$.

the last line should be much shorter, in order to be easily distinguished from the left boundary of the zone if the start of a paragraph isn't signaled by indentation or other methods. The natural width of `\parfillskip` can be set to the zone width of the second pass plus the difference of the widths of long and short lines, w, and at least 2 em:

$$\sqrt[3]{\frac{\texttt{\textbackslash tolerance}}{100}} \times r^+ + w + 2\,\text{em}. \qquad (4)$$

The recommendation states more precisely that the last line shall be 2 to 4 em shorter than the other lines if the `\hsize` allows this. Instead of using only the natural width of `\parfillskip` it seems better to add 4 em instead of 2 em in (4) and to use a shrinkability of 2 em. This assigns some demerits if the last line is only 2 em shorter so TeX more likely avoids such a short difference.

Of course, this setting might produce overfull lines. But this is a good indicator that something must be fixed by the author. And it is easy to enter `\parfillskip=0pt plus 1fil` at the end of a paragraph to restore the default behavior.

Another common problem is that a new line with only a word part is created. A large value for `\finalhyphendemerits` might fix such a situation.

Figures 4a) and 4b) on the next page apply the ideas. The first column uses `\raggedright` with `\parfillskip=65pt plus 1fil minus 20pt` and the next column adds `\finalhyphendemerits=50000`.

R7: no indentation for small `\hsize`. The plain TeX macro keeps TeX's default behavior for indentation of the first line of a paragraph. To change that either the `\parindent` is set to 0 pt or all of the paragraphs are started with a `\noindent`. A parshape specification similar to R2 "flutter" sets the indentation for all paragraphs to a given parameter and provides with a Boolean flag a way to start a paragraph without indentation. So the author must make the decision either not to indent all paragraphs or to manually enter a command to suppress the indentation.

As the test paragraph is typeset without indentation, Fig. 4c) goes the other way and uses an indentation of 20 pt with plain TeX's `\raggedright`.

My implementation invokes the above mentioned Boolean flag to avoid indentation:

```
\let\rrnoindent=\rrInoitrue
```

R8: fixed interword space. As mentioned above, interword spaces are usually built from `\fontdimen` parameters that TeX reads from the font metric file of the font, while a nonzero `\spaceskip` overrides these font parameters as glue specification of interword spaces and a nonzero `\xspaceskip` defines the interword space if the `\spacefactor` is ≥ 2000. Both parameters store a glue specification with three components for the natural width, the stretchability, and the shrinkability. But in ragged-right typesetting only the natural width shall be used.

Of course, the interword spaces should usually not deviate much from the font designer's setting documented in `\fontdimen2`. Note that a specification in the unit "em" involves `\fontdimen6`, the width of the font's quad [8, p. 60].

The recommendation requests a single width for interword spaces, that is, the value of `\xspaceskip` should stay at 0 pt.

In olden times when wishing
still helped one, there lived a
king whose daughters were all
beautiful, but the youngest was
so beautiful that the sun itself,
which has seen so much, was as-
tonished whenever it shone in
her face. Close by the king's cas-
tle lay a great dark forest, and
under an old lime-tree in the for-
est was a well, and when the day
was very warm, the king's child
went out into the forest and sat
down by the side of the cool foun-
tain, and when she was bored she
took a golden ball, and threw it
up on high and caught it, and
this ball was her favorite play-
thing.

In olden times when wishing
still helped one, there lived a
king whose daughters were all
beautiful, but the youngest was
so beautiful that the sun itself,
which has seen so much, was as-
tonished whenever it shone in
her face. Close by the king's cas-
tle lay a great dark forest, and
under an old lime-tree in the
forest was a well, and when the
day was very warm, the king's
child went out into the forest
and sat down by the side of the
cool fountain, and when she was
bored she took a golden ball, and
threw it up on high and caught
it, and this ball was her favorite
plaything.

In olden times when wish-
ing still helped one, there lived
a king whose daughters were all
beautiful, but the youngest was
so beautiful that the sun itself,
which has seen so much, was as-
tonished whenever it shone in her
face. Close by the king's castle
lay a great dark forest, and un-
der an old lime-tree in the forest
was a well, and when the day was
very warm, the king's child went
out into the forest and sat down
by the side of the cool fountain,
and when she was bored she took
a golden ball, and threw it up on
high and caught it, and this ball
was her favorite plaything.

Figure 4: a) changed `\parfillskip`; b) and `\finalhyphendemerits`; c) Fig. 1a) without `\noindent`.

R9: hyphenation. In case a) normal hyphenation rules are applied, so we can concentrate on R9b). It seems that the code of TEX must be changed to get "sense-conveying hyphenation" (see [6]), so this cannot be implemented by macros.

The assignment of 10000 to the two parameters `\hyphenpenalty` and `\exhyphenpenalty` switches off hyphenation. Smaller values define the cost of hyphens and thus their desirability. The default value for both parameters is 50 [8, pp. 96–97].

The first parameter sets the penalty that TEX uses in the formula for the line demerits [8, p. 98] if the break occurs either at a hyphen that was inserted by TEX or at a discretionary break in the text that the author has entered with `\-`. The second parameter stands for the penalty that a break receives if it occurs after a hyphen or a sequence of hyphens inside the text, i.e., hyphens entered by an author as `-`, `--`, or `---`. In order to be able to influence hyphenation the value for `\hyphenpenalty` must be less than 10000. Then an author can insert discretionary breaks to improve the output. Hyphenation can also be influenced by changing the values of the parameters `\doublehyphendemerits` and `\finalhyphendemerits` [8, p. 98].

If a word is given to TEX's hyphenation algorithm it first checks that it is a word of length $\geq$ `\lefthyphenmin` + `\righthyphenmin`. Such a word might be split into parts of at least `\lefthyphenmin` letters on the first line and `\righthyphenmin` letters on the second. In case R9b) it is recommended to assign one of the values $(4, 4)$, or $(4, 5)$, or $(5, 5)$ to the pair (`\lefthyphenmin`,`\righthyphenmin`) so that only words that have more than eight, nine, or ten letters get hyphenated; see [8, p. 454].

Figure 5 on the next page shows the effects of some of these parameters.

R10: emergency. The two methods that are suggested can be easily realized by TEX macros. The shrinkability of interword spaces can be used to implement the emergency shrink. Two assignments to `\lefthyphenmin` and `\righthyphenmin` change the length of word parts during hyphenation (see R9). And the default `\parfillskip` can be used.

Above, the dimension parameter `\hfuzz` was explained; it shall be left to the author for an emergency. And `\emergencystretch` is another option for problematic situations. As interword spaces cannot stretch it extends the width of the zone.

My implementation makes one change permanent; the other works only for a single paragraph. The macro `\rremergencyshrink` is named similar to the macro `\emergencystretch` and makes a permanent change. The help from a less restricted hyphenation is for a single paragraph only; an author might not know how to switch back to the settings that were active before.

Let's assume that the current settings of the parameters `\spaceskip` and `\xspaceskip` were captured in the macro `\rrMresetspaces`. The macro `\rremergencyshrink` is implemented in two simple steps. First, `\rrTd` gets the value by which the interword spaces shall shrink.

```
\def\rremergencyshrink{% shrink interword spaces
  \afterassignment\rrMshrinkspaces \rrTd}
```

In olden times when wishing
still helped one, there lived a
king whose daughters were all
beautiful, but the youngest was
so beautiful that the sun itself,
which has seen so much, was
astonished whenever it shone
in her face. Close by the king's
castle lay a great dark forest,
and under an old lime-tree in the
forest was a well, and when the
day was very warm, the king's
child went out into the forest
and sat down by the side of the
cool fountain, and when she was
bored she took a golden ball, and
threw it up on high and caught
it, and this ball was her favorite
plaything.

In olden times when wishing
still helped one, there lived a
king whose daughters were all
beautiful, but the youngest was
so beautiful that the sun itself,
which has seen so much, was
astonished whenever it shone
in her face. Close by the king's
castle lay a great dark forest,
and under an old lime-tree in the
forest was a well, and when the
day was very warm, the king's
child went out into the forest and
sat down by the side of the cool
fountain, and when she was bored
she took a golden ball, and threw
it up on high and caught it, and
this ball was her favorite play-
thing.

In olden times when wishing
still helped one, there lived a
king whose daughters were all
beautiful, but the youngest was
so beautiful that the sun itself,
which has seen so much, was
astonished whenever it shone in
her face. Close by the king's
castle lay a great dark forest, and
under an old lime-tree in the
forest was a well, and when the
day was very warm, the king's
child went out into the forest and
sat down by the side of the cool
fountain, and when she was bored
she took a golden ball, and
threw it up on high and caught
it, and this ball was her favorite
plaything.

Figure 5: Fig. 1a) and a) `\hyphenpenalty` = 10000; b) hyphenmins $(4, 4)$; c) Fig. 2c) with hyphenmins $(5, 5)$.

The second step can now test `\rrTd`; a negative value is not allowed and it must not be larger than 0.015 em; `\rrHabsdim` makes the value positive and gives a warning if a negative value occurs. We have to be careful not to apply the macro more than once to `\spaceskip`. In the code, `\rrTs` is a skip register.

```
\def\rrMshrinkspaces{%
  \rrHabsdim\rrTd=[\rrTd] % absolute value
  \ifdim\rrTd>0.015em \errhelp\rrEbadshrink
    \errmessage{Interword spaces shouldn't
       shrink more than 0.015em}\rrTd=0.015em
  \fi
  \rrTs=0pt plus 0pt minus \rrTd
  \rrMresetspaces % only one change is allowed
  \ifdim\spaceskip>0pt
    \advance\spaceskip by \rrTs
    \ifdim\xspaceskip>0pt
       \advance\xspaceskip by \rrTs
  \fi\fi}
```

Here is the second macro `\rrusehyphenmins`; it changes `\lefthyphenmin` and `\righthyphenmin`:

```
\def\rrusehyphenmins=(#1,#2){% change hyphenmins
% #1: lefthyphenmin; #2: righthyphenmin;
  \rrMdefaulthyphenmins % reset to initial values
  \ifnum#1<\lefthyphenmin
    \message{^^J\string\lefthyphenmin\space
             is not changed}%
  \else \lefthyphenmin=#1\relax \fi
  \ifnum#2<\righthyphenmin
    \message{^^J\string\righthyphenmin\space
             is not changed}%
  \else \righthyphenmin=#2\relax \fi
  \ignorespaces}
```

The code uses `\rrMdefaulthyphenmins`; a new macro that sets the values of `\lefthyphenmin` and `\righthyphenmin` back to the settings defined for the current language. In order to switch back to the values set by the ragged-right scheme a reset macro for the two “hyphenmins” is also coded. It is named `\rrMresethyphenmins`.

Addendum. Several times we saw that parameters have to be reset or tested for each paragraph. Thus, as the last code snippet, the new implementation of `\par` is presented.

```
\def\par{\endgraf % end the paragraph
  \ifnum\prevgraf>\rrCmax % test for more lines
    \message{^^JRagged right needs more lines:
            \the\prevgraf^^J}\fi
  \rrMresethyphenmins % reset parameters & flags
  \rrInoifalse \rrIslfalse \rrIlongonlyfalse
  \rrMresetparfillskip % see section 6
  \rrMresetparskip}%     see section 8
```

R2 “flutter” uses a constant `\rrCmax` for the maximum number of lines that a paragraph should have. At the end of each paragraph this is now checked and a violation is reported.

6 A generator for ragged-right macros

Except for R2 “flutter” and R1 “zone” — if the zone widths instead of the tolerance values shall be processed — the recommendations that can be implemented need only one or two assignments to TeX's parameters. That many parameters influence the typesetting of `\raggedright` is documented by the 15 different outputs shown in Figs. 1 to 5. So one can combine the assignments into a single macro with a similar structure as `\raggedright`.

But for my experiments I find it much more convenient to have a kind of input mechanism that assigns default values to unused parameters. Then the switch to a new scheme resets all values and only the defined ones affect the new scheme. In total eight macros are called: (1) to set the zone for 1st and 2nd pass, (2) to create the macros used to build the `\parshape` command with indent and the amount by which short lines are reduced, (3) to add natural width and shrinkability to `\parfillskip`, (4) to make the interword spaces fixed width, (5) to assign the minimal length of word parts for hyphenation, (6) to set the two penalties and (7) the two additional demerits parameters that are involved in hyphenation, and (8) to support the typesetting by changing `\adjdemerits` and by declaring an overshoot. The main macro has the following structure:

```
\def\defraggedrightscheme #1: #2\end{%
% #1: name of the scheme; #2: parameter set
 \let\rrMresetparskip=\relax     % make names
 \let\rrMresetparfillskip=\relax % known for
 \let\rrMresethyphenmins=\relax  % \par
 \expandafter\def\csname #1\endcsname{% scheme
  \rightskip = 0pt plus 0pt minus 0pt % init
  % macros for the recommendations
  \rrMsetzone #2 zone 1st,2nd:(0pt,0pt) \end
  \rrMsetparshape #2 %
         flutter indent,short:(0pt,0pt) \end
  \rrMsetparfillskip #2 %
                   last nw,sh:(0pt,0pt) \end
  \rrMsetspaces #2 %
             space word,punct:(0pt,0pt) \end
  \rrMsetlrmins #2 %
            hyphenmins left,right:(0,0) \end
  \rrMsetpenalties #2 %
                 hyphenpens ex,im:(0,0) \end
  \rrMsetdemerits #2 %
               hyphendems dbl,fin:(0,0) \end
  \rrMsetsupport #2 %
                 aid exceed,adj:(0pt,0) \end
  % change \par; see ‘‘Addendum’’ in section 5
  \rrMenhancepar}}
```

The three macros used in the definition of `\par` (see the “Addendum” in section 5) get the meaning `\relax` so that `\par` can be executed. As unused parameters are reset to plain TeX’s defaults the call

```
\defraggedrightscheme justified:\end
```

creates `\justified` that ends TeX’s `\raggedright` and restores justified typesetting and

```
\defraggedrightscheme rrplain:
   zone 1st,2nd:(2em,0pt)
   space word,punct:(0.3333em,0.5em)\end
```

makes `\rrplain` equivalent to TeX’s `\raggedright`; only `\par` keeps its new coding. As a more complex example, here is the specification for Fig. 3c):

```
\defraggedrightscheme figIIIc:
   zone 1st,2nd:(2em,0pt)
   space word,punct:(0.3333em,0.5em)
   flutter indent,short:(0pt,9pt)
   aid exceed,adj:(2pt,10000)\end
```

It doesn’t seem to be necessary to show all eight macros. Larger parts of the code for the first two were presented above, so let’s concentrate on the third to explain the general idea.

```
\def\rrMsetparfillskip
     #1last nw,sh:(#2,#3) #4\end{% def last line
  % #1: some other parameters (not used here)
  % #2: natural width of \parfillskip
  % #3: its shrinkability
  % #4: like #1 or empty: then use defaults
  \def\next{#4}\ifx\next\empty
    \parfillskip = 0pt plus 1fil minus 0pt
  \else % use only positive dimen parameters
    \rrHabsdim\rrTd=[#2] \rrHabsdim\rrTdd=[#3]
    \parfillskip = \rrTd plus 1fil minus \rrTdd
  \fi
  \edef\rrMresetparfillskip{% to the new setting
         \parfillskip=\the\parfillskip}}
\def\rrplainsparend{\parfillskip 0pt plus 1fil
  \par}% omit new specification in an emergency
```

If the `\parfillskip` isn’t set in a ragged-right scheme, i.e., there is no “last nw,sh:(x, y)” in its definition, then #4 is empty as the keywords of the `\def` are matched by the strings given at the end of `\rrMsetparfillskip` in the code of the above listed macro `\defraggedrightscheme`. Otherwise this instance must occur in #4 and the statements in the `\else` are executed. So #2 and #3 define the natural width and the shrinkability of `\parfillskip`.

The macro `\rrMresetparfillskip` assigns the values of the scheme for `\parfillskip` with every `\par` (see “Addendum” in section 5).

7 Experiments

As one might expect, no parameter setting produces the ideal that the ten recommendations describe. Not only that some cannot be mapped into TeX macros, the developed macros cannot guarantee that their output avoids all shapes, words that stick out, etc. Moreover, other problems remain, for example, overfull lines might occur that must be fixed. Hence the author must accept compromises.

Thus our interest must be to reduce manual adjustments as much as possible. This section concentrates on the macros; only a few ties are used. Manual line breaks are discussed in section 8.

In olden times when wishing still helped one, there
lived a king whose daughters were all beautiful,
but the youngest was so beautiful that the sun it-
self, which has seen so much, was astonished when-
ever it shone in her face. Close by the king's cas-
tle lay a great dark forest, and under an old lime-
tree in the forest was a well, and when the day was
very warm, the king's child went out into the for-
est and sat down by the side of the cool fountain,
and when she was bored she took a golden ball,
and threw it up on high and caught it, and this
ball was her favorite plaything.

Figure 6: TeX's `\raggedright` and `\noindent`.

Okay, let's start with examples. *TUGboat*'s columns have a width of 225 pt. The `\raggedright` macro together with a `\noindent` produces for our single paragraph the output shown in Fig. 6. It is clear that too many hyphens occur for such a wide measure and too many of them appear in a sequence.

In a new ragged-right scheme for *TUGboat* I keep the width of the zone and the width of the interword spaces, including the larger space after certain punctuation marks. To make the lines flutter I introduce short lines which get the width `\hsize` − 5 pt. As the measure is wide but indentation suppressed the last line should be much shorter than any short line. Finally, hyphenation shall only be applied to words with at least eight characters and even then it should cost more than in plain TeX. Moreover, consecutive hyphenated lines and a hyphenated second last line shall be very expensive. As the text is mixed with justified paragraphs I don't apply the *exceed* parameter.

Thus I use the following scheme for *TUGboat*:

```
\defraggedrightscheme secVII:
   zone 1st,2nd:(2em,0pt)
   space word,punct:(0.3333em,0.5em)
   flutter indent,short:(0pt,0.5em)
   last nw,sh:(6.5em,2em)
   hyphenmins left,right:(4,4)
   hyphenpens ex,im:(100,200)
   hyphendems dbl,fin:(20000,50000)\end
```

Figure 7 shows the output if the test paragraph is typeset with this scheme: With the same number of lines all hyphens disappear and more lines approximate the right margin. But there is no proper alternation of line lengths.

Whenever I see this effect I first try to start with a short line if "flutter" is specified in the ragged-right scheme: Figure 8a) shows the result. The line lengths are varying a little bit better and more lines are sufficiently filled; but new hyphens occur. As

In olden times when wishing still helped one, there
lived a king whose daughters were all beautiful,
but the youngest was so beautiful that the sun
itself, which has seen so much, was astonished
whenever it shone in her face. Close by the king's
castle lay a great dark forest, and under an old
lime-tree in the forest was a well, and when the
day was very warm, the king's child went out
into the forest and sat down by the side of the
cool fountain, and when she was bored she took a
golden ball, and threw it up on high and caught it,
and this ball was her favorite plaything.

Figure 7: Using the scheme `\secVII`.

specified, the word parts have at least four characters. The new hyphen in the second last line can only be avoided if a `\break` is inserted somewhere as it is the only way to typeset the paragraph with this scheme. A change to `\parfillskip` can change this with the risks that a lot of new line breaks occur and that the start of the next paragraph is not easily identified. Figure 8b) adds `\rrplainsparend` (see section 6) at the end of the paragraph. It lengthens the last line, changes all but five line breaks, and creates a single-letter word that sticks out.

In olden times when wishing still helped one,
there lived a king whose daughters were all beau-
tiful, but the youngest was so beautiful that the
sun itself, which has seen so much, was aston-
ished whenever it shone in her face. Close by the
king's castle lay a great dark forest, and under
an old lime-tree in the forest was a well, and
when the day was very warm, the king's child
went out into the forest and sat down by the
side of the cool fountain, and when she was bored
she took a golden ball, and threw it up on high
and caught it, and this ball was her favorite play-
thing.

a) The scheme `\secVII` and `\rrstartshort`.

In olden times when wishing still helped one,
there lived a king whose daughters were all beau-
tiful, but the youngest was so beautiful that the
sun itself, which has seen so much, was aston-
ished whenever it shone in her face. Close by the
king's castle lay a great dark forest, and under an
old lime-tree in the forest was a well, and when
the day was very warm, the king's child went out
into the forest and sat down by the side of the
cool fountain, and when she was bored she took a
golden ball, and threw it up on high and caught
it, and this ball was her favorite plaything.

Figure 8: b) Like a) but with the default `\parfillskip`.

Der Froschkönig
oder der eiserne Heinrich
In den alten Zeiten, wo das
Wünschen noch geholfen hat,
lebte ein König, dessen Töchter
waren alle schön, aber die
jüngste war so schön, daß die
Sonne selber, die doch so vieles
gesehen hat, sich verwunderte
sooft sie ihr ins Gesicht schien.
Nahe bei dem Schlosse des Königs
lag ein großer dunkler Wald,
und in dem Walde unter einer
alten Linde war ein Brunnen:
wenn nun der Tag sehr heiß war,
so ging das Königskind hinaus
in den Wald und setzte sich an
den Rand des kühlen Brunnens:
und wenn sie Langeweile hatte,
so nahm sie eine goldene Kugel,
warf sie in die Höhe und fing sie
wieder; und das war ihr liebstes
Spielwerk.
Nun trug es sich einmal zu, daß
die goldene Kugel der Königs-
tochter nicht in ihr Händchen
fiel, das sie in die Höhe gehalten
hatte, sondern vorbei auf die
Erde schlug und geradezu ins
Wasser hineinrollte. Die Königs-
tochter folgte ihr mit den Augen
nach, aber die Kugel verschwand,
und der Brunnen war tief, so
tief, daß man keinen Grund sah.
Da fing sie an zu weinen und
weinte immer lauter und konnte
sich gar nicht trösten. Und wie
sie so klagte, rief ihr jemand zu
'was hast du vor, Königstochter,
du schreist ja daß sich ein Stein
erbarmen möchte.' Sie sah sich
um, woher die Stimme käme, da
erblickte sie einen Frosch, der
seinen dicken häßlichen Kopf
aus dem Wasser streckte. 'Ach,
du bists, alter Wasserpatscher,'
sagte sie, 'ich weine über meine
goldene Kugel, die mir in den
Brunnen hinabgefallen ist.' 'Sei
still und weine nicht,' antwortete
der Frosch, 'ich kann wohl
Rat schaffen, aber was gibst du
mir, wenn ich dein Spielwerk
wieder heraufhole?' 'Was du
haben willst, lieber Frosch,'
sagte sie, 'meine Kleider, meine
Perlen und Edelsteine, auch
noch die goldene Krone, die ich
trage.' Der Frosch antwortete
'deine Kleider, deine Perlen und
Edelsteine, und deine goldene
Krone, die mag ich nicht: aber
wenn du mich lieb haben willst,
und ich soll dein Geselle und
Spielkamerad sein, an deinem
Tischlein neben dir sitzen, von
deinem goldenen Tellerlein essen,
aus deinem Becherlein trinken,
in deinem Bettlein schlafen:
wenn du mir das versprichst,
so will ich hinuntersteigen und
dir die goldene Kugel wieder
heraufholen.' 'Ach ja,' sagte
sie, 'ich verspreche dir alles, was
du willst, wenn du mir nur die
Kugel wiederbringst.' Sie dachte
aber 'was der einfältige Frosch
schwätzt, der sitzt im Wasser bei
seines Gleichen und quackt, und
kann keines Menschen Geselle
sein.'
Der Frosch, als er die Zusage
erhalten hatte, tauchte seinen
Kopf unter, sank hinab, und über
ein Weilchen kam er wieder
heraufgerudert; hatte die Kugel
im Maul und warf sie ins Gras.
Die Königstochter war voll
Freude, als sie ihr schönes Spiel-
werk wieder erblickte, hob es auf
und sprang damit fort. 'Warte,
warte,' rief der Frosch, 'nimm
mich mit, ich kann nicht so laufen
wie du.' Aber was half ihm,
daß er ihr sein quak quak so laut
nachschrie, als er konnte! Sie
hörte nicht darauf, eilte nach
Haus und hatte bald den armen
Frosch vergessen, der wieder in
seinen Brunnen hinabsteigen
mußte.
Am andern Tage, als sie mit dem
König und allen Hofleuten sich
zur Tafel gesetzt hatte und von
ihrem goldenen Tellerlein aß,
da kam, plitsch platsch, plitsch
platsch, etwas die Marmortreppe
heraufgekrochen, und als es oben
angelangt war, klopfte es an
der Tür und rief 'Königstochter,
jüngste, mach mir auf.' Sie lief
und wollte sehen wer draußen
wäre, als sie aber aufmachte, so
saß der Frosch davor. Da warf sie
die Tür hastig zu, setzte sich
wieder an den Tisch, und war
ihr ganz angst. Der König sah
wohl, daß ihr das Herz gewaltig
klopfte, und sprach 'mein Kind,
was fürchtest du dich, steht
etwa ein Riese vor der Tür
und will dich holen?' 'Ach nein,'
antwortete sie, 'es ist kein Riese,
sondern ein garstiger Frosch.'
'Was will der Frosch von dir?'
'Ach lieber Vater, als ich gestern
im Wald bei dem Brunnen saß
und spielte, da fiel meine goldene
Kugel ins Wasser. Und weil ich

Figure 9a: First part of the Brothers Grimm fairy tale "The Frog King" in German (see, e.g., [5, pp. 39–43]).

I wrote in the introduction that I developed the macros to typeset German texts. So the next experiment switches the language. The column width is again narrow (148 pt; 30 digits create an overfull line) as this is a typical case to use ragged-right typesetting. Figure 9 shows the typeset output (over two pages).

There are a couple of problems if a German text is typeset with plain TeX. First, the umlauts (like ä) must be entered with an accent and second the eszet (ß) is a control sequence; words with these letters cannot be hyphenated in plain TeX. Moreover, for hyphenation in general, the wrong patterns are loaded. I suppressed wrong hyphenations by declaring a few exceptions: zauderte, goldenes, goldenen, zweitenmal, and Spiel-kamerad. Three words with umlauts that occur frequently — Königskind (king's child), Königstochter (king's daughter), Königssohn

so weinte, hat sie der Frosch
wieder heraufgeholt, und weil
er es durchaus verlangte, so
versprach ich ihm, er sollte mein
Geselle werden, ich dachte aber
nimmermehr, daß er aus seinem
Wasser heraus könnte. Nun ist er
draußen und will zu mir herein.'
Indem klopfte es zum zweitenmal
und rief

'Königstochter, jüngste,
mach mir auf,
weißt du nicht, was gestern
du zu mir gesagt
bei dem kühlen Brunnenwasser?
Königstochter, jüngste,
mach mir auf.'

Da sagte der König 'was du
versprochen hast, das mußt du
auch halten; geh nur und mach
ihm auf.' Sie ging und öffnete
die Türe, da hüpfte der Frosch
herein, ihr immer auf dem Fuße
nach, bis zu ihrem Stuhl. Da saß
er und rief 'heb mich herauf zu
dir.' Sie zauderte bis es endlich
der König befahl. Als der Frosch
erst auf dem Stuhl war, wollte
er auf den Tisch, und als er da
saß, sprach er 'nun schieb mir
dein goldenes Tellerlein näher,
damit wir zusammen essen.' Das
tat sie zwar, aber man sah wohl,
daß sies nicht gerne tat. Der
Frosch ließ sichs gut schmecken,
aber ihr blieb fast jedes Bißlein
im Halse. Endlich sprach er
'ich habe mich satt gegessen
und bin müde, nun trag mich
in dein Kämmerlein und mach
dein seiden Bettlein zurecht,
da wollen wir uns schlafen
legen.' Die Königstochter fing
an zu weinen und fürchtete sich
vor dem kalten Frosch, den
sie nicht anzurühren getraute,
und der nun in ihrem schönen
reinen Bettlein schlafen sollte.
Der König aber ward zornig und
sprach 'wer dir geholfen hat, als
du in der Not warst, den sollst
du hernach nicht verachten.' Da
packte sie ihn mit zwei Fingern,
trug ihn hinauf und setzte ihn in
eine Ecke. Als sie aber im Bette
lag, kam er gekrochen und sprach
'ich bin müde, ich will schlafen
so gut wie du: heb mich herauf,
oder ich sags deinem Vater.' Da
ward sie erst bitterböse, holte
ihn herauf und warf ihn aus
allen Kräften wider die Wand,
'nun wirst du Ruhe haben, du
garstiger Frosch.'
Als er aber herabfiel, war er kein
Frosch, sondern ein Königssohn
mit schönen freundlichen Augen.
Der war nun nach ihres Vaters
Willen ihr lieber Geselle und
Gemahl. Da erzählte er ihr, er
wäre von einer bösen Hexe
verwünscht worden, und niemand
hätte ihn aus dem Brunnen
erlösen können, als sie allein, und
morgen wollten sie zusammen in
sein Reich gehen. Dann schliefen
sie ein, und am andern Morgen,
als die Sonne sie aufweckte, kam
ein Wagen herangefahren mit
acht weißen Pferden bespannt,
die hatten weiße Straußfedern
auf dem Kopf und gingen in
goldenen Ketten, und hinten
stand der Diener des jungen
Königs, das war der treue Hein-
rich. Der treue Heinrich hatte
sich so betrübt, als sein Herr
war in einen Frosch verwandelt
worden, daß er drei eiserne Bande
hatte um sein Herz legen lassen,
damit es ihm nicht vor Weh und
Traurigkeit zerspränge. Der
Wagen aber sollte den jungen
König in sein Reich abholen;
der treue Heinrich hob beide
hinein, stellte sich wieder hinten
auf und war voller Freude über
die Erlösung. Und als sie ein
Stück Wegs gefahren waren, hörte
der Königssohn, daß es hinter
ihm krachte, als wäre etwas
zerbrochen. Da drehte er sich
um und rief

'Heinrich, der Wagen bricht.'
'Nein, Herr, der Wagen nicht,
es ist ein Band von meinem
Herzen,
das da lag in großen Schmerzen,
als ihr in dem Brunnen saßt,
als ihr eine Fretsche (Frosch) wast
(wart).'

Noch einmal und noch einmal
krachte es auf dem Weg, und
der Königssohn meinte immer
der Wagen bräche, und es waren
doch nur die Bande, die vom
Herzen des treuen Heinrich
absprangen, weil sein Herr erlöst
und glücklich war.

The following scheme was used:

```
\defraggedrightscheme flatterXXX:
 zone 1st,2nd:(2em,3em)
 flutter indent,short:(0pt,0.5em)
 last nw,sh:(6.5em,2em)
 space word,punct:(0.33333em,0pt)
 hyphenmins left,right:(4,4)
 hyphenpens ex,im:(50,500)
 hyphendems dbl,fin:(10000,20000)
 aid exceed,adj:(2pt,0)\end
```

Figure 9b: Second part of "The Frog King", and the ragged-right scheme used to typeset the text.

(king's son) — are entered with a discretionary hyphen, for example, `K\"onigs\-tochter`. This is necessary to avoid overfull lines. I made hyphenation expensive and both "hyphenmins" are set to 4; see recommendation R9. But nevertheless, the result cannot be optimal as a second pass isn't able to utilize all possible hyphenation points.

For a German text I set all interword spaces to the same width, that is, the zero skip is used for `\xspaceskip`. The gutter is set to only 11 pt. I allow lines to overflow by 2 pt so that at least 9 pt separate the columns. This is more than two times the interword space and should be sufficient for a reader to identify the end of a line. As the column width is small I decided to make the last line in the worst case only 1 em shorter than the left side of the zone in a short line. Short lines are 5 pt shorter than long lines. Thus the zone width measures ≈ 35 pt

The communications of the TEX Users Group are published irregularly at Providence, Rhode Island, and are distributed as a benefit of membership both to individual and institutional members.

Submissions to *TUGBOAT* are for the most part reproduced with minimal editing, and any questions regarding content or accuracy should be directed to the authors, with an information copy to the Editor.

The deadline for submitting items for Vol. 7, No. 2, is April 28, 1986; the issue will be mailed in late June.

Manuscripts should be submitted to a member of the TUGBOAT Editorial Committee. Articles of general interest, those not covered by any of the editorial departments listed, and all items submitted on magnetic tape or as camera-ready copy should be addressed to the Editor, Barbara Beeton.

Contributions in camera copy form are encouraged, as is electronic submission of items on magnetic tape, via electronic mail, or transferred directly to the AMS computer; for instructions, write or call Barbara Beeton.

I carefully read the November issue of *TUGBOAT* and was shocked to see that I did not acknowledge the contributions of Pierre MacKay, Michael Spivak, and the founders of our organization. I correct this by thanking them and the AMS for creation of a good productive organization. Organizations like TUG always depend upon some good people contributing of their time. These people have really set a solid foundation for us to build upon.

Michael Spivak has resigned from the Steering Committee since he is now strictly in the TEX community for financial gain. I am sure that he will continue to be available as a sounding board.

Figure 10: *TUGboat* **7**:1 (1986) a) & b) five paragraphs from page 2; c) two paragraphs from page 7.

(only two lines are tight) or a rather large ≈ 23% of the `\hsize` to compensate for missed hyphens. See the end of Fig. 9 for the complete definition.

I made three additional changes. The text after the quotation in the first column of Fig. 9b was forced to start with a short line; same for the following paragraph. Moreover I added a tie to remove a short word that sticks out in the middle column of Fig. 9a: `versprichst, so~will`.

Of course, the result is not perfect; the ideal described by the above-discussed recommendations is not reached. But the output is much better than the result with TEX's `\raggedright` and `\noindent`. That creates four overfull lines, the start of two paragraphs are not identifiable as the last lines of the previous paragraphs are nearly filled, and many more lines end with a hyphen.

Back to English: *TUGboat* Volume 7:1, issue no. 14, published in March 1986, is special; see [1]. First, it has two guest editors, David Kellermann and Barry Smith, and second, a unique layout designed by Martha Gannett is used. On pages with three columns the column width seems to be 150 pt, the main font is `cmr9`, and all paragraphs are set ragged right. I assume the editors adjusted some line breaks manually. This offers an opportunity to see how the macros of this article handle this situation.

I took seven paragraphs from the issue: The first five are from page 2, the inside cover. These are the first five paragraphs in this issue. Two are copied from page 7, the first two paragraphs of the regular column "From the President". I don't know which macros and parameter settings were used by the editors. I guessed some values from the output and added others freely to try to achieve a good result, according to the recommendations we have seen.

```
\defraggedrightscheme issueXIV: % used with cmr9
   zone 1st,2nd:(20pt,26pt) % guess
   flutter indent,short:(20pt,5pt)
   space word,punct:(3.3333pt,4.4444pt) % guess
   hyphenmins left,right:(4,4)
   hyphenpens ex,im:(50,500)
   hyphendems dbl,fin:(10000,20000)
   aid exceed,adj:(2pt,0)\end
```

Figure 10a) shows the first two paragraphs. The first has the same line breaks as the one published; the second is one line longer because line 4 has in the issue a width of more than 150.9 pt, i.e., it is overfull and only the high `\hfuzz` of 1 pt avoids the overfull bar. In Fig. 10a) this line is short and can only have 147 pt.

The three paragraphs in Fig. 10b) have identical right margins to the ones in the issue; the last one is started with a short line though. I typed `submit\-ted` as this hyphen isn't found with the $(4, 4)$ setting for the "hyphenmins". I accepted this as the hyphen occurs in the original too.

The first paragraph in Fig. 10c) is different from the original. It has the same length in lines but avoids a word that sticks out. A tie in this paragraph between "build upon" lengthens the last line; otherwise "set" in the penultimate line appears at the end of the third last line. The second paragraph would add the word "be" in the penultimate line compared to the output in issue 14. This was changed by using a tie between "be available".

In olden times when wishing
still helped one, there lived a
king whose daughters were all
beautiful, but the youngest was
so beautiful that the sun itself,
which has seen so much, was
astonished whenever it shone in
her face. Close by the king's
castle lay a great dark forest,
and under an old lime-tree in
the forest was a well, and when
the day was very warm, the
king's child went out into the
forest and sat down by the side
of the cool fountain, and when
she was bored she took a golden
ball, and threw it up on high and
caught it, and this ball was her
favorite plaything.

In olden times when wishing still
helped one, there lived a king
whose daughters were all beau-
tiful, but the youngest was so
beautiful that the sun itself, which
has seen so much, was aston-
ished whenever it shone in her
face. Close by the king's castle
lay a great dark forest, and under
an old lime-tree in the forest
was a well, and when the day was
very warm, the king's child went
out into the forest and sat down
by the side of the cool foun-
tain, and when she was bored she
took a golden ball, and threw
it up on high and caught it, and
this ball was her favorite play-
thing.

In olden times when wishing
still helped one, there lived a
king whose daughters were all
beautiful, but the youngest was
so beautiful that the sun itself,
which has seen so much, was
astonished whenever it shone in
her face. Close by the king's
castle lay a great dark forest, and
under an old lime-tree in the
forest was a well, and when the
day was very warm, the king's
child went out into the forest
and sat down by the side of the
cool fountain, and when she was
bored she took a golden ball, and
threw it up on high and caught it,
and this ball was her favorite
plaything.

Figure 11: Using a) `\secVII`; b) `\secVIIn`; c) `\secVIIn` and high penalties.

Let's look again at our single paragraph. Using the `\hsize` of Figs. 1–5 the scheme `\secVII` creates the output of Fig. 11a). The scheme `\secVIIn`—that is similar to the `\secVII` but without a change to the penalty or demerits parameters for hyphens and with some overshoot—gives a very different result, as shown in Fig. 11b): The lines are filled with more material but four hyphens appear now. If the two penalties are set in the scheme `\secVIIn` to the values of scheme `\secVII` a third set of line breaks is found; see Fig. 11c). Note the outputs in Fig. 11 differ from the 15 variants shown in Figs. 1–5.

```
\defraggedrightscheme secVIIn:
   zone 1st,2nd:(2em,0pt)
   space word,punct:(0.3333em,0.5em)
   flutter indent,short:(0pt,0.5em)
   last nw,sh:(6.5em,2em)
   hyphenmins left,right:(4,4)
   aid exceed,adj:(2pt,10000)\end
```

8 Manual adjustments

There are at least two insights that one can learn from Fig. 11. First, each parameter influences the output; this is of course a well-known fact. Second, it seems to be impossible to find a set of parameters that works for a couple of paragraphs as each paragraph seems to be so sensitive to the chosen parameters of the scheme. That is, each paragraph might require its own set of parameters or a lot of manual intervention to be typeset in the best way.

Some commands to change the parameters temporarily have been presented in R10 "exceptions". The implementation of R2 "flutter" adds macros like `\rrstartshort` and others. Moreover, the usefulness of ties was already shown. Although an author possesses a lot of techniques for fine tuning it is necessary to add control sequences to our toolbox to allow manually inserted line breaks too.

The simplest forms are to enter `\hfil\break` or `\break`. But this often works only if an existing break is used; otherwise lines might become over- or underfull or previous line breaks change. It is better to use a variant of the answer to exercise 14.15 of *The TEXbook* [8]. A command first ends the paragraph with a larger zone and with more overshoot. Then a new paragraph is started. This permits using a different parshape for the next lines. For example, the next line can be made long or short or the parshape could use long lines only.

```
\def\rrHbreakhere{%
  \parfillskip 0pt plus 1.5em minus 1.5em\par
  \parskip=0pt \rrnoindent}
\let\rrbreakls=\rrHbreakhere
\def\rrbreaksl{\rrHbreakhere\rrstartshort}
\def\rrbreakll{\rrHbreakhere\rrnoparshape}
\edef\rrMresetparskip{\parskip=\the\parskip}
```

The values of `\parfillskip` and `\parskip` are reset by `\par` so no group is needed. (See "Addendum" at the end of section 5.) The shrinkability of `\parfillskip` glue allows extending the line beyond the value of `\hsize` similar to the way `\rightskip` was changed in R3 "extended zone" of section 5. Of course, since these new control sequences are for fine control, the complexity to find the best action in problematic situations increases.

In olden times when wishing
still helped one, there lived a
king whose daughters were all
beautiful, but the youngest was
so beautiful that the sun itself,
which has seen so much, was
astonished whenever it shone
in her face. Close by the king's
castle lay a great dark forest, and
under an old lime-tree in the
forest was a well, and when the
day was very warm, the king's
child went out into the forest
and sat down by the side of
the cool fountain, and when she
was bored she took a golden
ball, and threw it up on high and
caught it, and this ball was her
favorite plaything.

In olden times when wishing still
helped one, there lived a king
whose daughters were all beau-
tiful, but the youngest was so
beautiful that the sun itself, which
has seen so much, was astonished
whenever it shone in her face.
Close by the king's castle lay a
great dark forest, and under an
old lime-tree in the forest was
a well, and when the day was very
warm, the king's child went out
into the forest and sat down by
the side of the cool fountain, and
when she was bored she took
a golden ball, and threw it up on
high and caught it, and this ball
was her favorite plaything.

The deadline for submitting items
for Vol. 7, No. 2, is April 28, 1986;
the issue will be mailed in late June.
Manuscripts should be submit-
ted to a member of the TUGBOAT
Editorial Committee. Articles of
general interest, those not covered
by any of the editorial departments
listed, and all items submitted
on magnetic tape or as camera-ready
copy should be addressed to the
Editor, Barbara Beeton.
Contributions in camera copy
form are encouraged, as is elec-
tronic submission of items on
magnetic tape, via electronic mail,
or transferred directly to the AMS
computer; for instructions, write or
call Barbara Beeton.

Figure 12: a) one fix in Fig. 11a); b) four fixes in Fig. 11c); c) Fig. 10b) and two fixes.

Why 1.5 em in `\rrHbreakhere`? Well, it is clear that the stretch- and shrinkability should depend on the font, thus the unit "em". What is needed is to adjust the line breaks by moving a few characters from one line to another. The specification `\parfillskip 0pt plus 1fil minus \maxdimen` or something similar acts much more strongly but it might change earlier line breaks; for example, see Fig. 8b. This should not happen.

Applications. Does the sample paragraph require manual adjustment? If the recommendations of [3] are followed then the text should have no hyphens. The text was typeset without hyphens only four times in Figs. 1–5 and 11: 5a), 5c), 11a), and 11c). The simple case 5a) already fulfills all the explicitly stated criteria. Nevertheless, the longest line width in this output measures only ≈ 145.6 pt and not many lines reach a similar length. Fig. 11a) has the problem that a two-letter word sticks out. It can be fixed by applying `\rrbreaksl` after "shone"; the result is shown in Fig. 12a). Its longest line measures ≈ 147.4 pt. But most lines are much shorter.

And Fig. 5a) is far from obeying the recommendations for Flattersatz. The lines do not alternate well in their lengths and especially the first four lines build a "staircase down" violating R4 "no shape". This problem occurs in Fig. 12a) too, although it does not have the second staircase in lines 13–16 anymore.

Applying the latest used scheme of Fig. 11c) I entered four line breaks: `\rrbreaksl` after "still" and `\rrbreakls` after "was", "took", and "on". And in the last line I suppressed the overfull line warning with `\rrbreakll`. Thus, I tolerate a hyphen in the second line as it seems to be impossible to avoid all hyphens in the text and I accept that the start of the next paragraph might be difficult to identify.

It's simple to move the word "should" one line up in Fig. 10a) to make this paragraph identical to the one in the issue. A fix for the B-like contour that two paragraphs built in Fig. 10b) is more complex. With `\rrbreakls` in each paragraph (after "submitted" and "copy") the result of Fig. 12c) is accomplished. It has one more hyphenated line though.

9 Summary

To typeset a text with a straight-left and a ragged-right margin, typographers recommend certain settings for the interword spaces, the indentation and the length of the last line of a paragraph. Moreover, topics like hyphenation and the length of line pairs play a major rôle. The occurrence of a structure at the right margin adds a new aesthetic element but it should not attract the attention of the reader. Thus, it increases the complexity of finding a pleasing set of line breaks. And it raises the question: When does the ragged-right margin look acceptable?

There is more than one idea of how ragged-right output should look. German typographers use different words to distinguish two concepts. One works with small measures and accepts a lot of hyphenated words; this is called "Rauhsatz". The other requests less hyphenation and can be used with larger `\hsize`s, even as a replacement for justified typesetting, the "Flattersatz". English tradition for good ragged-right typesetting agrees with some aspects of the Flattersatz but it seems to deviate not only for hyphenation but also in how to rate a ragged-right margin as good.

This article extracts ten recommendations for ragged-right typesetting from the statements of a well-known German typographer. The plain TEX macro `\raggedright` implements only two of them. The article tries to identify for the others which parameter settings and macros support plain TEX's `\raggedright` to typeset text in Flattersatz.

1. Two assignments, one to the stretchability of `\rightskip` and one to the `\tolerance`, implement a zone once for the first and once for the second and third pass.
2. The creation of a `\parshape` command from pairs of lines with different line lengths seems to avoid most unwanted shapes of the right margin. But even a neutral `\parshape` helps to avoid words that stick out and is quite useful.
3. As no straight right margin is visible an assignment to the shrinkability of `\rightskip` lets lines extend beyond the `\hsize`.
4. If the start of a paragraph is mainly signaled by the length of the last line of the previous paragraph then the natural width and the shrinkability of `\parfillskip` should make last lines 2 to 4 em shorter than any other line.
5. With narrow line lengths indentation should be avoided, i.e., `\parindent` set to 0 pt if the indentation isn't specified in the `\parshape`.
6. Either the natural width of `\spaceskip` is used to get fixed-width interword spaces or the three `\fontdimen` 3, 4, 7 of all fonts are set to 0 pt.
7. To control hyphenation, six parameters shall be considered: a) two penalties: `\hyphenpenalty` and `\exhyphenpenalty`, b) the length of word parts: `\lefthyphenmin` and `\righthyphenmin`, and c) two demerits: `\doublehyphendemerits` and `\finalhyphendemerits`. At least the values for the word parts should be larger than for justified typesetting; e.g., set both to 4.
8. In an emergency a very small amount of shrinkability can be used for `\spaceskip`: ≤ 0.015 em.
9. Use a `\parfillskip` that has small amounts of stretch- & shrinkability for manual line breaks.

An author must be aware that a lot of time-consuming manual adjustments to the text — applying among other things no. 9 — remain necessary.

References

[1] Barbara Beeton (ed.), *TUGboat* **7**:1 (1986); guest editors David Kellermann and Barry Smith; layout design Martha Gannett. `tug.org/TUGboat/tb07-1/tb14complete.pdf`

[2] Max Bollwage, *Typografie kompakt*, Berlin, Germany: Springer-Verlag, 2001.

[3] Robert Bringhurst, *The Elements of Typographic Style*, Seattle, Washington: Hartley & Marks Publishers, 4th edition, version 4.2, 2016.

[4] Friedrich Forssman & Ralf de Jong, *Detailtypographie*, 3rd edition, Mainz, Germany: Verlag Hermann Schmidt, 2004.

[5] Brüder Grimm, *Kinder- und Hausmärchen*, 14th edition, Darmstadt: Wissenschaftliche Buchgesellschaft, 1991; under license by Winkler Verlag, München, 1949.

[6] Yannis Haralambous: "New hyphenation techniques in Ω_2", *TUGboat* **27**:1 (2006), 98–103. `tug.org/TUGboat/tb27-1/tb86haralambous-hyph.pdf`

[7] 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 [10], 67–155.

[8] Donald E. Knuth, *The TEXbook*, Volume A of *Computers & Typesetting*, Boston, Massachusetts: Addison-Wesley, 1984.

[9] Donald E. Knuth, *The METAFONTbook*, Volume C of *Computers & Typesetting*, Boston, Massachusetts: Addison-Wesley, 1986.

[10] Donald E. Knuth, *Digital Typography*, Stanford, California: Center for the Study of Language and Information, CSLI Lecture Notes No. 78, 1999.

[11] Leslie Lamport, *LATEX: A Document Preparation System*, 2nd edition, Reading, Massachusetts: Addison-Wesley, 1994.

[12] Frank Mittelbach and Michael Goossens, *The LATEX Companion*, 2nd edition, Boston, Massachusetts: Addison-Wesley, 2004.

[13] Richard Southall, "First principles of typographic design for document production", *TUGboat* **5**:2 (1984), 79–90. Errata in *TUGboat* **6**:1 (1985), 6. `tug.org/TUGboat/tb05-2/tb10south.pdf` `tug.org/TUGboat/tb06-1/tb11gendel.pdf`

[14] Udo Wermuth, "Tracing paragraphs", *TUGboat* **37**:3 (2016), 358–373; Errata in *TUGboat* **38**:3 (2017), 414 (see [15]). `tug.org/TUGboat/tb37-3/tb117wermuth.pdf`

[15] Udo Wermuth, "A note on `\linepenalty`", *TUGboat* **38**:3 (2017), 400–414; Errata in *TUGboat* **39**:1 (2018), 87. `tug.org/TUGboat/tb38-3/tb120wermuth.pdf` `tug.org/TUGboat/tb39-1/tb121wermuth-adem.pdf`

[16] Hans Peter Willberg, Friedrich Forssman, *Lesetypographie*, 2nd printing, Mainz, Germany: Verlag Hermann Schmidt, 2005.

◇ Udo Wermuth
Dietzenbach, Germany
`u dot wermuth (at) icloud dot com`

Webnotes: Practical variations

David Walden

Over the past several years, I was involved with three published papers that had many more endnotes and references than could fit within the publishing journal's page quota. Instead, some of the less important notes and references were posted on the Web; we called them "webnotes". Thus we needed methods of generating a separate set of notes than those that were part of the published paper and ways of cross referencing between webnotes and the published papers. We (the authors and I) used a slightly different approach to the generating and cross-referencing in each case.

The three papers were all published in the *IEEE Annals of the History of Computing*. They were:

- The Font Wars, parts 1 and 2, by Charles Bigelow, *Annals* vol. 42, no. 1, 2020
- Interleaf, Inc.—1981 to 2000, by Mark Dionne and David Walden, *Annals* vol. 42, no. 1, 2020
- TEX: A branch in desktop publishing evolution, parts 1 and 2, by Barbara Beeton, Karl Berry, and David Walden, *Annals* vol. 40, no. 3, 2018, and vol. 41, no. 2, 2019

All three papers were part of the history-of-desktop-publishing project described at `history.computer.org/annals/dtp`. The Interleaf and TEX papers were composed in LATEX; the Font Wars paper was composed in Word.

1 Webnotes for the TEX paper

For the earliest paper, on TEX history, Karl Berry and I chose to model our webnotes approach on non-fiction books which avoid footnotes and even superscripted note numbers in the main text by starting each note or reference at the end of the book with a page number from the main text and a short quote from that page indicating the virtual position in the main text of the end-of-book note. Perhaps you have seen such a book. An example of our webnotes in this style is at `tug.org/pubs/annals-18-19/part-2-webnotes.pdf`.

We used the LATEX endnotes package for the notes and references that fit within the journal's page quota, and they were referenced from the main text with a single sequence of superscripted numbers (as in this paper).

For the webnotes, I modified the endnotes package and called it `Webnotes.sty`. My approach to the modification was trial and error. I only changed enough to get the capability we needed (I didn't do a complete rework of the package). Mainly I searched for instances of the text "endnote" and replaced "end" by "Web". I also added a few commands to the paper's main LATEX file to interface to the modified style file. My trial and error approach is useful because it allows me to change LATEX or its packages to accomplish what I need even though I don't have real LATEX expertise.

See Appendix A for more details.

2 Webnotes for the Font Wars paper

The Font Wars paper was composed in Word by its author. He and I also decided to have webnotes for the notes and references that would not be included in the published paper. He put the text "[Note *n*]" in his Word file where he wished he could have another note or reference, and he put all the notes to go on the Web in a separate Word file where each note had a heading like "**Note** *n*". (With a change of which part of IEEE produced the *Annals*, we hoped that putting webnote numbers in the published articles would be possible, and they accepted this approach.)

The author sent that file to me, and I put a Word bookmark at each instance of "**Note** *n*" throughout the file and added to the top of the file a list of note numbers with links to the bookmarks. You can see the result at `history.computer.org/annals/dtp/fw`. The HTML version of the file was created with "Save As `.html`" in Word, and the PDF file was done using Word's "Save As `.pdf`" option for PDF/A. Word's `.docx` to HTML conversion has some limitations, but it is good enough.

3 Webnotes for the Interleaf paper

The Interleaf paper used superscripted lowercase letters in the main text to refer to webnotes. An example of this is in line 51 at `tug.org/TUGboat/tb41-1/webnotes/interleaf-main-page.pdf`. Using the letters required a small modification to what had been done in modifying the endnotes package for the TEX history paper, which was then renamed `ILWebnotes.sty`. Again my approach to modifying the `.sty` file included trial and error; however, in the end I asked Karl Berry how to do something I seemed unable to figure out.

See Appendix B for more details.

4 Observations

I enjoyed having LATEX available to create the webnotes for the TEX and Interleaf history papers. I would not have tried to create the HTML version of webnotes for the Font Wars paper had this not been easy to do with Word.

Appendix A

You can see `Webnotes.sty` at `tug.org/TUGboat/tb41-1/webnotes/tex-webnotes.txt` and compare it with the unmodified `endnotes.sty`. The following went in the preamble of our LaTeX file and is explained below.

```
%for webnotes for published version
\newcounter{Mythepage}
%set starting page number
\addtocounter{Mythepage}{29}
\def\Nextpage{\stepcounter{Mythepage}}

%for webnotes published or not
\makeatletter \input{Webnotes.sty}\makeatother
\newcounter{Webnotecounter}
\def\Webnotecnt{\stepcounter{Webnotecounter}%
  \theWebnotecounter}
\def\ieeecsNonDisplayingText#1#2%
  {\textsuperscript{}%
  \Webnotetext[\theMythepage]% thepage or
                               % theMythepage
  {\quad\textbf{#1}\quad{}#2\vspace{4pt}}}
                           %[\Webnotecnt]}}
```

The following went at the end of the file:

```
\pagestyle{empty}%no page numbers on webnotes
\newpage
%webnotes don't have a section heading
\renewcommand{\notesname}{}
%the webnotes do have a header and
    %some explanation about the webnotes
\textbf{Additional references and notes for
  \TeX: A branch in desktop publishing
        evolution}
... explanation about end notes ...
{\frenchspacing\theWebnotes}
```

I will come back to the first five lines of the commands added to the preamble. The next three commands load the modified style file (I didn't bother to think about how to make `\usepackage` work). The next three lines define a webnote counter for use in superscripted webnote numbers; we didn't think the IEEE would allow us to use these in the published paper, but they are handy to have for composing the paper and its webnotes.

The rest of the lines for the preamble define a new command to use like `\endnote` in the main text. It has two arguments: a quote from the main text, and the webnote itself. (We were submitting our paper using an IEEE style which didn't allow any user defined commands; helpfully, however, the IEEE style maintainer was willing to add the command `\ieeecsNonDisplayingTex` to the IEEE style defined to throw away its two arguments. Thus we didn't have to embed our webnote commands in if-statements that were switched on and off depending if the output was for us or the IEEE.)

The (empty) argument of the `\textsuperscript{}` command in the first line of the definition produces a superscript in the main text — nothing in the published version. However, while composing and editing the paper, we could have the command `[\Webnotecnt]` be the argument of the `\textsuperscript` command, and `\Webnotecnt` would increment our webnote counter and put a superscript number in the main text (the square brackets are to distinguish the webnote numbers from the regular endnote superscripted numbers). When doing this for ourselves, we replaced the two final right curly braces of the definition with the text in the comment on the next line, passing the webnote number to the webnote pages.

Now let's look at the first five lines added to the preamble that I skipped over above. A new page number counter is defined and set to 29, which was the first page of part 2 of the paper in the published journal. With the published paper in hand, I found in our LaTeX file the locations of page breaks in the published paper and inserted a call there to `\Nextpage` which steps our page counter. (While we developed the paper, we instead used `\thepage`, so the page numbers were numbered from 1.) The command `\Webnotetext` on the third line of the definition of `\ieeecsNonDisplayingText` passes this page number to the webnotes pages via the `.Went` file. (A generic call to `\endnote` puts macro calls and the note text, etc., in a file named `\Jobname.ent`; and `\theendnotes` inputs that file, and its macro calls output the endnotes.)

Appendix B

For the Interleaf history paper, I added the following code to the preamble of the LaTeX file, where `\Noprint` is the command name instead of `\ieeecsNonDisplayingText` as it was in Appendix A. Some explanations are given in comments.

```
\usepackage{alphalph}%generate letter sequence
\makeatletter %same as in Appendix A
\input{ILWebnotes.sty}\makeatother
\newcounter{Webnotecounter}
%
%next definition steps the webnote counter;
 %\alpha will be executed as \Noprint is called
\def\Webnotecnt{\stepcounter{Webnotecounter}%
  \alphalph{\theWebnotecounter}}%
%
%similar to how things worked in Appendix A,
 %except the \Noprint definition puts a
 %superscript lowercase letter in the main text;
 %and \Webnotetext passes the quote, note, and
 %note number to the .Went file for later
 %processing into the webnotes.
\long\def\Noprint#1#2{%
  \textsuperscript{\Webnotecnt}%
  \Webnotetext[\theWebnotecounter]%
      {\quad\textbf{#1}\\\quad{}#2\vspace{4pt}}}
```

There was stuff at the end of the file as in the TeX history paper, ending with `{\frenchspacing\theWebnotes}`. Finally, the additional change to `Webnotes.sty` for `ILWebnotes.sty` (for letters instead of numbers): `\def\theWebnote{\@arabic\c@Webnote}` to `\def\theWebnote{\alphalph{\c@Webnote}}`.

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

Reading 29,000 COVID-19 papers

Jonathan Fine

On 17 March 2020, *The Register* wrote [5]:

> A dataset of more than 29,000 scientific papers focused on COVID-19, and the coronavirus family as a whole, has been publicly shared to ultimately help the medical world thwart the bio-nasties.
>
> Specifically, it is hoped AI-based tools can be developed to comb through this COVID-19 Open Research Dataset (CORD-19) and dig up vital clues and insights on how to treat and contain the virus.

This article considers how TEX, or more exactly LATEX, fits into this massive and important research effort. We read in [1] that each paper is represented as a single JSON object, which conforms to a schema [2].

And now we see the LATEX problem. Even though the source document for the PDF is an informally structured document, it does not conform to a standard. It does not have a schema. It is sure to be machine readable by only one software system, namely LATEX, along with the specified preamble.

Similarly, the resulting PDF is machine readable only by a PDF reader, and in general the only machine readable semantic information it contains are the links. In particular, screen readers often fail to work well with LATEX-produced PDF files.

The US National Library of Medicine provides a Journal Article Tag Suite (JATS) [4] which "is a continuation of the NLM Archiving and Interchange DTD work begun in 2002" by the (US) National Center for Biotechnology Information.

My preliminary web search shows little activity in the area of JATS and LATEX, even though JATS is an important schema for scholarly articles. The Scholastica journal management platform [6] provides a typesetting service (probably based on LATEX) that will "generate HTML, PDF, and full-text JATS XML versions of articles".

Scholastica "was founded in 2012 in response to a growing need in academia for an easier, more modern way to peer review research articles and publish high-quality open access journals online" by three people who met when they were graduate students at the University of Chicago [7].

Fields medallist Tim Gowers is a key Editor [3] of the journals *Discrete Analysis* and *Advances in Combinatorics*, which are arXiv overlay journals, published on the Scholastica platform.

We now return to the dataset of 29,000 scientific papers focused on COVID-19. Clearly, the primary value of these papers is the experience, training, skill and dedication of the authors of these papers. Any system that allows for discovery, analysis, extracting and other use of these papers adds further value.

This is the theme of the article in *The Register*. Can Artificial Intelligence help humanity make sense of and use these 29,000 articles? I hope the answer is Yes, because any contribution helps. We can also ask: Has LATEX similarly helped humanity?

For us as TEX and LATEX users (and developers) more important than a Yes or No is this question: What can we do now, and in the future, to make TEX more useful for scientific and human challenges such as COVID-19?

References

[1] https://pages.semanticscholar.org/coronavirus-research

[2] https://ai2-semanticscholar-cord-19.s3-us-west-2.amazonaws.com/2020-03-13/json_schema.txt

[3] https://gowers.wordpress.com/2019/10/30/advances-in-combinatorics-fully-launched/

[4] https://jats.nlm.nih.gov/

[5] https://www.theregister.co.uk/2020/03/17/ai_covid_19/

[6] https://scholasticahq.com/

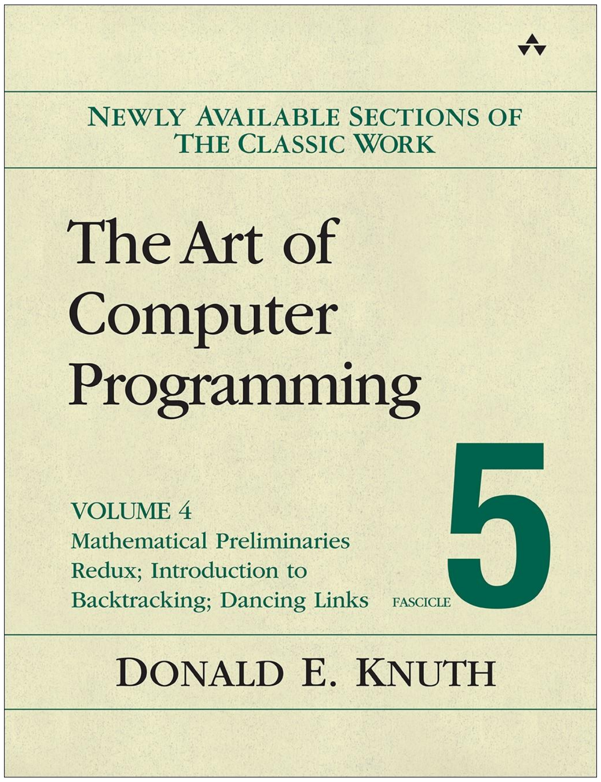
[7] https://en.wikipedia.org/wiki/Scholastica_(company)

◇ Jonathan Fine
jfine2358@gmail.com
https://jfine2358.github.io

About *The Art of Computer Programming, Volume 4, Fascicle 5*

David Walden

Donald E. Knuth, *The Art of Computer Programming, Volume 4, Fascicle 5*. Addison-Wesley, 2019, 382 pp., softcover, US$34.99, ISBN 978-0-13-467179-6. `tug.org/l/f5-aw`


NEWLY AVAILABLE SECTIONS OF THE CLASSIC WORK
The Art of Computer Programming
VOLUME 4
Mathematical Preliminaries Redux; Introduction to Backtracking; Dancing Links
FASCICLE
5
DONALD E. KNUTH

Fascicle 5 for Volume 4B of *The Art of Computer Programming* (*TAOCP*) was published shortly before Christmas 2019. It received some news coverage.[1,2,3]

I cannot presume to review Knuth's new fascicle in the sense of judging the mathematical or algorithm value of its contents. Also, there is no point in judging the presentation — Knuth always works to his own standard of what's "useful and beautiful". (Speaking about writing *TAOCP* with TEX, Knuth says that what he does first has to appeal to him and, if he didn't like something, he would change it.[4]) Instead, I will report a bit *about* the fascicle.

1 Topics in Fascicle 5

As shown on the cover image of Fascicle 5, its major sections are Mathematical Preliminaries Redux, (Introduction to) Backtracking, and Dancing Links.

Mathematical Preliminaries Redux. The fascicle begins with an unnumbered section that adds

```
7.1 Zeros and ones
    7.1.1 Boolean basics
    7.1.2 Boolean evaluations
    7.1.3 Bitwise tricks and techniques
    7.1.4 Binary decision diagrams
7.2 Generating all possibilities
    7.2.1 Generating basic combinatorial patterns
      7.2.1.1 Generating all n-tuples
      7.2.1.2 Generating all permutations
      7.2.1.3 Generating all combinations
      7.2.1.4 Generating all partitions
      7.2.1.5 Generating all set partitions
      7.2.1.6 Generating all trees
      7.2.1.7 History and further references
```
[Fascicle 5 below; volume 4A above]
```
    7.2.2 Backtrack programming
```
[14 unnumbered but named subsections]
```
      7.2.2.1 Dancing links
```
[20 unnumbered but named subsections]

[Fascicle 5 above; Fascicle 6 below]
```
      7.2.2.2 Satisfiability
```
[17 unnumbered but named subsections]

Figure 1: Contents of Volume 4A, Fascicle 5, and Fascicle 6.[6,7]

more probability theory techniques to what was described in section 1.2, Mathematical Preliminaries, of Volume 1 of *TAOCP*. Knuth says that he has "run across" various such techniques in his years of preparing Volume 4 and would have included them in section 1.2 if he "had been clairvoyant enough to anticipate them in the 1960s." These additional mathematical preliminaries are presented with 11.5 pages of explanation and 15.5 pages of exercises.

Backtracking. This second topic in Fascicle 5 is the first 16 subsubsections of section 7.2.2 of Volume 4 as shown in Figure 1.[5]

The section has 26 pages of explanation about backtrack programming followed by 79 exercises. The explanation introduces and compares several (some historical) algorithms for backtracking, ways to improve on the algorithms, and ways to better program the algorithms (e.g., data structure tricks).

Dancing Links. This third topic is subsection 7.2.2.1 of Volume 4. Knuth's discussion of dancing links takes about 50 pages of explanation followed by three sets of exercises (450 exercises total). The explanation starts by noting that in backtrack programming there is a lot of doing and undoing, and doubly linked lists are helpful for this. But, when elements are dropped out of a list, there is often a better approach than leaving a deleted list item for a garbage collector and creating a new list item when one is added to the list. A better approach at various points in backtracking is to leave a deleted list item where it is in memory and to later reconnect it to the list as if it had never been deleted. This is "dancing links".

Descriptive style. In both the backtracking and dancing links portions of the fascicle, the algorithms and some implementation examples are presented as sort of a verbal flow chart, perhaps including some instructions from Knuth's MMIX computer and assembly language, for example:

B1. [Initialize.] Set $l \leftarrow 1$, and initialize the data structures needed later.
B2. [Enter level l] (Now $P_{l-1}(x_1,\ldots,x_{l-1})$ holds.) If $l > n$, visit $x_1x_2\ldots x_n$ and goto B5. Otherwise set $l \leftarrow$ min D_l, the smallest element of D_l.

This method of sketching algorithms has been used at least since the third edition of Volume 1; however, there seem to be fewer instances of sequences of assembly language instructions than I remember being in Volumes 1, 2, and 3.[8,9] As a one-time computer programmer, I wish that Volume 4A and the Volume 4B fascicles used something a little closer to a programming language for showing algorithms.

There is still plenty of discussion in the fascicle of low-level ways to implement algorithms efficiently. Here is some example text from the bottom of page 65: "Interesting details arise when we flesh out the algorithm and look at appropriate low-level mechanisms. There's a doubly linked 'horizontal' list of all the active options that involve it." The discussion continues on the next two pages including two 22x16 diagrams of the list in memory.

Throughout *TAOCP* Knuth refers to prior volumes by section number without a volume number.[5] The presentation style also presumes the reader has read the prior volumes. For example, the definition of "visit" in step B2 above of Algorithm B is given in Volume 1 (page 320); it means "do whatever activity is intended as the tree is being traversed".

Exact cover. Early on in the discussion of dancing links Knuth also introduces exact covering. He gives a simple example of exact covering on page 64. Suppose there are the subsets {c e}, {a d g}, {b c f}, {a d f}, {b g}, and {d e g} of the set S of letters {a b c d e f g}. The first, fourth, and fifth subsets provide an exact cover for S, in that taken together the three subsets contain all the items in S once and only once. This is perhaps clearer if set up as finding an exact cover within a 7x6 matrix of zeros and ones (see Figure 2).

With the exact cover concept introduced and possibilities for efficient implementation discussed, Knuth then gives Algorithm X (for exact cover via dancing links), the suggestion that the reader do an exercise, and then 30 more pages of variations, applications, and optimizations.[10]

$$\begin{array}{lcccccccl}
 & \mathrm{a} & \mathrm{b} & \mathrm{c} & \mathrm{d} & \mathrm{e} & \mathrm{f} & \mathrm{g} & \\
\text{row 1} & 0 & 0 & 1 & 0 & 1 & 0 & 0 & \{\mathrm{c\ e}\} \\
\text{row 2} & 1 & 0 & 0 & 1 & 0 & 0 & 1 & \{\mathrm{a\ d\ g}\} \\
\text{row 3} & 0 & 1 & 1 & 0 & 0 & 1 & 0 & \{\mathrm{b\ c\ f}\} \\
\text{row 4} & 1 & 0 & 0 & 1 & 0 & 1 & 0 & \{\mathrm{a\ d\ f}\} \\
\text{row 5} & 0 & 1 & 0 & 0 & 0 & 0 & 1 & \{\mathrm{b\ g}\} \\
\text{row 6} & 0 & 0 & 0 & 1 & 1 & 0 & 1 & \{\mathrm{d\ e\ g}\}
\end{array}$$

Figure 2: A combination of Knuth's formulas 5 and 6 on page 64 of Fascicle 5; rows 1, 4, and 5 form an exact cover of the set {a, ..., g}.

Puzzles. Many puzzles are about exact covers, such as the eight queens puzzle where the goal is to place eight queens on a chess board such that no two queens are in the same column, row, or diagonal. Backtrack programming, perhaps with the help of dancing links, can often be used for finding an exact cover.

(While describing backtracking, Knuth had already noted that one of the best ways to understand backtracking is to execute the basic backtracking algorithm, Algorithm B, by hand for the four queens puzzle — placing four queens on a 4 by 4 chessboard so no queen attacks any other queen. I spent a bunch of time doing this, and it helped me understand both backtracking and the potential subtleties of implementing it in code.)

As the dancing links discussion continues, Knuth develops various algorithms and presents some theorems, often using puzzles as illustrations, e.g., sudoku, polyominoes, and kenken. Knuth chats about this in the fascicle's preface. He sees puzzles as often being the best way to illustrate an algorithm. The odds are good, he says, that a page selected at random in the fascicle will mention a puzzle. He makes the point that the methods that he is describing are useful for creating puzzles as well as solving them. He also discusses the history of the puzzles and sees the fascicle as a contribution to the world of recreational mathematics as well as teaching computer methods.

Knuth has said that Volume 4 covers the kind of algorithms he enjoys most.[11] A quote from the Fascicle 5 preface: "I have had loads of fun writing the other fascicles, but without a doubt this one has been the funnest."

I do wish, in addition to all the puzzle examples, that the fascicle spent more time on real world applications. On the Internet,[12] I found the following statement, which helped me somewhat:

> By far the most relevant, large size, important application of set covering is in personnel shift planning (mainly in large airline companies). There, elements to be covered are the single shifts (or single flights), and sets are legal combinations of

work/no work schedules. These easily go to millions or even billions of variables, as the number of combinations is huge.

I guess I comprehend that the general topic of Volume 4, combinatorics, is relevant to a wide variety of real life problems.

Knuth lectures. If you haven't yet bought the book and want to know more about dancing links, Knuth's 2018 Christmas lecture is on the topic,[13] and there is a previous (2000) Knuth lecture also on dancing links.[14] Notice that these two lectures on the same topic are years apart; Knuth states in Volume 4A that he has been saving up various methods and examples for years to eventually select among them for Volume 4 of *TAOCP*. (He also emphasizes that there is much he doesn't cover; he has to "cut, cut cut", keeping only what he believes will remain of fundamental importance for decades.)

Knuth's 2019 Christmas lecture[15] is nominally about π; among other things, he gives a bunch of examples of π being used in his books as a source of random data. In the lecture Knuth also talks about Fascicle 5, gives examples (especially puzzle examples) from the fascicle, and promotes it ("it will be a good Christmas present"). Talk about Volume 4B starts at about minute 27 of the lecture's video, first with a bit about Fascicle 6 and then about Fascicle 5. Giving example after example of sudoku, Knuth says that he has studied sudoku so deeply, it is no wonder it took him a long time to write the book. He has said that this book is "tons of fun and teaches a few algorithms on the side".[16]

There is also a Knuth video on the subject of Fascicle 6, satisfiability and SAT solvers.[17] Figure 1 shows where Fascicle 6 fits within the topics of Volume 4 of *TAOCP*. Volume 4A discusses manipulation of 0s and 1s and methods of generating basic combinatorial patterns; Fascicle 5 discusses backtracking and how to do it more efficiently with dancing links; and then Fascicle 6 on satisfiability shows the use of those techniques to develop SAT solvers which can be applied to many, typically massive, real world problems. Knuth touches on some of the latter in the video. In the Preface to Fascicle 6, Knuth says, "The story of satisfiability is a tale of the triumph of software engineering blended with rich doses of beautiful mathematics." For any readers who have been wondering if Knuth's emphasis on efficient algorithms is still relevant with today's computers which are so much more powerful than in 1962 when Knuth started *TAOCP*, Fascicle 6 justifies the continuing thrust for maximum efficiency. Knuth reports that "modern SAT solvers are able to deal routinely with practical problems" involving "many thousands of variables" that were "regarded as hopeless just a few years ago". In Fascicle 6 he is describing a rapidly developing field — especially over the past few decades. Knuth has been actively learning and contributing to the field in various ways, but he says that he knows he must move on. Fascicle 6 is a 2016 snapshot of the field, leaning toward the implementation rather than theoretical side of things; and Knuth hopes that it contains a "significant fraction of concepts that will prove to be the most important as time passes".

2 Main text, exercises, and answers

Fascicle 5 is definitely an unusual book (although not so much for Knuth) in terms of the ratio of main text to exercises and answers. Fascicle 5 has 100 pages of main text, 88.5 pages of exercises (663 exercises total), and 176 pages of answers. Knuth says that he wrote 600 programs while writing this fascicle because he needs to program things to really understand them. A small set of the most important programs are available, written in CWEB, to help readers solve problems.

Digressing for a moment to Fascicle 6 (section 7.2.2.2, on satisfiability — see Figure 1), it has 132.5 pages of main text, 50.5 pages of exercises (526 exercises total), 106 pages of answers, and 310 pages altogether in the book. This gives us, so far in Volume 4B, 232.5 pages of main text, 139 pages of exercises (1,189 exercises), and 282 pages of answers.

In his Notes on the Exercises near the beginning of *TAOCP* Volume 4A, Knuth explains about the benefits of exercises, noting that the exercises allow (are designed for) "self-study as well as for classroom use." He continues, saying

> It is difficult, if not impossible, for anyone to learn a subject purely by reading about it, without applying the information to specific problems and thereby being encouraged to think about what has been read. Furthermore, we all learn best the things that we have discovered for ourselves. Therefore the exercises form a major part of this work; a definite attempt has been made to keep them as informative as possible and to select problems that are enjoyable as well as instructive.

Knuth has had this view a long time. I remember that in Knuth's interview in the book *Mathematical People*,[18] he said that when first in college he had doubts about his abilities. Therefore, he worked all the exercises in the textbook, not just the assigned exercises, and then found he really understood things. Also there was his problem solving course at Stanford: people we know who took this course said it was wonderful — the teacher and students jointly solved new problems with the teacher using his experience to guide the students in useful directions.[19]

3 What *TAOCP* is and isn't

I previously spoke to what *TAOCP* is and isn't: see my 2011 "appreciation" of Volume 4A in *TUGboat*.[20] I will repeat a couple of points here because there has been a lot of overstatement about *TAOCP* in the popular press which is all many lay people know about the *TAOCP*: (1) *TAOCP* is not a book for teaching computer programming to the typical person learning to program. It does teach (explicitly) analysis of algorithms and (less explicitly) problem solving in the sense of finding algorithms to solve problems. (2) *TAOCP* is not a definitive treatment of computer science (although it may have been closer to comprehensive at the project's start in 1962); it doesn't cover lots of computer science, for example, artificial intelligence, computer networks, and parallel processing. What it does cover, though, it covers unusually deeply; it also contains a significant amount of history of mathematical and computing algorithms. Don't misunderstand me — *TAOCP* was and remains a monumental achievement and superb contribution to computer science and mathematics, regardless of its present state with only two-thirds of Volume 4B published.

TUGboat's reviews editor has asked me about the ideal reader for *TAOCP*. I think this has changed from volume to volume and over time. When Volumes 1, 2, and 3 came out in relatively rapid succession from 1968 to 1973, the volumes and their contents (all organized in distinct volumes) were more or less unprecedented. A practicing computer programmer could turn to the volumes to find the best approach or implementation for a not-unusual problem, e.g., random number generation, hash coding, or sorting. Later, undoubtedly partially stimulated by Knuth's work, many other books and papers on these topics were published, and a programmer needing a method might turn to one of these instead of *TAOCP*.

By the time Knuth got to his planned (not too long) chapter on combinatorial algorithms, the field was expanding rapidly (he "was confronted with ... a prodigious explosion of new ideas!"). Now, extrapolating from the length of fascicles 5 and 6, we can expect a total count in Volumes 4A and 4B of over 1,000 pages, and Knuth's outline continues on to Volumes 4C and 4D. The reader of the primary topics of Volume 4 is probably now a math or algorithms specialist (or someone studying to be one), or someone studying a particular type of puzzle who can find his or her way through the math, or someone developing a solution to a *big* real world problem. General computing practitioners may be more likely to skip to the bits of history Knuth includes; these remain fascinating.

Also, these days practitioners needing techniques covered in Volume 4 may well be able to find needed algorithms on the Web, perhaps even coded in the programming language the programmer is using for his or her larger project, perhaps even provided by an explicit library on the topic (in some cases the code found will be an implementation of an algorithm from Volume 4). On the other hand, having spent as many hours with Fascicle 5 as I have writing this description of the book, I am tempted to spend more time with the book in order to really understand the backtrack and exact cover algorithms, even lacking a problem to solve that needs the methods.

4 In conclusion . . .

Fascicle 5 (and Fascicle 6) is another spectacularly impressive production by Knuth. It is incredible that one man can collect the topics and prior publications, understand both the problems and the solutions, sometimes extend them, write hundreds of programs and develop hundreds of exercises and answers, typeset the book himself, and do all this carefully enough that he can offer rewards for mistakes that are found. And, while doing all this, he also finds time for giving the occasional lecture, writing a major work for organ[21], and who knows what other projects he has underway. I eagerly await the next publication from Donald Knuth.

Notes

[1] `slashdot.org/story/364386`

[2] `tug.org/l/f5-xmas-pi`

[3] Readers of this journal likely will know, at least roughly, the story of Don Knuth's decades long work on his magnum opus, *The Art of Computer Programming*: he started it in 1962, published three volumes in 1968, 1969, and 1973, suspended work in 1977 to develop TeX, and returned to work on Volume 4 in 2001. For anyone who wants a more detailed history, there are descriptions on the Web about the original intention for the book(s), how the project was originally received, and how the project has evolved, for example: `tug.org/l/taocp-amsreview`, `tug.org/l/taocp-wiki`, `tug.org/l/taocp-softpano`.

Knuth has noted that one of the reasons things have taken so long is that he keeps discovering new content that needs to be included.

[4] `youtube.com/watch?v=2BdBfsXbST8`, minutes 1:36:00 to 1:38:00.

[5] Section 2.2.2 begins, "Now that we know how to generate simple combinatorial patterns ... we're ready to tackle more exotic patterns ..." Presumably we got this know-how from reading the 223 pages of narrative and

exercises and 149 pages of answers in section 7.2.1 in Volume 4A.

[6] In this and the other books of *TAOCP*, some of the numbered and unnumbered section titles are prefixed with an asterisk. These are sections that Knuth says can be skipped upon first reading and come back to later. Udo Wermuth pointed me to this explanation in Volume 1. Udo, who is well known to readers of *TUGboat*, is one of the few people Knuth acknowledges by name in Volume 4A and Fascicles 5 and 6.

[7] Donald E. Knuth, *The Art of Computer Programming, Volume 4, Fascicle 6*. Addison-Wesley, 2016, 310 pp., softcover, US$29.99, ISBN 978-0-13-439760.
tug.org/l/f5-aw

[8] In the days of MIX, before Knuth developed his MMIX RISC computer architecture:
www-cs-faculty.stanford.edu/~knuth/mmixware.html

[9] Martin Ruckert has written a book which reimplements the MIX code examples in Volumes 1, 2, and 3 as MMIX code examples: Martin Ruckert, *The MMIX Supplement: Supplement to The Art of Computer Programming Volumes 1, 2, 3 by Donald E. Knuth*, Addison-Wesley, 2015.

Ruckert has also published several papers in *TUGboat*, including some that stem from his work developing the MMIX supplement, e.g., Computer Modern Roman fonts for ebooks, *TUGboat* 37:3 (2016), pp. 277–280, tug.org/TUGboat/tb37-3/tb117ruckert.pdf.

[10] A worked example of Algorithm X is in Wikipedia: en.wikipedia.org/wiki/Knuth%27s_Algorithm_X

[11] At youtube.com/watch?v=2BdBfsXbST8, about minute 32:10; keep watching for a few more minutes to hear Knuth describe the original purpose of *TAOCP*. See lexfridman.com/donald-knuth/ for a table of contents for the video. There is a lot of interesting stuff in this interview.

[12] tug.org/l/f5-cover-use

[13] tug.org/l/knuth-xmas18

[14] tug.org/l/knuth-xmas00

[15] tug.org/l/knuth-xmas19

[16] A list of Knuth lectures, including Christmas lectures, is at: tug.org/l/f5-xmas-list.

[17] youtube.com/watch?v=g4lhrVPDUG0

[18] Donald J. Albers and Gerald L. Alexanderson, *Mathematical People: Profiles and Interviews*, Mathematical Association of America, Birkhäuser, 1985.

[19] For example, see:
www-cs-faculty.stanford.edu/~knuth/papers/cs1055.pdf
i.stanford.edu/pub/cstr/reports/cs/tr/89/1269/
i.stanford.edu/pub/cstr/reports/cs/tr/87/1154/

[20] David Walden, An appreciation: *The Art of Computer Programming, Volume 4A*, *TUGboat* 32:2 (2011), pp. 230–232.
tug.org/TUGboat/tb32-2/tb101reviews-knuth.pdf

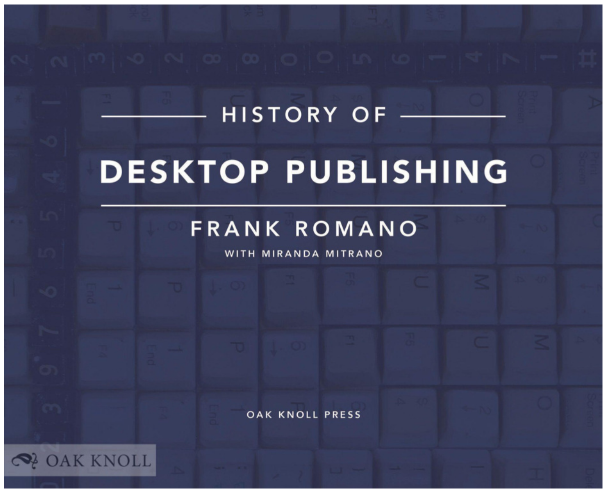
[21] youtube.com/watch?v=e_1a6bHGQGo

◇ David Walden
walden-family.com/texland

Book review: *History of Desktop Publishing*, by Frank Romano

David Walden

Frank Romano, *History of Desktop Publishing*. Oak Knoll Press, 2019, 400 pp. Hardcover, US$75. ISBN 9781584563808.
oakknoll.com/pages/books/133473



With the *History of Desktop Publishing*, Frank Romano has completed a trilogy of books covering the history of typesetting from hot metal (*History of the Linotype Company*), through phototypesetting (*History of the Phototypesetting Era*), and now digital desktop publishing. (The first of these was reviewed in *TUGboat*.[1])

The three publications are an astonishing effort — 480, 340, and 400 pages of a unique and fascinating story published in 2014 and 2019. It is hard to imagine another person better positioned to recount this history and share recollections than Frank Romano. He began his career in printing and publishing at Mergenthaler Linotype Company in 1959, worked at several other companies involved with typesetting and publishing, edited or published several trade-press publications, collected a massive library of books, journals, and newsletters and other artifacts about printing and publishing, co-founded the Museum of Printing,[2] wrote or co-authored dozens of history and tutorial books, and consulted and lectured widely about printing and publishing. Romano also has had a decades long affiliation with the Rochester Institute of Technology (RIT), where TUG plans to hold its 2020 annual conference. In particular, Romano was Melbert B. Cary, Jr. Distinguished Professor of Graphic Arts from 1992 to 1998, an RIT position Hermann Zapf and Charles Bigelow have also held.[3]

Figure 1: Table of contents of *History of Desktop Publishing*

Desktop publishing history. The desktop publishing (DTP) book is based on Romano's personal journey over nearly 60 years; Romano has called it a time capsule of what he has seen and materials he has collected. The book is nominally about a dozen years from the mid-1970s to the mid-1980s, but Romano provides lots of background material of what went before.

Because the book is based on Romano's experiences, the desktop publishing systems from different companies get different levels of treatment, as can be seen from the page numbers in the above table of contents (with page numbers). For instance, Interleaf is not covered as extensively as FrameMaker. TeX gets little more than two-and-a-third of the book's narrow short columns in a chapter covering typesetting in the pre-WYSIWYG era.[4]

It is not possible to summarize an author's thesis for this book. Rather, Romano is presenting "everything" he saw or has done, and not a succinct overall story. He says, "This is my story of the people, companies, technologies, and devices that defined desktop publishing for just over a decade. It is an amazing ride. Come along and relive it." (It occurs to me that one reading approach is to read a chapter a day as a way "living through" the era and in so doing develop my own mental picture of the era.) While most of the story is narrated in the third person, quite often Romano relates things more personally ("I ...") — a style I like.

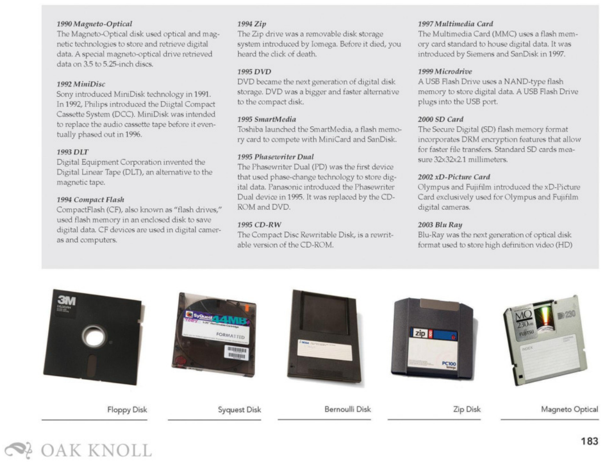
The book has hundreds of photographs and other images, numerous charts and lists, and many sidebars — all providing a reader with insight into desktop publishing history. To see examples of typical pages, go to the publisher's page for the book (`oakknoll.com/pages/books/133473`), click on the book cover image on the right of the page, and then navigate among the half dozen sample pages.

Romano's son Richard is credited with a significant contribution to the book — a nearly 100 page list of all of the product announcements in *TypeWorld* from 1983 through 1995. I see these pages as a marvelous resource for anyone wanting to follow the chronology of development of desktop publishing. The archive of *TypeWorld* issues from which this list was drawn is in the Romano Library at the Museum of Printing. It would be an even more valuable resource if a computer searchable copy was available. This is an instance where I wish a data CD came with a book; paying full price for a hardbound copy of the book and a searchable *TypeWorld* list would be easy for me to justify. (By the way, Romano is very welcoming of researchers who visit his library at the Museum of Printing.)

This book lists a secondary author: "With Miranda Mitrano"; she did the "book design" which I take to mean final composing in InDesign (Romano did the writing and draft composing in QuarkXPress). The dedication is to Jonathan Seybold for his bringing the major desktop publishing players together.[5] Barbara Beeton gets a special thanks after the dedication, and Kim Pickard is thanked for photographing items from Romano's collection.

In his review of the prior Linotype book,[1] Boris Veytsman complained slightly about the format of that book — 10 1/2″ by 8 1/2″ landscape view (as shown in the cover image at the beginning of this review) and with three narrow columns of ragged right text. (The phototypesetting and Linotype books have the same shape and column characteristics.) Romano uses landscape mode because of the number of images he wants to show across one page (for example, top left of next page).[6] Once in landscape mode, it makes sense to have three narrower columns than two wider columns, and with narrower columns ragged right makes sense.

As Boris Veytsman said in his review of the Linotype book,[1] Frank Romano's *History of Desktop Publishing* is an essential book for anyone studying the field and for many just generally interested in the history of printing and typesetting.[7] You will



1990 Magneto-Optical
The Magneto-Optical disk used optical and magnetic technologies to store and retrieve digital data. A special magneto-optical drive retrieved data on 3.5 to 5.25-inch discs.

1992 MiniDisc
Sony introduced MiniDisk technology in 1991. In 1992, Philips introduced the Diigtal Compact Cassette System (DCC). MiniDisk was intended to replace the audio cassette tape before it eventually phased out in 1996.

1993 DLT
Digital Equipment Corporation invented the Digital Linear Tape (DLT), an alternative to the magnetic tape.

1994 Compact Flash
CompactFlash (CF), also known as "flash drives," used flash memory in an enclosed disk to save digital data. CF devices are used in digital cameras and computers.

1994 Zip
The Zip drive was a removable disk storage system introduced by Iomega. Before it died, you heard the click of death.

1995 DVD
DVD became the next generation of digital disk storage. DVD was a bigger and faster alternative to the compact disk.

1995 SmartMedia
Toshiba launched the SmartMedia, a flash memory card to compete with MiniCard and SanDisk.

1995 Phasewriter Dual
The Phasewriter Dual (PD) was the first device that used phase-change technology to store digital data. Panasonic introduced the Phasewriter Dual device in 1995. It was replaced by the CD-ROM and DVD.

1995 CD-RW
The Compact Disc Rewritable Disk, is a rewritable version of the CD-ROM.

1997 Multimedia Card
The Multimedia Card (MMC) uses a flash memory card standard to house digital data. It was introduced by Siemens and SanDisk in 1997.

1999 Microdrive
A USB Flash Drive uses a NAND-type flash memory to store digital data. A USB Flash Drive plugs into the USB port.

2000 SD Card
The Secure Digital (SD) flash memory format incorporates DRM encryption features that allow for faster file transfers. Standard SD cards measure 32x32x2.1 millimeters.

2002 xD-Picture Card
Olympus and Fujifilm introduced the xD-Picture Card exclusively used for Olympus and Fujifilm digital cameras.

2003 Blu Ray
Blu-Ray was the next generation of optical disk format used to store high definition video (HD)

find information in this book on and around desktop publishing that you are unlikely to find anywhere else — certainly not all collected in one place.

Phototypesetting history book. Romano's previous book, *History of the Phototypesetting Era*,[8] also deserves a few words. Romano calls the book a "time capsule for a bygone era" and, as with the desktop publishing book, states that the narrative is primarily based on his personal recollections.

Romano notes that this book is based on his 54 years of participation in the printing and publishing industry but the actual book was a collaborative project. In the fall quarter of 2013, Romano served (for the third time) as a "Research Professor from Industry" in Cal Poly's Graphic Communication Institute which has a "learn by doing" teaching philosophy.[9] The book design, prepress, and printing work was done by students with the book being printed in the university's graphic communication laboratories (160 pages on one type of press and 188 pages on another press, presumably to give the students broader experience). Thirteen students are listed as co-authors of the book. Frank's son Richard Romano is listed on the title page for "editorial services", son Robert Romano is listed for "schematics", and Museum of Printing co-founder Kim Pickard is listed for "original photography".

This book is dedicated to John W. Seybold (father of Jonathan, the dedicatee of the desktop publishing book), "a true pioneer in automated typesetting", and there is a half page sketch of Seybold's contributions as the industry developed.[10]

Romano says the book covers the period roughly from 1945 when phototypesetting first became available until 1985 when the last phototypesetter was built. However, six of his first seven chapters cover typesetting and printing history before the first phototypesetter, and there is plenty of background description in the rest of his 28 chapters. The book is not just about the evolution of phototypesetting technology; it also covers development of the industry, specific companies, and specific machines. The book contains viii pages of frontmatter, 308 main pages divided into 28 chapters, a 16-page index, a 14-page chart of second and third generation phototypesetters from 1952 to 1985, a one-page-26-item bibliography, and a page with a colophon and afterword. Within the book are numerous other charts and diagrams and hundreds of photographs and other images. There is so much detailed content that some readers may not easily get the overall picture of the era. John Seybold's book *The World of Digital Typesetting*[11] has a bit more of a tutorial feel to it than Romano's book does; and, ending as it does with 1984, covers the same period as Romano's book. It might make sense to read the two books in parallel.

In any case, as with my view of the desktop publishing book and Boris Veytsman's review of the Linotype book, I see Romano's phototypesetting book as being an essential resource to anyone studying the phototypesetting era or the history of printing and typesetting more generally.[12]

Given the value of Romano's phototypesetting and desktop publishing books, I must now order my own copy of the Linotype book which I had previously only looked at in the gift shop and bookstore at the Museum of Printing.

At present, the Linotype book is available from its publisher, Oak Knoll, or via people selling through Amazon. The phototypesetting book is available on Amazon from third-party sellers. The desktop publishing book is available from publisher Oak Knoll directly or via Amazon. I presume all three books are on sale in the gift shop of the Museum of Printing in Haverhill, Massachusetts.

Oak Knoll Press. Romano's desktop publishing history book was published by Oak Knoll Press. Learning a bit more about the press and the bookstore with which the press is associated may be interesting to people interested in fine book publishing, the used book business, and, in particular, books about books.

The url `oakknoll.com` takes one to the website of both Oak Knoll Press and Oak Knoll Books. The former is "publishers and distributors of fine books about books since 1978". The latter is a bookseller of rare and out-of-print books, particularly books about books. Three sources of more information about Oak Knoll are a book by founder Robert D. Fleck,[13] a similar history on the Web,[14] and an interview of him.[15] Since Fleck's death in 2016, one of his sons runs the company; the Wikipedia entry for the company is also informative.[16]

The press does a good bit of co-publishing — jointly publishing with a company or institution that develops a new book. In this way Oak Knoll expands its list of publications while avoiding increasing book development costs; the financial results are split under an arrangement that benefits both entities, and visibility of the book and its distribution is expanded. (It seems that Romano's desktop publishing history was at least somewhat of a co-published book as, in addition to Mitrano preparing a designed book in InDesign, Romano says he and Mitrano did the prepress for the book.)

In the fall 2019 Oak Knoll Press catalog I find books distributed for:

- Grolier Club
- Cotsen Children's Library, Princeton University
- Edizioni Valdonega, Verona, Italy
- American Antiquarian Society
- Bibliographical Society of the University of Virginia
- Penn Libraries/Kislak Center
- Sheridan Libraries, Johns Hopkins University
- Smithsonian Institution
- Thomas Fisher Library, University of Toronto
- AdVenture SA, Athens, Greece
- Center for Book Arts
- CODEX Australia
- CODEX Foundation
- Verso
- Douglas Stewart Fine Books

The same catalog lists Oak Knoll Press books about bibliography, bookbinding, book collecting and bookselling, calligraphy and writing, fine press and artists' books, illustration and design, libraries, papermaking and paper decoration, printing history and publishing, type and typography, and more books about books.

Other Oak Knoll books recently reviewed for this journal include Nancy Stock-Allen's book about Carol Twombly[17] and Jerry Kelly's biography of Herman Zapf.[18]

Notes

[1] *History of the Linotype Company*, reviewed by Boris Veytsman, *TUGboat* 36:1, 2015, pp. 58–59, `tug.org/TUGboat/tb36-1/tb112reviews-romano.pdf`

[2] `museumofprinting.org/libraries`

[3] Romano also served as Roger K. Fawcett Distinguished Professor of Digital Publishing from 1998 to 2005, at which point he became a professor emeritus. In 2002 he had turned over the position of administrative chair of RIT's School of Printing Management to a new chair as part of the school's renaming to the School of Print Media, with the goal, according to Romano, of increasing "the benefits of an RIT education to the printing and publishing industries" (`tug.org/l/romano-chair`). He is now professor emeritus. He also received RIT's Cary Award (`tug.org/l/romano-cary-award`) and a special recognition award (`tug.org/l/romano-special-recog`).

[4] A complementary treatment of DTP history may be found via `history.computer.org/annals/dtp`.

[5] For more about Jonathan Seybold, see "Seybold Seminars" on page 225 of "Studying the histories of computerizing publishing and desktop publishing, 2017–19", David Walden, *TUGboat* 40:3, 2019. `tug.org/TUGboat/tb40-3/tb126walden-history.pdf`

[6] Frank Romano email of 2019-11-25.

[7] Historians would be aided by maximum accessibility into the book. However, it is impractical for the book's current index to comprehensively index such a massive amount of content. Ideally a way could be found for the entire book (not just the *TypeWorld* part mentioned above) to be searchable by people who own a copy — without sacrificing the publisher's and author's income. I suspect this capability can be developed relatively inexpensively. For instance, the book could have a url in it. At the url is a form into which a search term is typed, but not a PDF of the book. Executing the search returns a couple of lines from the book and a book page number for each found instance of the term.

[8] *History of the Phototypesetting Era*, Graphic Communication Institute at Cal Poly State University, San Luis Obispo, California, 2014. `rit.edu/press/history-phototypesetting-era`

[9] `cla.calpoly.edu/news/2015/frank-romano`.

[10] For more about John Seybold, see "Rocappi" on page 219 and "Seybold Reports" on page 224 of the paper cited in note 5. Together John and Jonathan influenced the course of publishing technology for three decades.

[11] `computerhistory.org/collections/catalog/102740425`

[12] As I said about the desktop publishing book (note 7), I hope at some point the phototypesetting book becomes available in searchable PDF form to better enable use of the resource.

[13] *Books about Books: A History and Bibliography of Oak Knoll Press, 1978–2008*, Oak Knoll Press, 2008. `oakknoll.com/pages/books/99582`

[14] Oak Knoll Books, `tug.org/l/fleck-web-history`

[15] Robert D. Fleck interview, Antiquarian Booksellers' Association of America. `youtube.com/watch?v=auKxVJpipig`

[16] `en.wikipedia.org/wiki/Oak_Knoll_Books_and_Press`

[17] `tug.org/TUGboat/tb40-3/tb126reviews-stock-allen.pdf`

[18] `tug.org/TUGboat/tb40-3/tb126reviews-kelly-zapfbio.pdf`

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

ArsTEXnica #27–28 (April and October 2019)

ArsTEXnica is the journal of GuIT, the Italian TEX user group (`www.guitex.org`).

ArsTEXnica #27 (April 2019)

Claudio Beccari, Editoriale [From the editor]; pp. 3–4

A short overview of the present issue.

Claudio Beccari, Language management and patterns for line breaking; pp. 5–28

Language management is supported by different files according to the language manager `babel` or `polyglossia`; they are similar to a certain extent, but differ in the way they handle the language patterns. There are also small differences when using XeLaTeX compared to LuaLaTeX. Obviously patterns are different from language to language, but there are also some languages with variants. Therefore the language-supporting compiler-structure has to manage a variety of situations.

Claudio Beccari, La bandiera europea e la sezione aurea [The European flag and the Golden Ratio]; pp. 29–33

The European flag contains a circle of twelve five-pointed stars distributed at the vertices of a regular dodecagon. This article shows how it is possible to draw the flag using only the `picture` environment, both to draw the stars and to put them in the correct position.

Roberto Giacomelli, Parsing di opzioni in LuaTEX [Option parsing in LuaTEX]; pp. 34–41

This paper explains how to implement a parser in Lua — the elegant and easy to use programming language included in LuaTEX — for a list of options in ⟨*key*⟩ = ⟨*value*⟩ format.

Such a parser is useful in package development, allowing an option system specifically designed for efficiency and syntax expressiveness.

Jean-Michel Hufflen, Antichi sistemi di notazione musicale [Early musical notations]; pp. 42–52

Some music engraving programs, such as MusiXTEX and LilyPond, support rendering of Gregorian chant's square notation on four-line staves. In this tutorial we explain how scores using this notation are organised. Then we show how it developed until the notations used in the early baroque era. [Originally published in proceedings of BachoTEX 2018. (*Tr. Tommaso Gordini.*)]

Joseph Wright, `siunitx`: passato, presente e futuro [`siunitx`: Past, present and future]; pp. 53–56

[Originally published in *TUGboat* vol. 39, no. 2 (2018), pp. 119–121. (*Tr. Tommaso Gordini.*)]

Frank Mittelbach, Il concetto di ritorno al passato per le classi e i pacchetti [A rollback concept for packages and classes]; pp. 57–63

[Originally published in *TUGboat* vol. 39, no. 2 (2018), pp. 107–112. (*Tr. Claudio Beccari.*)]

ArsTEXnica #28 (October 2019)

Claudio Beccari, Editoriale [From the editor]; pp. 5–7

A short overview of the present issue.

Gianluca Pignalberi, Massimiliano Dominici, Introduction to LaTeX and to some of its tools; pp. 8–46

Writing has a long history. Shorter is the history of typesetting and even shorter is the history of digital typography. Nevertheless, the latter has gained an unprecedented importance because of its capability to speed up the process of giving human beings well-composed information.

Our lessons, of which this is number zero, are focused on a digital typesetting system that has come to light in the late 1970s. It was intended to typeset scientific books; it is used to typeset nearly everything. It is TEX.

In this short course we will give an overview of how TEX and its most famous macro package LaTeX help engineers, scientists and professionals to compose their documents, whether they are books, papers, reports, presentations, posters, or any other material.

Enrico Gregorio, TEX, LaTeX and math; pp. 47–57

We discuss some aspects of mathematical typesetting: choice of symbols, code abstraction, fine details. Relationships between math typesetting and international standards are examined. A final section on typesetting of numbers and units reports on some recent developments in the field.

Guido Milanese, Bibliographies, LaTeX and friends; pp. 58–64

This article deals with the treatment of bibliographies within a LaTeX framework and workflow. A comparison of BibTEX bibliography format with other widely used formats shows that BibTEX has several advantages. The classical BibTEX programme is now obsolete, and `biblatex` + `biber` offer a highly customisable choice for bibliographies in any research area. GUI environments are also discussed, as well as possible future developments.

Agostino De Marco, Graphics for LaTeX users; pp. 65–101

This article presents the most important ways to produce technical illustrations, diagrams and plots, which are relevant to LaTeX users. Graphics is a huge subject per se, therefore this is by no means an exhaustive tutorial. And it should not be so since there are usually different ways to obtain an equally satisfying visual result for any given graphic design. The purpose is to stimulate readers' creativity and point them in the right direction. The article emphasizes the role of `tikz` for programmed graphics and of `inkscape` as a LaTeX-aware visual tool. A final part on scientific plots presents the package `pgfplots`.

Grazia Messineo, Salvatore Vassallo, Presentations with Beamer; pp. 102–109

In this article we briefly introduce the LaTeX class `beamer` for presentations. We give some tips to build an effective presentation and describe the main features of the class.

Claudio Beccari, The TOPtesi package. Typesetting a PhD thesis with LaTeX; p. 110

This article uses the information given in the previous five tutorials in order to describe how to use the TOPtesi LaTeX package to typeset a PhD thesis. This package has a specific option to configure the typesetting of such a thesis in the format agreed upon by ScuDo, the doctoral School of Politecnico di Torino.

Ulrike Fischer, Creating accessible PDFs with LaTeX; pp. 135–137

This article describes the current state and actions planned for the future to improve the accessibility of PDFs created with LaTeX, as it is currently undertaken by the LaTeX Team.

D. Ahmetovic, T. Armano, C. Bernareggi, A. Capietto, S. Coriasco, B. Doubrov, A. Kozlovskiy, N. Murru, `Axessibility` 2.0: creating tagged PDF documents with accessible formulae; pp. 138–145

PDF documents containing formulae generated by LaTeX are usually not accessible by assistive technologies for visually impaired people (i.e., by screen readers and Braille displays). The LaTeX package `axessibility.sty` that we developed manages this issue, allowing creation of PDF documents where the formulae are read by such assistive technologies, through the insertion of hidden comments. In this paper we describe the evolution of the package, that in the latest version also automatically generates the tagging of the formulae. The package however does not generate documents tagged according to the PDF/UA standard.

Gianluca Pignalberi, Uno script bash di ausilio alla redazione di manoscritti [A bash script to help edit manuscripts]; pp. 146–156

A manuscript editing session puts us in full view of a series of authors' repeated bad practices. Entering corrections entirely by hand can be a source of oversights. We will see how a bash script allows us to minimize them.

Jean-Michel Hufflen, A direct bibliography style for ArsTeXnica; pp. 157–159

We describe the `mlb-arstexnica` program, part of mlBibTeX's new version, and suitable for generating bibliographies for ArsTeXnica articles. First, we recall the notion of direct bibliography style related to mlBibTeX and mention the advantages of such a program. We show that our program provides additional services suitable for ArsTeXnica, compared to BibTeX's bibliography style `arstexnica.bst`.

Claudio Fiandrino, `smartdiagram`: The package and its journey; pp. 160–163

The `smartdiagram` package was born as a response to a question on TeX.stackexchange. The challenge was to emulate a feature that Microsoft PowerPoint provides: the capability of automating a diagram with animations. This feature allows the creation of diagrams from lists, so the user interface had to be as simple as possible, i.e., a list. In this article, I review the basic idea that overcomes the challenge and I expose the main features of the package along with a bit of its history.

Claudio Vincoletto, Metamorfosi dei tipi sublacensi [Metamorphoses of the Subiaco typefaces]; p. 164

The first and accurate digital revival of a classic typeface, based on medieval calligraphy and used for the first time in Italian incunabula. A copy of this original was employed by the last representative of the private press movement.

[Received from Massimiliano Dominici.]

Zpravodaj 2019/1–4

Zpravodaj is the journal of CSTUG, the TEX user group oriented mainly but not entirely to the Czech and Slovak languages (`cstug.cz`).

Petr Sojka, Úvodník staronového předsedy [Introductory Words from the once and future president]; pp. 1–10

The editorial discusses CSTUG's past three years, possible future directions of the group, membership issues and shaping the organization in the Internet era, together with changes related to the publishing of Zpravodaj CSTUG. The recent visit of the Grand Wizard is also reported, with insights.

Go forth and participate in CSTUG to make the bright future of TEX & Friends a reality! You can!

Dávid Lupták, Fantasia Apocalyptica: Česká premiéra [Fantasia Apocalyptica: The Czech première]; pp. 11–18

On Friday, October 11, 2019, the Czech première of the multimedia organ work Fantasia Apocalyptica by Donald Knuth took place in Brno on the occasion of the 25th anniversary of the foundation of the Faculty of Informatics of Masaryk University, with the author's participation. The article presents the event report and the Czech version of the brochure that we prepared for this concert.

Jano Kula, 14th ConTEXt Meeting 2020; pp. 19–23

From Sunday, September 6, 2020 to Saturday, September 12, 2020, the 14th ConTEXt Meeting, organized by the ConTEXt Group, will be held in Sibřina, Czech Republic.

Tomáš Hála, Tabulky v ConTEXtu: přístupy, možnosti, algoritmy [Tables in ConTEXt: Ways, possibilities, algorithms]; pp. 24–43

In the publication process, the typesetting of tables is one of the more complicated tasks. This paper reviews old and current ways of typesetting tables in ConTEXt (environments `table`, `tabulate`, `TABLE`, `xtables`), and compares them mutually and with the "rival" LATEX.

Tables can be generated from other formats such as the frequently used CSV. Therefore, the paper deals also with database processing.

Finally, some simple algorithms for easy extensions of the available repertoire are presented.

Lucie Schaynová, Jan Šustek, Aplikace parametrů řádkového zlomu a output rutiny k formátování sazby v TEXu [Parameters of the line breaking algorithm and the output routine and their applications for typesetting in TEX]; pp. 44–65

In the paper we go through inner parts of TEX to show how the particular characters of the `.tex` input file get to the `.pdf` output file. First we focus on the line breaking algorithm, explaining how its parameters affect the paragraph alignment. Then we focus on the output routine, showing how to put the typeset text on the page. Finally we mention the way to find the exact position of a particular point on the page with an application of MetaPost figures.

Jiří Rybička, Výsledky výuky zpracování textů [The results of teaching text processing]; pp. 66–72

Teaching of text processing has been offered at the Faculty of Business and Economics of Mendel University in Brno as an optional subject for more than 18 years. The concept of the subject has over time somewhat changed from the initial more technical concept to the current one focused on more general knowledge of typography and on technical texts, especially the final works. The article deals with the outline of the analysis of the learning outcomes in this subject, by processing selected exam results.

Petr Sojka, Ondřej Sojka, The unreasonable effectiveness of pattern generation; pp. 73–86

[Printed in *TUGboat* **40**:2.]

Jano Kula, ConTEXt marks; pp. 87–91

For approximately ten years now, we have seen the separation of the ConTEXt format into ConTEXt MkII and ConTEXt MkIV. In this article, I will explain the naming and the differences between ConTEXt formats.

Peter Wilson, It Might Work VIII — Drawings; pp. 92–104

[Printed in *TUGboat* **30**:1. Translated to Czech by Jan Šustek.]

[Received from Vít Novotný.]

Die TEXnische Komödie 4/2019–1/2020

Die TEXnische Komödie is the journal of DANTE e.V., the German-language TEX user group (`dante.de`). Non-technical items are omitted.

Die TEXnische Komödie 4/2019

Esther Bonhag, Sebastian Bonhag, Johannes Hielscher, Nils Pickert, Don Knuth's 80. Geburtstag – oder: Wie der Erlanger TEX-Stammtisch auf Polarexpedition ging [Don Knuth's 80th birthday — or: How the Erlangen TEX User Group went on a polar expedition]; pp. 11–19

It all started the day when Donald E. Knuth had his 117_8th birthday. Triggered by this event Heise Online had published an article on the hexadecimal dollar of Don Knuth. As usual this article made Sebastian continue reading in Wikipedia and Don's homepage, where a presentation held by Knuth, "A Possibly Swedish Pipe Organ Fantasy", on the next Friday at the KTH Royal Institute of Technology in Stockholm was announced.

Uwe Ziegenhagen, "Making TEX Great Again" – Die TEX Users Group-2019-Tagung in Palo Alto ["Making TEX Great Again" — The TEX Users Group 2019 Meeting in Palo Alto]; pp. 19–26

The TEX Users Group meeting for the 40th birthday of TEX gave me the opportunity to go to California. In this article I present impressions from this trip.

Falk Zscheile, DANTE e.V. auf den Kieler Open Source und Linuxtagen [DANTE at the Kiel Open Source and Linux Days]; pp. 27—31

When DANTE e.V. asked for someone to take care of their booth at the Open Source and Linux days in Kiel, a mere three weeks had passed since the DANTE folks had managed to help me significantly when I visited their booth in Sankt Augustin. This showed me how important personal relationships were even in our digital world, and convinced me to follow their call to Kiel.

Johannes Hielscher, Kirchheim „unter TEX": Die DANTE-Herbsttagung 2019 [Kirchheim "unter TEX": The DANTE Autumn Event 2019]; pp. 32–33

Besides the members meeting led by Martin Sievers, the focus of this event was the activities sponsored by DANTE. Doris Behrendt and Johannes Hielscher presented their impressions from the LATEX Village in the Chaos Communication Camp in Mildenberg, close to Berlin.

Mathias Magdowski, Personalisierte Aufgaben und passende Musterlösungen zu den Grundlagen der Elektrotechnik automatisiert mit LATEX, `pgfplots` und CircuiTi*k*Z erstellen [Personalized exercises and solutions for electronic sciences, automated with LATEX, `pgfplots` and CircuiTi*k*Z]; pp. 34–44

All students in our lecture on the basics of electronics receive their personal exercise via e-mail, can solve this manually and send their solution electronically for corrections. To reduce the effort for the lecturers, the students correct the exercises of their peers based on personalized solutions as well. The method is implemented in Matlab and, being highly automated, it also scales very well. Compared with multiple choice or number-unit exercises, the approach and calculations can be graded well. In this article we describe how the typesetting is generated in LATEX with `pgfplots` and CircuiTi*k*Z.

Herbert Voss, Erzeugung von animierten PDF-Dokumenten oder GIF-Dateien [Creating animated PDFs and GIF files]; pp. 45–52

The usage of TEX and company in schools is limited, unfortunately. But LATEX is very useful, with its packages covering nearly all possible aspects, in the creation of complex animated PDF files. In this article we show an application for use in senior math classes.

Ulrike Fischer, The package `luaotfload`; p. 53

The newest version of the `luaotfload` package allows one to "bolden" fonts artificially. This can, for example, be used to create bold math fonts for headlines, etc.

Marion Lammarsch, Elke Schubert, Reference sheet for a thesis with LATEX 2_ε and KOMA-Script; pp. 54–74

This LATEX Reference Sheet is for writing a thesis with one of the KOMA-Script document classes (`scrartcl`, `scrreprt`, `scrbook`) and all related packages that a thesis in the natural sciences may need. The source code and all parts are provided for creating your own version of a reference sheet, adapted to your personal needs.

Die TEXnische Komödie 1/2020

Doris Behrendt, Tagungsbericht GuIT meeting [Conference report of the GuIT Meeting]; pp. 8–13

The 2^{2^2}th meeting of the GuIT (Gruppo Utilizzatori Italiani di TEX, the Italian TEX user group), took place in Turin on October 26, 2019. I visited the meeting on behalf of DANTE to deliver birthday greetings.

Walter Entenmann, LuaTEX und `luamplib` [LuaTEX and `luamplib`]; pp. 14–30

This article describes the embedding of METAPOST code for images in a LATEX document with the package `luamplib` and the engine LuaTEX. This has the advantage that text and images are combined in one document and only one document has to be created, edited and maintained. The structure of this new innovative solution is presented and the commands are explained in detail and illustrated using examples.

Christine Römer, Diskussion um die Wiederverwendbarkeit von älteren Dokumenten [Discussion about reusability of old LATEX documents]; pp. 31–33

At the end of January there was an interesting discussion on the internal mailing list for members of DANTE e.V. about the reusability of old LATEX files.

Alan Munn, Ein zweiseitiger Leitfaden für LATEX-Pakete [A two-page guide to LATEX packages]; pp. 35–36

This is a list of the packages that are in my opinion the most useful for writing papers and theses in linguistics. I have made no attempt to justify the choices, but I find almost all of these packages essential in my own work. The essential packages and the basic linguistics packages (relevant to your particular field) should probably be loaded in every document you write. Unless noted, all packages are part of TEX Live.

Rolf Niepraschk, Die Schriftfamilie JetBrains Mono [The font family JetBrains Mono]; p. 37

The company JetBrains recently released the new Typewriter font JetBrains Mono under the free Apache 2.0 license. According to its statement, it is particularly good for displaying program source texts suitably.

[Received from Herbert Voß.]

The Treasure Chest

These are the new packages posted to CTAN (`ctan.org`) from October 2019–March 2020, along with a few notable updates. Descriptions are based on the announcements and edited for extreme brevity.

Entries are listed alphabetically within CTAN directories. More information about any package can be found at `ctan.org/pkg/`*pkgname*. A few entries which the editors subjectively believe to be of especially wide interest or otherwise notable are starred (*); of course, this is not intended to slight the other contributions.

Nearly all the packages in this column are included in the TEX Live 2020 release; the few exceptions are those where the package did not meet TEX Live's criteria for inclusion. For notable updates to the TEX engines and other software in TL'20 (and previous years), see `tug.org/texlive/doc/texlive-en/texlive-en.html#news`.

We hope this column helps people access the vast amount of material available through CTAN and the distributions. See also `ctan.org/topic`. Comments are welcome, as always.

◇ Karl Berry
`tugboat (at) tug dot org`

fonts

`clara` in `fonts`
Extensive family created by Séamas Ó Brógáin for *A Dictionary of Editing* (2015).

`cmupint` in `fonts`
Upright integral symbols for Computer Modern.

`domitian` in `fonts`
Extended Palladio (Palatino) with small caps, old-style figures, scientific inferiors, etc.

`erewhon-math` in `fonts`
Utopia-based OpenType math font.

`etbb` in `fonts`
Extended XETBook (which extended Tufte's ETBook).

`gfsdidotclassic` in `fonts/greek/gfs`
Classic GFSDidot font in OpenType.

`haranoaji` in `fonts`
Harano Aji fonts, Mincho and Gothic.

`haranoaji-extra` in `fonts`
Additional variants of Harano Aji fonts.

`lexend` in `fonts`
Lexend variable font.

`mpfonts` in `fonts`
  Computer Modern Type 3 fonts, a direct conversion from METAFONT via MetaPost.

*`newcomputermodern` in `fonts`
  OpenType Computer Modern with Greek and Cyrillic; additions requested.

`noto-emoji` in `fonts`
  Noto Color Emoji.

`qualitype` in `fonts`
  45 now-free OpenType fonts from Qualitype; a wide variety of designs.

`scholax` in `fonts`
  Extended TeXGyreSchola (New Century Schoolbook) with math support and more.

`twemoji-colr` in `fonts`
  COLR/CPAL-based color emoji from the Twemoji collection.

`wasy-type1` in `fonts`
  Type 1 version of Roland Waldi's `wasy` fonts, version 2.4.

graphics

`circuit-macros` in `graphics`
  M4 macros for electric circuit diagrams.

`chemplants` in `graphics/pgf/contrib`
  Process flow diagrams of chemical processes.

`kblocks` in `graphics/pgf/contrib`
  Typeset control block diagrams and signal flow graphs.

`lie-hasse` in `graphics/pgf/contrib`
  Draw Hasse diagrams for Lie algebras.

`pinoutikz` in `graphics/pgf/contrib`
  Draw chip pinouts.

`tikz-3dtools` in `graphics/pgf/contrib`
  Manipulate 3d coordinates and other tools.

`tikz-trackschematic` in `graphics/pgf/contrib`
  Create railway track diagrams.

`yquant` in `graphics/pgf/contrib`
  Typeset quantum circuits from a description that is human-readable.

info

`expose-expl3-dunkerque-2019` in `info`
  Article (in French) on using `expl3` to implement numerical algorithms.

language/japanese

`bxjatoucs` in `language/japanese`
  Convert character code values from Japanese encodings to Unicode.

language/korean

`pmhanguljamo` in `language/korean`
  Poor man's Hangul Jamo input method.

language/mongolian

`xecyrmongolian` in `language/mongolian`
  Basic support for Cyrillic Mongolian in Unicode.

macros/generic

`expkv-def` in `macros/generic`
  Define keys for `expkv`.

`expkv` in `macros/generic`
  Expandable key=val implementation.

macros/latex/contrib

`accessibility` in `macros/latex/contrib`
  Generate tagged and structured PDF files, with special support for KOMA-Script.

`algxpar` in `macros/latex/contrib`
  Support multiple lines of pseudo-code text.

`apa7` in `macros/latex/contrib`
  Format documents in 7th edition APA style.

`autofancyhdr` in `macros/latex/contrib`
  Compute `headlength` for `fancyhdr`.

`bearwear` in `macros/latex/contrib`
  Shirts for `tikzbears`.

`biblatex2bibitem` in `macros/latex/contrib`
  Convert BibLaTeX-generated bibliography to `\bibitem`s.

`bibleref` in `macros/latex/contrib`
  Format bible citations.

`brandeis-thesis` in `macros/latex/contrib`
  Class for Brandeis University M.A. theses.

`circledsteps` in `macros/latex/contrib`
  Typeset circled numbers.

`euclideangeometry` in `macros/latex/contrib`
  Extended `picture` environment for geometric ruler and compass constructions.

`fewerfloatpages` in `macros/latex/contrib`
  Produce fewer half-empty float pages. See article in this issue, pp. 54–68.

`fontsetup` in `macros/latex/contrib`
  Easily switch between different math fonts.

`fontsize` in `macros/latex/contrib`
  Set main document font to an arbitrary size.

`hep-paper` in `macros/latex/contrib`
  Support for high energy physics writing.

`hitszthesis` in `macros/latex/contrib`
  Template for bachelor dissertations at Harbin Institute of Technology.

`hvqrurl` in `macros/latex/contrib`
  Typeset the QR code for a url in the margin.

`langsci-avm` in `macros/latex/contrib`
  Attribute–value matrices and feature structures for linguistics.

`latino-sine-flexione` in `macros/latex/contrib`
  Support for Peano's Interlingua.

`leiletter` in `macros/latex/contrib`
Letter class for Leiden University.

`letterswitharrows` in `macros/latex/contrib`
Scalable arrows over math symbols.

`metastr` in `macros/latex/contrib`
Store and compose arbitrary strings.

`oops` in `macros/latex/contrib`
Framework for organizing definitions inline.

`pdfpc` in `macros/latex/contrib`
Support for `pdfpc` presentation viewer.

`physconst` in `macros/latex/contrib`
Macros for commonly used physical constants, per CODATA 2018.

`physunits` in `macros/latex/contrib`
Macros for physical units, including both SI and cgs.

`pmdb` in `macros/latex/contrib`
Poor man's database for building exams, homework, etc.

`rest-api` in `macros/latex/contrib`
Format a REST API description.

`schulmathematik` in `macros/latex/contrib`
Support for German-speaking teachers of math and physics.

`sdaps` in `macros/latex/contrib`
Creating machine-readable questionnaires processable with SDAPS.

`secnum` in `macros/latex/contrib`
Specify section numbering intuitively.

`shortmathj` in `macros/latex/contrib`
Shorten titles of mathematical journals.

`simplebnf` in `macros/latex/contrib`
Typeset Backus-Naur form expressions, possibly annotated.

`thorshammer` in `macros/latex/contrib`
Assessment based on AcroTEX quizzes.

`tkz-base` in `macros/latex/contrib/tkz`
Drawing tools for cartesian coordinate systems.

`verifica` in `macros/latex/contrib`
Typeset exercises, especially for Italian high schools.

`xkcdcolors` in `macros/latex/contrib`
Color names from the xkcd survey. (`xkcd.com/color/rgb`)

---

macros/latex/contrib/beamer-contrib/themes

`hitszbeamer` in `m/l/c/b-c/themes`
Harbin Institute of Technology beamer theme.

---

macros/latex/contrib/biblatex-contrib

`biblatex-ajc2020unofficial` in `m/l/c/biblatex-contrib`
BibLATEX style for the *Australasian Journal of Combinatorics*.

`biblatex-german-legal` in `m/l/c/biblatex-contrib`
Citation style for German legal texts.

`biblatex-jura2` in `m/l/c/biblatex-contrib`
Citation style for the German legal profession.

---

macros/luatex

`barracuda` in `luatex/generic`
Lua package to draw barcode symbols.

`emoji` in `luatex/latex`
Emoji support.

`lua-ul` in `luatex/latex`
Underlining for LuaLATEX.

`optex` in `luatex`
LuaTEX format based on plain TEX and OPmac.

---

macros/xetex/latex

`parsa` in `macros/xetex/latex`
Theses and dissertations at Iranian universities.

`xepersian-hm` in `macros/xetex/latex`
Fix kashida feature in `xepersian`.

---

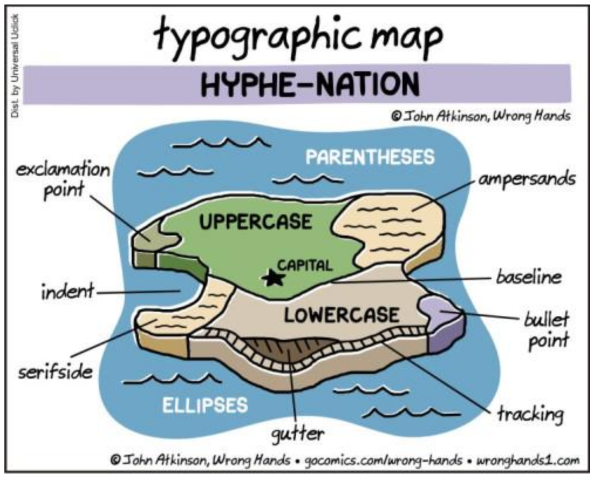
support

* `lua-uca` in `support`
Lua implementation of the Unicode collation algorithm.

`texlab` in `support`
Cross-platform Language Server Protocol, for LATEX code completion.

`texlive-dummy-enterprise-linux-8` in `support/texlive`
Dummy rpm to satisfy dependencies on TEX Live resources without additional installation.

`texplate` in `support`
Create document structure based on templates.



Comic by John Atkinson (`https://wronghands1.com`).

TUG financial statements for 2019

Karl Berry, TUG treasurer

The financial statements for 2019 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 slightly down in 2019 compared to 2018; this was the second year of offering trial memberships, and we ended the year with 1,238 members (16 more than 2018, a welcome rise). Contributions were up sharply, about $4,500, and product sales (mainly Lucida) and interest income were also up. Overall, 2019 income was up ≈ 3%.

Cost of Goods Sold and Expenses highlights; the bottom line

TUGboat production cost was up a little, due to page count. Postage-related expenses increased; other categories remained about the same.

The bottom line for 2019 was strongly negative, about $8,500, though still an improvement over 2018.

Balance sheet highlights

TUG's end-of-year asset total is down by around $5,160 (3%) in 2019 compared to 2018, 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 2019 Conference number is almost entirely from the TUG'19 conference, which essentially broke even (despite Palo Alto being our most expensive location to date). At this writing, we do not know if TUG'20 will be held.

The Prepaid Member Income category is member dues that were paid in earlier years for the current year (and beyond). The 2019 portion of this liability was converted into regular Membership Dues in January of 2019. The payroll liabilities are for 2019 state and federal taxes due January 15, 2020.

Upcoming

For 2019, we have increased the trial membership fee to $30; this will result in TUG approximately breaking even on trial members, instead of incurring a loss. We hope to continue to gain members; ideas are always welcome.

TUG 12/31/2019 (vs. 2018) Revenue, Expense

	Dec 31, 19	Dec 31, 18
ORDINARY INCOME/EXPENSE		
Income		
Membership Dues	76,125	77,825
Product Sales	5,238	3,672
Contributions Income	13,995	9,463
Interest Income	1,934	870
Advertising Income	345	270
Total Income	94,952	92,879
Cost of Goods Sold		
TUGboat Prod/Mailing	(18,836)	(17,410)
Software Prod/Mailing	(2,194)	(2,550)
Members Postage/Delivery	(2,236)	(1,470)
Lucida Sales to B&H	(1,965)	(1,465)
Member Renewal	(420)	(317)
Total COGS	(25,651)	(23,211)
Gross Profit	69,301	68,890
Expense		
Contributions made by TUG	(1,000)	(2,000)
Office Overhead	(13,642)	(14,301)
Payroll Expense	(63,091)	(63,078)
Interest Expense	(25)	(4)
Total Expense	(77,757)	(79,383)
Net Ordinary Income	(10,493)	(18,568)
OTHER INCOME/EXPENSE		
Prior year adjustment	(78)	
NET INCOME	(8,535)	(10,493)

TUG 12/31/2019 (vs. 2018) Balance Sheet

	Dec 31, 19	Dec 31, 18
ASSETS		
Current Assets		
Total Checking/Savings	171,560	176,530
Accounts Receivable	280	470
Total Current Assets	171,840	177,000
LIABILITIES & EQUITY		
Current Liabilities		
Committed Funds	47,270	44,442
TUG Conference	(100)	1,000
Administrative Services	1,498	2,698
Prepaid Member Income	9,175	6,375
Payroll Liabilities	1,301	1,353
Total Current Liabilities	59,244	55,869
Equity		
Unrestricted	121,131	131,624
Net Income	(8,535)	(10,493)
Total Equity	112,596	121,131
TOTAL LIABILITIES & EQUITY	171,840	177,000

TEX Consultants

The information here comes from the consultants themselves. We do not include information we know to be false, but we cannot check out any of the information; we are transmitting it to you as it was given to us and do not promise it is correct. Also, this is not an official endorsement of the people listed here. We provide this list to enable you to contact service providers and decide for yourself whether to hire one.

TUG also provides an online list of consultants at `tug.org/consultants.html`. If you'd like to be listed, please see there.

Aicart Martinez, Mercè
Tarragona 102 4^o 2^a
08015 Barcelona, Spain
+34 932267827
Email: `m.aicart (at) ono.com`
Web: `http://www.edilatex.com`
We provide, at reasonable low cost, LATEX or TEX page layout and typesetting services to authors or publishers world-wide. We have been in business since the beginning of 1990. For more information visit our web site.

Dangerous Curve
+1 213-617-8483
Email: `typesetting (at) dangerouscurve.org`
We are your macro specialists for TEX or LATEX fine typography specs beyond those of the average LATEX macro package. We take special care to typeset mathematics well.

Not that picky? We also handle most of your typical TEX and LATEX typesetting needs.

We have been typesetting in the commercial and academic worlds since 1979.

Our team includes Masters-level computer scientists, journeyman typographers, graphic designers, letterform/font designers, artists, and a co-author of a TEX book.

Dominici, Massimiliano
Email: `info (at) typotexnica.it`
Web: `http://www.typotexnica.it`
Our skills: layout of books, journals, articles; creation of LATEX classes and packages; graphic design; conversion between different formats of documents.

We offer our services (related to publishing in Mathematics, Physics and Humanities) for documents in Italian, English, or French. Let us know the work plan and details; we will find a customized solution. Please check our website and/or send us email for further details.

Latchman, David
2005 Eye St. Suite #6
Bakersfield, CA 93301
+1 518-951-8786
Email: `david.latchman (at) texnical-designs.com`
Web: `http://www.texnical-designs.com`
LATEX consultant specializing in the typesetting of books, manuscripts, articles, Word document conversions as well as creating the customized LATEX packages and classes to meet your needs. Contact us to discuss your project or visit the website for further details.

Sofka, Michael
8 Providence St.
Albany, NY 12203
+1 518 331-3457
Email: `michael.sofka (at) gmail.com`
Personalized, professional TEX and LATEX consulting and programming services.

I offer 30 years of experience in programming, macro writing, and typesetting books, articles, newsletters, and theses in TEX and LATEX: Automated document conversion; Programming in Perl, C, C++ and other languages; Writing and customizing macro packages in TEX or LATEX, `knitr`.

If you have a specialized TEX or LATEX need, or if you are looking for the solution to your typographic problems, contact me. I will be happy to discuss your project.

Veytsman, Boris
132 Warbler Ln.
Brisbane, CA 94005
+1 703 915-2406
Email: `borisv (at) lk.net`
Web: `http://www.borisv.lk.net`
TEX and LATEX consulting, training, typesetting and seminars. Integration with databases, automated document preparation, custom LATEX packages, conversions (Word, OpenOffice etc.) and much more.

I have about two decades of experience in TEX and three decades of experience in teaching & training. I have authored more than forty packages on CTAN as well as Perl packages on CPAN and R packages on CRAN, published papers in TEX-related journals, and conducted several workshops on TEX and related subjects. Among my customers have been Google, US Treasury, FAO UN, Israel Journal of Mathematics, Annals of Mathematics, Res Philosophica, Philosophers' Imprint, No Starch Press, US Army Corps of Engineers, ACM, and many others.

(continued)

Veytsman, Boris (cont'd)
We recently expanded our staff and operations to provide copy-editing, cleaning and troubleshooting of TeX manuscripts as well as typesetting of books, papers & journals, including multilingual copy with non-Latin scripts, and more.

Warde, Jake
Forest Knolls, CA, 94933, USA
6504681393
Email: `jwarde (at) wardepub.com`
Web: `http://myprojectnotebook.com`
I have been in academic publishing for 30+ years. I was a Linguistics major at Stanford in the mid-1970s, then started a publishing career. I knew about TeX from Computer Science editors at Addison-Wesley who were using it to publish products. Beautiful, I loved the look. Not until I had immersed myself in the production side of academic publishing did I understand the contribution TeX brings to the reader experience.
Long story short, I started using TeX for exploratory projects (see the website referenced) and want to contribute to the community. Having spent a career evaluating manuscripts from many perspectives, I am here to help anyone who seeks feedback on their package documentation. It's a start while I expand my TeX skills.

TeX Live 2020 news

Karl Berry

TeX Live 2020 was released online on April 10, 2020. The TeX Collection DVD is in process.

As new versions of packages are uploaded to CTAN, they are imported into TL, and available over the Internet via the `tlmgr` program. See the TL web site and documentation if needed.

As always, in the 2020 release there are pervasive updates to hundreds of packages and programs. We can't list them all, but here are some major user-visible changes in the principal programs.

General. The `\input` primitive in all TeX engines, including `tex`, now also accepts a group-delimited filename argument, as a system-dependent extension. The usage with a standard space/token-delimited filename is completely unchanged. The group-delimited argument was previously implemented in LuaTeX; now it is available in all engines. ASCII double quote characters (`"`) are removed from the filename, but it is otherwise left unchanged after tokenization. This does not currently affect LaTeX's `\input` command, as that is a macro redefinition of the standard `\input` primitive.

New option `--cnf-line` for `kpsewhich`, `tex`, `mf`, and all other engines, to support arbitrary configuration settings on the command line.

The addition of primitives to the engines in this and previous years is intended to result in a common set of functionality available across all engines; see LaTeX News #31 in this issue, pp. 34–38.

eptex, euptex. New primitives `\Uchar`, `\Ucharcat`, `\current(x)spacingmode`, `\ifincsname`; revise `\iffontchar` and `\fontchar??`. For `euptex` only: `\currentcjktoken`.

LuaTeX. Integration with HarfBuzz library, as new engines `luahbtex` and `luajithbtex`; `luahbtex` is now the engine used for the `lualatex` format.

Loading of dso (`.dll/.so`) objects is now forbidden if `--shell-restricted`.

New primitives `\eTeXglue[stretch|shrink]order`.

Scaling of virtual fonts supported.

Enhancements to `tex.runtoks`, which permits limited nested running of TeX.

pdfTeX. New primitive `\pdfmajorversion`; this merely changes the version number in the PDF output, and has no effect on PDF content.

`\pdfximage` and similar now search for image files in the same way as `\openin`.

pTeX. New primitives `\ifjfont`, `\iftfont`. Also in `eptex`, `uptex`, and `euptex`.

XeTeX. Fixes for `\Umathchardef`, `\pdfsavepos`, `\xetexinterchartoks`.

dvips. Output encodings for bitmap fonts, for better copy/paste capability.

tlmgr, TL infrastructure. Automatically retry (once) packages that fail to download, try to reinitialize LWP connection after several errors.

New option `tlmgr check texmfdbs`, to check consistency of `ls-R` files and `!!` specifications.

Use versioned filenames for package containers, as in `tlnet/archive/`*`pkgname`*`.rNNN.tar.xz`; should be invisible to users, but a notable change in distribution.

MacTeX. MacTeX and `x86_64-darwin` now require macOS 10.13 or higher (High Sierra, Mojave, Catalina); `x86_64-darwinlegacy` supports 10.6 and newer.

MacTeX is now notarized and command line programs have hardened runtimes, as now required by Apple for install packages. BibDesk and TeX Live Utility are no longer in MacTeX because they are not notarized, but a `README` file lists urls where they can be obtained.

More information, including urls for all references, is online.

◇ Karl Berry
`karl (at) freefriends dot org`
`https://tug.org/texlive`

Calendar

2020

Apr 24 – 25	Before & Beyond Typography: Text in Global & Multimodal Perspective, Stanford University, Stanford, California. `www.eventbrite.com/e/before-beyond-typography-text-in-global-multimodal-perspective-tickets-69068930029`
Apr 29 – May 3	BachoTeX 2020, 28th BachoTeX Conference, Bachotek, Poland. `www.gust.org.pl/bachotex`
Jun 4 – 6	Markup UK 2020. A conference about XML and other markup technologies, King's College, London. `markupuk.org`
Jun 7 – 12	19th Annual Book History Workshop, Texas A & M University, College Station, Texas. `library.tamu.edu/book-history`
Jun 8 – Jul 11	TypeParis20, intensive type design program, Paris, France. `typeparis.com` TypeCon / SoTA will fund a scholarship; see details at `typecon.com`
Jun 15 – 19	SHARP 2020, "Power of the written word". Society for the History of Authorship, Reading & Publishing. Amsterdam, Netherlands `www.sharp2020.nl`
Jun 17 – 19	Grapholinguistics in the 21st century — From graphemes to knowledge, to be presented online. `grafematik2020.sciencesconf.org`
Jun 29 – Jul 2	Book history workshop, Institut d'histoire du livre, Lyon, France. `ihl.enssib.fr`
Jul 1	**Preprints due for TUG 2020 program**
Jul 1 – 3	Eighteenth International Conference on New Directions in the Humanities, "Transcultural Humanities in a Global World", Ca' Foscari University of Venice, Venice, Italy. `thehumanities.com/2020-conference`
Jul 13 – 17	International Society for the History and Theory of Intellectual Property (ISHTIP), 12th Annual Workshop, "Landmarks of Intellectual Property". Bournemouth University, UK. `www.ishtip.org/?p=1027`
Jul 19 – 23	SIGGRAPH 2020, "Think beyond", Washington, DC. `s2020.siggraph.org`
Jul 22 – 24	Digital Humanities 2020, Alliance of Digital Humanities Organizations, Carleton University and the University of Ottawa, Ottawa, Canada. `adho.org/conference`

TUG 2020	**Cary Graphic Arts Collection, Rochester Institute of Technology, Rochester, New York**
Jul 23	Calligraphy workshop (participation limited).
Jul 23	Opening reception, 4:00–6:00 PM.
Jul 24 – 26	The 41st annual meeting of the TeX Users Group. `tug.org/tug2020`

Jul 27 – 31	Balisage: The Markup Conference, Rockville, Maryland. `www.balisage.net`
Aug 3	**Final papers due for TUG 2020 proceedings**
Sep 6 – 12	14th International ConTeXt Meeting, Prague-Sibřina, Czech Republic. `meeting.contextgarden.net/2019`
Sep 13 – 18	XML Summer School, St Edmund Hall, Oxford University, Oxford, UK. `xmlsummerschool.com`
Sep 29 – Oct 2	20th ACM Symposium on Document Engineering, San Jose, California. `www.documentengineering.org/doceng2020`
Oct 2 – 4	Oak Knoll Fest XXI, "Women in the Book Arts", New Castle, Delaware. `www.oakknoll.com/fest`

Owing to the COVID-19 pandemic, schedules may change. Check the websites for details.

Status as of 1 April 2020

For additional information on TUG-sponsored events listed here, contact the TUG office (+1 503 223-9994, fax: +1 815 301-3568, email: `office@tug.org`). For events sponsored by other organizations, please use the contact address provided.

User group meeting announcements are posted at `tug.org/meetings.html`. Interested users can subscribe and/or post to the related mailing list, and are encouraged to do so.

Other calendars of typographic interest are linked from `tug.org/calendar.html`.