TUGboat

Volume 42, Number 3 / 2021

General Delivery 223 From the president / *Boris Veytsman*

224 Editorial comments / *Barbara Beeton*
R.I.P. Walter Schmidt; R.I.P. Chuck Geschke; R.I.P. Rogério Brito; *Computers & Typesetting*; Hyphenation patterns and non-TEX uses; Update on *TUGboat* DOIs; News from GUT; An overview of TEX history; MacKichan Software is no more; More Knuth references

226 Michael D. Spivak, 1940–2020 / *Barbara Beeton*

228 How to keep your sanity when preparing a transcript of an online interview for publication / *Barbara Beeton*

Typography 231 Form, pattern & texture in the typographic image / *Charles Bigelow*

Multilingual Document Processing 247 Arabic text justification using LuaLATEX and the DigitalKhatt OpenType variable font / *Amine Anane*

Graphics 258 Rendering open street maps / *Hans Hagen*

267 Controlling captions, fullpage and doublepage floats: `hvfloat` / *Herbert Voß*

Fonts 284 Preventing tofu with pdfTEX and Unicode engines / *Frank Mittelbach*

287 The `unicodefonttable` package / *Frank Mittelbach*

LATEX 305 LATEX news, issue 34, November 2021 / *LATEX Project Team*

Hints & Tricks 310 Production notes / *Karl Berry*

311 A new unit for LMTX: The `dk` / *Hans Hagen*

312 The treasure chest / *Karl Berry*

Abstracts 313 *La Lettre GUTenberg*: Contents of issues 41–44 (2020–2021)

315 *Les Cahiers GUTenberg*: Contents of issue 58 (2021)

315 *ConTEXt Group Journal*: 14th meeting (2020)

Cartoon 316 Comic: Fontzie / *John Atkinson*

TUG Business 222 *TUGboat* editorial information

222 TUG institutional members

Advertisements 317 TEX consulting and production services

318 TEXnology Inc.

319 Pearson | Addison-Wesley

News 320 Calendar

TeX Users Group

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

Individual memberships
2021 dues for individual members are as follows:
- Trial rate for new members: $30.
- Regular members: $105.
- Special rate: $75.

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

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

Journal subscriptions
TUGboat subscriptions (non-voting) are available to libraries and other organizations or individuals for whom memberships are either not appropriate or desired. Subscriptions are delivered on a calendar year basis. The subscription rate for 2021 is $115.

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

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

[printing date: November 2021]

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`

Volunteer technical support:
`support@tug.org`

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

Gareth was at his old bench: [...] he was needed to set type for a few hours a day. [...] In his concentration and the fluidity of his movements, he looked to me like a painter or a composer, his placement of type as deliberate as notes on a sheet of music.

Kip Williams
The Dictionary of Lost Words (2020)

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP
EDITOR BARBARA BEETON

VOLUME 42, NUMBER 3, 2021
PORTLAND, OREGON, U.S.A.

TUGboat editorial information

This regular issue (Vol. 42, No. 3) is the last issue of the 2021 volume year. The deadline for the first issue in Vol. 43 is March 31, 2022. Contributions are requested.

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

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

TUGboat advertising

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

Submitting items for publication

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

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

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

Other TUG publications

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

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

TUG Institutional Members

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

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

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

Aware Software, *Newark*, *Delaware*

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

CSTUG, *Praha*, *Czech Republic*

Duke University Press, *Durham*, *North Carolina*

Hindawi Foundation, *London*, *UK*

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

L3Harris, *Melbourne*, *Florida*

Maluhy & Co., *São Paulo*, *Brazil*

Marquette University, *Milwaukee*, *Wisconsin*

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

Nagwa Limited, *Windsor*, *UK*

Overleaf, *London*, *UK*

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

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

TEXFolio, *Trivandrum*, *India*

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

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

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

VTEX UAB, *Vilnius*, *Lithuania*

From the president

Boris Veytsman

My previous column discussed various licenses used for TeX-related software, and mentioned that 56% of CTAN packages use the LPPL, the license developed by the LaTeX3 project. The LPPL insists on the right of the users to know whether they are dealing with the original package or with a modified version — and the right of the maintainers of a package to preserve its integrity. This insistence is quite unusual in the other parts of the free software movements. I can fork Emacs, change its functions (for example, mapping Ctrl-X Ctrl-S by default to the deletion of all files in the current directory), and still distribute the result as Emacs. I cannot do this with the current LaTeX packages, however: I can change the code, but I must prominently advertise to the users that they are *not* using the "real LaTeX".

While technically this license was created for LaTeX and its packages, its spirit is, in my opinion, traditional for the TeX world in general. Knuth's distribution conditions for the original TeX code were basically "you may do whatever you want with this program, but if you want to call it TeX, it must pass this suite of tests". The old-timers may remember a quite strong reaction from DEK when certain Linux distributions included a copy of Computer Modern fonts with slightly modified shapes. It is not surprising, however, that the Latin Modern fonts, which are based on Computer Modern, but do not pretend to *be* Computer Modern, have not caused objections by Knuth.

This feature was met with certain resistance in the free software community. I remember it being dismissed as an attempt at "a poor person's trademark", and it was questioned whether LPPL is too limiting to be considered free software. Of course any license with the exception of the purely public domain CC-0 limits the rights of users (some people, for example, consider the requirements of the GPL far too onerous). The question is in the balance between these limitations and the benefits to the community. The LaTeX team was able to convince the gatekeepers of free software licensing that their license has an acceptable balance.

One of the reasons for this insistence on integrity of software in the TeX world may be the following. As Chris Rowley observed a decade ago, TeX and especially LaTeX became *the* language of the scientific community. Any language serves not just the communication of information, but also its preservation. One of the concerns of DEK during the design of TeX was the requirement that documents should be typeset identically on all computers. It is well known that this led to TeX using integer arithmetic with the unit of length being 5.4 nm. If we want our documents to be the same on all computers, we also want them to be the same in all times. The TeX community has always put a great emphasis on the stability of historical documents, trying to preserve them with nanometer scale accuracy. Of course, macro packages written by ordinary people rather than DEK are not as stable as his TeX, but the effort to make the old documents compilable, producing the same results today as many years ago is quite extraordinary for the software community.

To tell the truth, many software developers are not too concerned about backward compatibility or legacy code. While the sheer disdain that Python maintainers seem to feel towards their users is rather rare, the expectation that users constantly adjust their programs to keep them working is quite common. The huge effort that the LaTeX3 team puts into compatibility with old documents is almost unheard of. This work would be impossible without the guarantee of integrity provided by the LPPL.

Another aspect of this is the attention the community pays to continuing maintenance of old packages when the original author moves on. The Oberdiek Packages Support Group (`github.com/ho-tex`) is a great example of this. A number of Heiko's packages became key libraries for much LaTeX code. Due to the stewardship of the Group we can be assured these packages are going to be supported in the future. Now it looks like we might need another effort in this direction. Peter Wilson, the author of *memoir*, *fonttable*, *ledmac*, *ledpar*, *cutwin* and 83 other packages including unique fonts, document styles and much more, retired some time ag. Many have been adopted by other people and groups, with the majority taken by Will Robertson. Now Will, after years of great work maintaining these packages, wishes to focus on other tasks. There are now talks with the LaTeX team about taking them over. We also need maintainers for other old but still important code by other authors.

We are trying to prevent code from becoming unmaintained. Thus if you are a package author, and you feel you are no longer able to support your work, or if you are looking for a package to maintain, please contact us. We will be happy to match outgoing and incoming maintainers, and help to continue the TeX tradition of care about history and preservation.

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

doi.org/10.47397/tb/42-3/tb132pres

Editorial comments

Barbara Beeton

R.I.P. Walter Schmidt

Walter A. Schmidt (15 August 1960–12 October 2021) was best known in the TeX community for his support of fonts, in particular the support of PostScript standard fonts via the `psnfss` package. An article in *The PracTeX Journal*, "Font selection in LaTeX, The most frequently asked questions" (`tug.org/pracjourn/2006-1/schmidt/`), remains useful after 15 years, although some fonts mentioned there have been superseded by newer OpenType versions. His listing on CTAN (`https://ctan.org/author/schmidt`) shows 41 entries, including the German translation of the "Short Introduction to LaTeX 2_ε". He reorganized and updated the font metric files for the Lucida Type 1 distribution from TUG and PCTeX. He was a long-time member of DANTE, and one of the organizers of the Stammtisch (monthly local meeting) in Erlangen.

R.I.P. Chuck Geschke

Charles Matthew ("Chuck") Geschke (11 September 1939–16 April 2021), together with John Warnock, founded Adobe Inc. in 1982, after failing to convince Xerox that their work at Xerox PARC on the page description language Interpress was commercially viable.

Based on this work, Adobe developed PostScript, which was adopted by Apple and integrated into one of the first desktop publishing systems. The integration of a laser printer with a personal computer provided an attractive platform for individual use of TeX. In 1992, PostScript was followed by the Portable Document Format (PDF). Although there was resistance to PDF for some years, while emphasis was on the dynamic Web, the importance of print was finally recognized, and PDF is now the default format for nearly all print applications, personal or at commercial scale.

TeX has been used pervasively with both PostScript and PDF almost since the formats became available. So Geschke's contribution to the TeX world, although indirect, was clearly a crucial one.

R.I.P. Rogério Brito

Rogério Theodoro de Brito, of São Paulo, Brazil, succumbed to COVID-19 in April 2021. He was the maintainer of the `algorithms` bundle of packages (`algorithm` and `algorithmic`) and participated in other free software projects, in particular Debian, where he was a contributor for more than 15 years.

Computers & Typesetting

The entire series of *C&T* has been updated following the 2021 tuneup (`tug.org/TUGboat/tb42-1/tb130knuth-tuneup21.pdf`). In addition to hardcover volumes (the softcover versions have been discontinued), e-versions (PDF) have been created; some users may find these more convenient than a physical book.

TUG members are eligible for a substantial discount on the printed volumes or the e-volumes until the end of the year. (A discount on an e-plus-paper combination is available to all and the TUG discount may not be added.) See the notice by Pearson/Addison-Wesley at the end of this issue.

Hyphenation patterns and non-TeX uses

During the past month, considerable activity has taken place on the list `tex-hyphen@tug.org`. The usual traffic consists of notices regarding updates to existing patterns for various languages and announcements regarding additional languages. (There are currently patterns for around fifty languages on CTAN, some with multiple versions supporting historical dialects or variations due to spelling reforms.) But the recent flurry has been concerned with the potential use of the patterns for projects and applications not related to TeX, and questions regarding the implications of the various licenses that are attached to the pattern files. I wasn't aware that use of the TeX hyphenation patterns is so widespread,

A thorough exposition of the hyphenation effort appeared in the 2016 article "Hyphenation in TeX and elsewhere, past and future", by Mojca Miklavec and Arthur (Reutenauer) Rosendahl, `tug.org/TUGboat/tb37-2/tb116miklavec.pdf`. This both addressed TeXnical considerations and presented an extensive discussion on the available licenses, in particular the LPPL. While some of the provisions of the LPPL are highly desirable (such as the provisions relating to having a recognized maintainer), it was determined that in other respects the LPPL is not ideal for files of hyphenation patterns.

The recent discussions have also pointed out that neither the GPL nor the LPPL are typically desired for hyphenation patterns, barring personal choice, since they require that any project incorporating a module under their license must itself be licensed compatibly in its entirety.

That cuts down the number of suitable licenses, and it appears that the MIT license is the most appropriate in this case. A request was sent out through the list to maintainers of the pattern files to (re)consider the licenses applied to their patterns, and responses concerning license changes are still

doi.org/10.47397/tb/42-3/tb132beet

coming in. Some maintainers have updated their license and submitted the updated version directly to CTAN, while others have requested that this be done by the team maintaining the hyphenation pattern infrastructure.

To determine the current license status for a particular set of patterns, it's best to check the files of interest on CTAN. The list archives are public, so the discussion can be reviewed at `lists.tug.org/tex-hyphen`.

Update on *TUGboat* DOIs

As reported in volume 41:3, Document Object Identifier (DOI) information is being added to the *TUGboat* archive. Going forward, the identifier appears below the bottom of the first column of each item to which a DOI has been assigned; earlier content will not be reprocessed to add this notation.

For all issues in the archive, the notation "(doi)" will be added for each item in the online TOC for the issue, linked to a separate page that contains the bibliographic information for the item as well as a summary of its content and the reference list, if one is present. This page provides a quick way for a prospective reader to decide whether to read the full article, and will be present even for articles that are still closed to non-members. These pages are now present for issues starting with 41:3, and will be created for earlier issues as (human) time permits.

News from GUTenberg

GUTenberg, the French TeX user group, has been reconstituted and resumed issuing its publication, *La Lettre*. The latest issue, No. 44, can be read online from a link on the organization's website, `www.gutenberg.eu.org`. As for previous issues, a single font was chosen to set the PDF version. The font for No. 44 is Libertinus, a variant of Linux Libertine; an article in the issue gives the history and a description of the distinct features of both the original and the other members of the family. (An illustration of the available ligatures includes "fj", a feature usually absent.)

Also linked from the website is a collection of videos from GUTenberg's June meeting, which took place online.

An overview of TeX history

An online document by Arno Trautman, still under construction, promises to untangle all the terminology associated with TeX since its creation. Entitled "An overview of TeX, its children and their friends ...", it can be found at `github.com/alt/tex-overview/blob/master/tex-overview.pdf`.

Still lacking a table of contents, active links, and other features expected from a "finished" document, it is nonetheless a useful reference. The author invites contributions and corrections. I expect to give it a go, and suggest that other longtime TeX users might find this worthwhile as well.

MacKichan Software is no more

MacKichan Software, creators and suppliers of the Scientific Word and Scientific Workplace software, has ceased operation. The notice on their website, `mackichan.com`, carries this information:

> Scientific Word 6.1 for Windows is now available for FREE
>
> A link to it and some installation programs is available HERE. In time, the source will be posted on Github.

These two programs provided an input format that many users found easier to master than direct LaTeX input, and produced as output a LaTeX file that could be submitted directly to many journals. Scientific Word was essentially a word processor; Scientific Workplace included a third-party computer algebra system.

Details of the closure are given on the cited website.

More Knuth references

A featured article in the August 2021 issue of *The American Organist*, entitled "A Pipe Dream Come True", was written by Jan Overduin, the organist who performed the premiere of Don's *Fantasia Apocalyptica* at Don's 80th birthday celebration in Sweden. Both a personal account of his interaction with Don, how the collaboration came to be, and a detailed description of the work, Overduin's introduction conveys his enthusiasm for its concept and realization. Examples of the musical references include quotations from modern popular music as well as traditional religious and secular works. Although the essay is written for a musically literate audience, it should be comprehensible to anyone who appreciates music, even without specialized training.

Finally, yet another interview about Don's work as a computer scientist has been posted online at `youtube.com/watch?v=lFkyhz_yDCs`. Part of the Simons Institute series on the Foundations of Computing, it covers both old and new territory. A reference has been added to the growing list of links to Don's interviews on the TUG website (`tug.org/video`).

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

Michael D. Spivak, 1940–2020

Barbara Beeton

Michael David Spivak was born May 25, 1940, in Queens, New York. He died October 1, 2020, in Houston, Texas. He suffered a broken hip earlier in the fall, and had been confined to an extended care facility following that mishap.

I met Mike when I was sent to Stanford in the summer of 1979 to learn TeX. A house had been rented on the Stanford campus for the month of July to accommodate a small contingent whose remit was to learn TeX and construct a working environment that could be used in production of AMS books and journals. Dick Palais, then the chair of the AMS Board of Trustees, was in charge of the group; it was Dick who had learned of TeX from Don Knuth's Gibbs Lecture at the 1978 annual meeting [1] and realized that this was a program directly applicable to AMS publications.

The rest of the crew in this little commune included these individuals, all of whom became active in TUG at its founding:

- Robert A. (Bob) Morris [2], who was to develop the macro interface to format the math structures that appear in AMS journals;
- Michael Spivak, who was charged with documenting the macros in a user manual for authors and their secretaries;
- Rilla Thedford, from *Math. Reviews*, to learn what would be needed to produce *MR* internal documents and ultimately, *MR* itself;
- myself, to learn how to install TeX at the AMS Providence headquarters and how to use it to develop macros first to produce "administrative publications" (including the AMS publications catalog and the journal *Notices*) and then journals and books.

The reason for Mike's assignment was his known ability to write clearly on mathematical topics, in particular evidenced by his five-volume set, *A Comprehensive Introduction to Differential Geometry*, work for which he was awarded the 1985 Leroy P. Steele Prize in Expository Writing [3].

For reasons unknown to me, Bob Morris decided not to complete his assignment of writing the macros. Mike took over this task, and not only produced the documentation, but also developed a comprehensive and well-designed set of macros. He had a finely developed sense for what math should look like on the printed page, as well as a sensitivity for naming mathematical structures in a way that would be familiar to a mathematician, building on the strong base provided by Don Knuth. Together, these strengths have contributed to the acceptance of TeX as a *lingua franca* among mathematicians.

Working together, Mike and the inhouse AMS technical staff developed the macros to produce a "preprint" style, `amsppt.sty`, which together with the math macros provided a structure for producing AMS journals in their accustomed format. (The goal was to be able to switch from the previous system to TeX in such a way that the change would be immediately noticeable only to readers who were paying extremely close attention.) The first formal edition of *The Joy of TeX* [4] was published in 1982, and contained the instructions that would be needed for authors or their secretaries to be able to prepare manuscripts in AMS-TeX for publication in AMS journals. (The first all-TeX issue of the AMS *Transactions* was printed in January 1985, following more than a year of experimentation and pre-production work.)

Joy is special in a number of ways. The title is a play on the title of a then-popular book [6] on a quite unrelated subject, and *that* book's title was a play on *Joy of Cooking* [5], a respected, time-honored, and well-organized recipe book. (One part of *Joy of TeX* returns to that source with the heading "Sauces and Pickles".) Pronouns are gender-neutral — *E*, *Em*, *Eir* — but even though these are now called "Spivak pronouns", when I asked, Mike said he didn't originate them. (On the other hand, that his name attached to them indicates that his use was widely noticed.) The material covered is clear and easy to follow. As in *The TeXbook*, the appendices were named alphabetically (A was "Answers To All The Exercises", B, "Bibliographies", ..., G, "{TeX Users}"); sadly, when the second edition was prepared at AMS, what was originally the last section of the main text was moved to Appendix A, compromising the alphabetical alignment. The second edition updated the technical coverage of the macros (which *were* Mike's work), but he had no part in the updating of *Joy*, which was the work of the AMS editorial staff. Long after AMS-TeX was superseded by AMS-LaTeX, AMS-TeX was no longer accepted or supported by AMS, and with Mike's permission, a PDF copy of the "final" corrected edition of *Joy* was posted to CTAN [4].

The original macros were documented by Mike, but he initially refused permission to post this material on CTAN. However, many years later, Mike did relent, and the documentation files were added to the CTAN collection [7] early in 2019.

AMS-TeX was the production workhorse at AMS for several years, but it had obvious limitations that were noted by users with increasing frequency. The

doi.org/10.47397/tb/42-3/tb132beeton-spivak

most serious was the absence of automatic numbering and cross-referencing facilities. If theorems were to be numbered, or referred to later, that had to be input by hand, with the obvious chances for errors. The same was true for displayed equations and bibliographic references. These gaps became a real problem if an author decided to rearrange the exposition.

By 1990, LaTeX, which *did* have automatic numbering and cross-referencing capabilities, was in wide use, and with increasing pressure from authors, AMS gave in, and commissioned the adaptation of the math macros into LaTeX, resulting in what is now the `amsmath` package, a required part of LaTeX.

In the font realm, Mike favored Times Roman, which had been used to set math journals and textbooks for many years before TeX and Computer Modern came along. Being quite particular about the appearance of math on the page, he created his own variation of Times, with a full complement of stylistically compatible symbols, which he called MathTime Professional 2, or MTPro2. This was made available through Personal TeX (later PCTeX), a small supplier of TeX software located in northern California. The founder and owner of this company was Lance Carnes [8], someone who is also well known in the TeX community, and who has helped me by checking this remembrance (thanks, Lance).

Mike disagreed with the manner in which LaTeX implemented the text-related features, and instead devised his own methods, which he implemented in a structure he called LAMS-TeX, with an accompanying manual, "LAMS-TeX — the Synthesis". But LaTeX had soaked too deeply into the TeX publishing fabric, and LAMS-TeX never became the hoped-for alternative. The LAMS-TeX macros are posted at CTAN, but the sources for the manual were lost. We have found a scanned copy, and will try to obtain permission from Mike's estate to post it on CTAN so that the curious don't have to work so hard to try it.

Mike was active in TUG for several years at the beginning. He was a founding member of the Board (at the time known as the "Steering Committee") and was acting Chairman from 1981–1983 while Dick Palais was on sabbatical. He left the Board in 1985.

In his "other" life, Mike founded Publish-or-Perish Press, which published several of his books, both new works and updated versions or older works first issued by other publishers, as well as books by other authors [10]. (Thanks to Bob Palais for helping to find this source.) Although further information on the press itself is not easy to find, it is known that academic libraries all over the world still hold copies of many of his works, and they are still referred to by readers needing clear, reliable instruction on topics in math and physics.

References

[1] Knuth, Donald E., "Mathematical typography", *Bull. Amer. Math. Soc. (N.S.)* **1** (1979), no. 2, pp. 337–372. `https://ams.org/journals/bull/1979-01-02/S0273-0979-1979-14598-1`

[2] Morris, Robert (Bob), Obituary, *TUGboat* **42**:1 (2021), p. 4. `tug.org/TUGboat/tb42-1/tb130beet.pdf`, and Interview, `https://tug.org/interviews/morris.html` (2018).

[3] "1985 Steele Prizes Awarded at Summer Meeting in Laramie", *Notices of the Amer. Math. Soc.*, **32** (October 1985), no. 5, pp. 575–576. `https://ams.org/journals/notices/198510`

[4] Spivak, M.D., Ph.D., *The Joy of TeX, A Gourmet Guide to Typesetting with the AMS-TeX macro package*, 2nd edition, reprinted with corrections, American Mathematical Society, Providence, RI, 2004. `https://ctan.org/pkg/joy-of-tex`

[5] Rombauer, Irma, and Marion Rombauer Becker, *Joy of Cooking*, Fifth revision, Bobbs-Merrill Company, Indianapolis, IN, 1975.

[6] Comfort, Alex, M.D., Ph.D., *The Joy of Sex, A Gourmet Guide to Love Making*, Crown, New York, 1972.

[7] CTAN package AMS-TeX, `https://ctan.org/pkg/amstex`

[8] Carnes, Lance, Interview, `https://tug.org/interviews/carnes.html`, 2004.

[9] CTAN package LAMS-TeX, `https://ctan.org/pkg/lamstex`

[10] Books published by Publish or Perish, `https://openlibrary.org/publishers/Publish_or_Perish`

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

How to keep your sanity when preparing a transcript of an online interview for publication

Barbara Beeton

Abstract

Three interviews conducted during TUG 2021 online were transcribed and edited for publication in the conference proceedings. Accomplishing this to the desired quality proved far more difficult than anticipated. The reasons for this are presented here, along with lessons learned that might enable cooperation of future interview participants, both interviewer and subject, in making this process more straightforward and less painful for the editor.

Background

Over many years I've had direct or indirect experience with these transcription mechanisms:

- transcription by experienced court stenotypists;
- transcription directly from an audio recording (tape or computer, including cell phone);
- editing an auto-transcription from an online video service (Zoom and YouTube).

In most cases, I was present (either in person or online) at the original presentation, so I was familiar with the subject matter, or had at least heard it myself. In all but the first situation, I was the individual responsible for preparing the transcript for publication. It's not for the faint of heart.

On the assumption that most interviews these days are conducted online, or the result is posted and viewable there, most of what follows will be specific to that medium.

Overview

The TUG 2021 interviews were conducted electronically, over Zoom, with an additional transmission via YouTube, and recorded for future viewing. The fact that interviewer and subject, and in some cases attendees asking questions, were not in the same location raised some complications with respect to communication.

Only one of the interviewees was a native speaker of (British) English, as was one of the individuals involved in later discussion. The other participants were from varied linguistic backgrounds; all of them are fluent in English, but most accents were quite distinct from the U.S. English norm. A built-in complication was the subject matter, which was highly technical, and not all familiar to me.

Both online services provided auto-transcriptions as starting text. Neither was ideal, but the difficulties were quite different between the two.

The "automatic" transcriptions

Since participants in the Zoom thread had to be registered, their names were known, and were present in the transcript. A new "paragraph" was started with each change of speaker, with a blank line between two entries. If a speaker continued talking for a relatively long time, or the discourse was interrupted by a brief silence, the contribution might be broken by additional blank lines. These segments were numbered consecutively in the transcript, and for each segment, the starting and ending times (relative to the start of the file at 00:00) were given. Occasionally, at a transition, a word or two would be assigned to the wrong speaker, but in general, as long as more than one person wasn't talking at once, the speaker identification was accurate. The accuracy of the text, on the other hand, left much to be desired. More about that later.

The YouTube auto-transcript was quite different. The identity of the speakers wasn't known, and no attempt was made to mark a change of speaker. The text was presented like a "stream of consciousness" in strings of irregular length separated by blank lines. On my monitor, the edit windows are usually set to a width of 80 characters, and a run-on line is ended with a (meaningless) backslash, for an effective length of 79 characters per line. At least one "line" in one transcript was 88 lines long, or nearly 7,000 characters. No useful punctuation (except for an occasional period in a url). And the accuracy of the text was no better in general than what was provided by Zoom.

Aside from the "flow" and presence or absence of speaker identification, the content was far from identical. Both systems were equally unfamiliar with the specific technical environments represented by the three interviewees. The only way to obtain a script worthy of publication was to start with the video recordings and listen carefully. In that respect, there was not much difference (although the YouTube recording of one interview was lost when the session was not closed before 12 hours had elapsed).

Consistent lapses

The terms TeX and LaTeX were spoken frequently throughout the interviews, but appeared almost nowhere in the transcripts. Instead, "tech" was a frequent substitution, as were "later", "late tech", and "late hack". Preliminary edit passes, searching for "tech" and "late", were effective in eliminating these misinterpretations, along with "TEX Live", "MiKTEX", "WriteLATEX", "ShareLATEX", and a few other compounds.

Company names were also consistent failures. "Overleaf" sometimes survived, but also occurred frequently as "overly"; another search, for "over", took care of that. "Fujitsu" didn't fare so well, and couldn't be cleaned until a comparison was made with the recording; I think my favorite miscue was "42" instead of "Fujitsu".

A few other terms occurred frequently enough to be attacked by a global search and explicit replace, but mostly they weren't discovered until the voice/text comparison. When such a case did arise, the word-for-word comparison was interrupted for a more targeted cleanup.

To be able to recognize and correct such lapses, it's highly desirable that the person editing the transcript be familiar with the speaker and the general subject of the interview; without this knowledge, it will likely be necessary to ask the interviewee to provide the needed corrections. Another problem area is people's names; here again a close personal knowledge of the interviewee is useful.

Mechanical considerations — know your equipment

Now it's time to attack the details of the text, so that the final transcript records the interview accurately. Unless you can type as fast as people talk, it will be necessary to stop the audio from time to time in order to catch up. Other reasons to stop are so that you can listen again to something that isn't clear the first time, or to verify a passage against another source.

How to reset the position in the audio file may be a puzzle. It took me several tries to find out that the back arrow on my keyboard could move the recording back in 15 second increments. This was far more efficient than trying to position the slider to align with the timing reported in the Zoom transcript, although using the timer was effective when the goal was to review a larger section. A few minutes of practice before starting can pay off handsomely later.

Details of the text

As noted earlier, the texts of the Zoom and YouTube transcriptions were not alike physically:

- Zoom: segmented, timed, speakers identified, sentence structure marked.
- YouTube: run-on, no speaker ID, no punctuation or case differentiation, interminable strings of words.

The textual content was far from identical as well. When a word (often a technical term) was unknown to the system, its representation in one text might be quite different from what appeared in the other. This turned out to be useful when the version chosen as the starter text made no sense, and the likely meaning couldn't be determined from the audio; it was usually possible to check what was in the other version and come to a sensible conclusion. For technical terms, YouTube was slightly better. However, homing in on the same passage was not easy; finding a matching term near the questioned material that could be used to search in a file that is just a jumble of words involves careful guessing.

Another weakness is the possibility that the interview participants are not skilled at this activity. An unscripted, unrehearsed interview may be littered with repetitions, meaningless interjections ("uh", "I mean"), and even an occasional interruption. While the primary goal of a published transcript is to record the content accurately, the result should also make sense if read without prior exposure to the event. Here is where editorial intervention is required. Consider carefully whether that "I mean" is just filler, or does in fact mean that the speaker is trying to clarify a particular point.

Occasionally, especially in an online interview, there may be unexpected interruptions. If an interruption is relevant to the topic of the interview, it can be worthwhile to include the details in the transcript. However, if it isn't relevant, and the interruption is short enough, it can be omitted; a longer interruption can be noted briefly in a [bracketed comment]. The choice depends on an estimation of whether noting or omitting would be more disruptive to someone reading the transcript and watching the interview at the same time.

Some details, finally, will require explanation or confirmation by the speakers themselves. An instance in the TUG 2021 interviews was when one of the interviewees referred to colleagues by only their first names. Since I don't know these individuals, direct contact was necessary. That said, it's always a good idea to ask a subject to review the transcript before publication to avoid surprises.

Examples of misinterpretation

As mentioned earlier, familiarity with the subject matter is a great advantage. I encountered this long ago, when observing the result of a symposium on mathematical physics, recorded by experienced court stenotypists. The transcribed phrase "brownie in motion" was determined to mean "Brownian motion". (The stenotypists would undoubtedly have produced letter-perfect transcriptions for medical terminology.) Be warned.

Here is a not entirely random sample of terms that led to head-scratching in the TUG 2021 interviews.

spoken	Zoom	YouTube
a local editor	a low planetary	a local editor
BachoTEX	Bangladesh	bob attack parody
`biblatex`	the block back	people attack
CTAN	Stacy time and tse-tung	say see town sita ctan
Fujitsu	42	fujitsu
John Lees Miller	john these Miller	john lee's miller
LaTeX	late night	latex, later late act
Overleaf	overly	overly
Overleaf usage	obese usage	overleafs usage
quarantine	current time	current time
ShareLaTeX	show a tech	chelatec
TeX Live	deck live tech live	deck live tech live
WriteLaTeX	right lasik right later	right latex

Suggestions for a prospective editor

0a. Make sure your equipment and support software are in good working order. You should be a competent user of your editing software, and ideally, this software should be designed for use with TeX text files.

0b. If you haven't already listened to the session, do so, completely, before starting to work on the transcript. Becoming even slightly familiar with the individuals involved, their manner of speaking, and the subject matter is worth the time and effort.

1. Collect all recordings (audio or video) and text files in a convenient area. If the interview is part of a larger recording, remove any unrelated material from beginning or end, so that only the relevant content will be part of the working set.
2. Create a "working" copy with a new name. If more than one auto-transcript is available, choose the one that provides the text in the form closest to the final product. "Lock" all original files so that they can't be changed.
3. Analyze the text for consistently misinterpreted items. Fix these globally in whatever manner is most efficient and accurate. There may be a function available with your chosen editor, or a "search-and-replace" utility (such as `sed` on Unix).
4. Now you are ready to compare the text file to the recording. Review the mechanism for stopping the recording quickly and backing up just a few seconds.
5. Update the text. Listen to the recording while reading the corresponding text. Make corrections as necessary. If something is unclear, go over it again, and if it still doesn't make sense, refer to an alternate transcript if you have one, or leave a comment in the file for later attention, and keep a separate list of questions.
6. It was suggested by a reviewer that "waypoints" (timing indicators) be inserted in the transcript so that readers can find locations in the video if they want. Since this was not done in the transcripts that led to this article, I'm unable to offer specific suggestions on how to do this in a way that doesn't detract from the natural flow.
7. Process the file to "final" form, and reread it, preferably while listening to (and watching) the recording. Make additional corrections as needed, and clean up stammering, repetitions, etc., that would cause confusion for someone reading (but not watching) the interview for the first time.
8. Ask the participants to review the result, being specific about questions that arose during the editing. After approval, make any final corrections and process for the final release.

There are organizations that offer transcription services for a fee. It might be worth considering use of such a service. Even if the transcriptionist is not familiar with the technical details discussed in the interview, the resulting text is likely to be much closer to what was actually said than what is produced by Zoom or YouTube.

References to the three interviews, both printed transcripts and videos, that led to this article can be found at `tug.org/TUGboat/tb42-2`.

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

Form, pattern & texture in the typographic image

Charles Bigelow

Abstract

In this essay we examine two fundamental typographic principles, size and combination, and show that out of their interaction emerge three qualitative levels of the typographic image: form, pattern, and texture.

Preamble

This essay was originally published in 1989 in the journal *Fine Print: The Review for the Arts of the Book* (vol. 15, no. 1). In the intervening three decades, the terrain of typography has shifted, and many books, as well as other forms of typographic texts, are now read on digital displays, not in print. Discussions of typographic art must now consider analog and digital, in-print and on-screen, long-form and short-form texts. Hence, looking back on this essay of thirty-two years ago, I see that some of my views have changed in light of new media, new knowledge, and new discoveries about old knowledge.

The parts about vision science could and certainly should be updated. Since the 1980s, reading scientists have discovered more about the visual processes underlying reading, and many recent findings appear relevant to understanding typography, and may at least expand our perception of its richness. But, to rewrite this essay to include all the newer technological developments in typography and the scientific discoveries pertaining to reading would force it into a book length disquisition. Instead, the essay appears here as originally written because its basic distinctions are, I believe, still the same; the shift to digital typography has not changed them.

There are a few additional references and illustrations that touch on selected topics in reading science. Also, some of the original illustrations have been replaced with slightly different images, but their purposes within the essay are unchanged. The endnotes are marked in the main body of text by numbers in parentheses, e.g., (1), and then listed by number at the end of the essay.

The original print essay was composed in Lucida Bright, a typeface design first introduced in Scientific American magazine in 1987 and later released for general use. This republication for *TUGboat* uses Lucida Book, a new variant of Lucida, still under development but making its inaugural appearance here.

1 Introduction

In her famous essay, “The Crystal Goblet, or Printing Should Be Invisible”, Beatrice Warde argues that typography is not an art.

> Type well used is invisible as type… That is why it is mischievous to call any printed piece a work of art, especially fine art: because that would imply that its first purpose was to exist as an expression of beauty for its own sake and for the delectation of the senses. … printing in English will not qualify as an art until the present English language no longer conveys ideas to future generations and until printing itself hands its usefulness to some yet unimagined successor. (1)

In a lesser-known but no less important essay, “Clay in the Potter’s Hand”, Jan Tschichold argues that typography is an art:

> Decisions on matters of higher typography, such as in a title page, need a really highly developed taste, related to what is needed in creative art. They may produce forms which are quite as perfect as good painting or sculpture. From the experts they should receive even more respect since the typographer is more strictly bound than any other artist by the unchangeable wording of the material before him. None but a master can call the dead leaden letters to true life.
>
> Perfect typography is certainly the most elusive of all arts. Out of stiff, unconnected little parts a whole must be shaped which is alive and convincing as a whole. Sculpture in stone alone comes near in its obstinacy to perfect typography. For most people it offers no special aesthetic charm as it is as difficult of access as the highest music… (2)

Warde bases her argument on pragmatics: because typography is useful, because it conveys ideas, it cannot be art, for art is, by implication, aesthetic and sensual rather than utilitarian and rational. Tschichold bases his argument on sophistication, in the sense of complexity or refinement: because typography can be as developed, perfect, difficult, and elusive as the fine arts, it must be an art itself.

In claiming that the typographic whole is constructed from elementary parts, that it is greater than the sum of its parts, and that it can be alive, Tschichold is espousing “holism”, a philosophy

doi.org/10.47397/tb/42-3/tb132bigelow-form

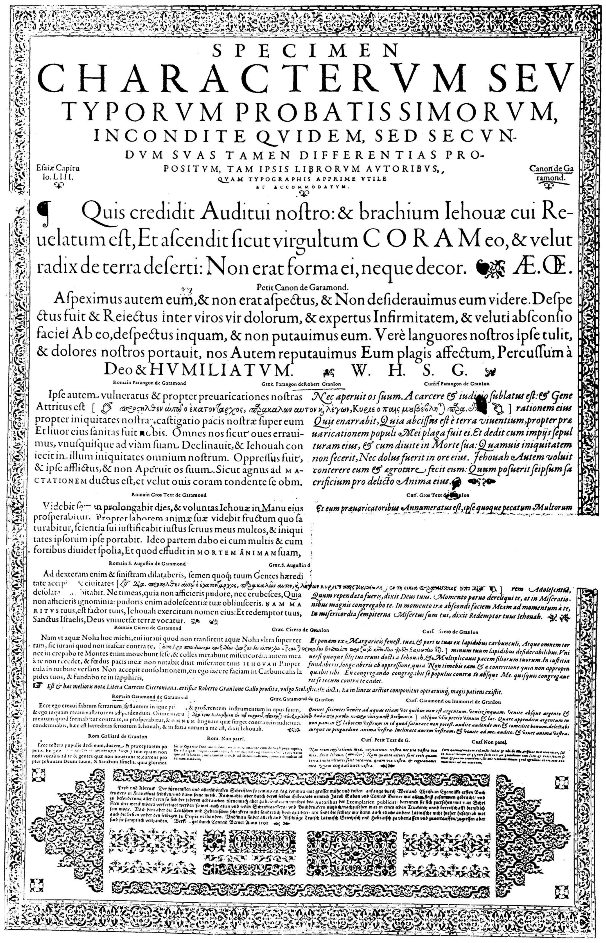
which asserts that complex systems exhibit emergent characteristics that cannot be predicted from knowledge of their components—simply put, that the whole is greater than the sum of its parts—and "vitalism", a relative of holism, which says that a certain kind of complex system possesses a vital, living essence that does not exist in its constituent parts.

Typography is a complex system which comprises type faces, which in turn comprise alphabets, which in turn comprise letters, all of which may be selected, combined, and arranged according to many different principles in a vast number of ways. The perceptual effect of a typographic work cannot, according to a holistic view, be deduced from simple knowledge of the individual letterforms.

What Warde and Tschichold both are trying to do in these apparently contradictory essays is to define and understand the aesthetic principles of typography. They both observe that most readers do not notice good typography. Nevertheless, because printed text is a dominant visual experience of modern civilization, those readers will spend hours a day and years in a lifetime viewing printed pages.

Therefore the principles that govern the typographic image are potentially important to everyone who works with printed texts—writers, editors, publishers, teachers, librarians, and bibliophiles—as well as to the printing historians and typographic designers who scrutinize typeset pages with professional eyes.

In this essay we examine two fundamental typographic principles, size and combination, and show that out of their interaction emerge three qualitative levels of the typographic image: form, pattern, and texture.



SPECIMEN
CHARACTERVM SEV
TYPORVM PROBATISSIMORVM,
INCONDITE QVIDEM, SED SECVN-
DVM SVAS TAMEN DIFFERENTIAS PRO-
POSITVM, TAM IPSIS LIBRORVM AVTORIBVS,
QVAM TYPOGRAPHIS APPRIME VTILE
ET ACCOMMODATVM.

Eſaiæ Capitu lo. LIII.

Canon de Garamond.

¶ Quis credidit Auditui noſtro: & brachium Iehouæ cui Reuelatum eſt, Et aſcendit ſicut virgultum C O R A M eo, & velut radix de terra deſerti: Non erat forma ei, neque decor. Æ. Œ.

Petit Canon de Garamond.

Aſpeximus autem eum, & non erat aſpectus, & Non deſiderauimus eum videre. Deſpectus fuit & Reiectus inter viros vir dolorum, & expertus Infirmitatem, & veluti abſconſio faciei Ab eo, deſpectus inquam, & non putauimus eum. Verè languores noſtros ipſe tulit, & dolores noſtros portauit, nos Autem reputauimus Eum plagis affectum, Percuſſum à Deo & HVMILIATVM. W. H. S. G.

Romain Parangon de Garamond — Græc. Parangon de Robert Granlon — Curſiff Parangon de Granlon

Ipſe autem vulneratus & propter preuaricationes noſtras Attritus eſt [[illegible]] propter iniquitates noſtras, caſtigatio pacis noſtræ ſuper eum Et liuor eius ſanitas fuit nobis. Omnes nos ſicut oues errauimus, vnuſquiſque ad viam ſuam Declinauit, & Iehouah conieccit in illum iniquitates omnium noſtrum. Oppreſſus fuit, & ipſe afflictus, & non Aperuit os ſuum. Sicut agnus ad MACTATIONEM ductus eſt, et velut ouis coram tondente ſe obm.

Nec aperuit os ſuum. A carcere & iudicio ſublatus eſt: & Generationem eius Quis enarrabit, Quia abciſſus eſt è terra viuentium, propter prævaricationem populi Mei plaga fuit ei. Et dedit cum impijs ſepulturam eius, & cum diuite in Morte ſua: Quamuis iniquitatem non fecerit, Nec dolus fuerit in ore eius. Jehouah Autem voluit conterere eum & ægrotare fecit eum: Quum poſuerit ſeipſum ſacrificium pro delicto Anima eius.

Romain Gros Text de Garamond — Curſ. Gros Text de Granlon

Videbit ſemen prolongabit dies, & voluntas Iehouæ in Manu eius proſperabitur. Propter laborem animæ ſuæ videbit fructum quo ſaturabitur, ſcientia ſui iuſtificabit iuſtus ſeruus meus multos, & iniquitates ipſorum ipſe portabit. Ideo partem dabo ei cum multis & cum fortibus diuidet ſpolia, Et quod effudit in MORTEM ANIMAM ſuam,

Et eum præuaricatoribus Annumeratus eſt, ipſe quoque pecatum Multorum

Romain S. Auguſtin de Garamond — Græc. S. Auguſtin d

Ad dexteram enim & ſiniſtram dilataberis, ſemen quoq; tuum Gentes hæreditate accip[illegible] & ciuitates [[illegible]] deſolat[illegible] habitabit. Ne timeas, quia non afficieris pudore, nec erubeſces, Quia non afficieris ignominia: pudoris enim adoleſcentiæ tuæ obliuiſceris. NAM MARITVS tuus, eſt factor tuus, Jehouah exercitum nomen eius: Et redemptor tuus, Sanctus Iſraelis, Deus vniuerſæ terræ vocatur.

rem Adoleſcentiæ, Quum rependata fueris, dixit Deus tuus. Momento paruo dereliqui te, at in Miſerationibus magnis congregabo te. In momento iræ abſcondi faciem Meam ad momentum à te, In miſericordia ſempiterna Miſertus ſum tui, dixit Redemptor tuus Iehouah.

Romain Cicero de Garamond — Græc. Cicero de Granlon — Curſ. Cicero de Granlon

Nam vt aquæ Noha hoc michi, cui iuraui quod non tranſirent aquæ Noha vltra ſuper terram, ſic iuraui quod non iraſcar contra te, [[illegible]] nec increpabo te Montes enim mouebunt ſeſe, & colles metabunt miſericordia autem mea à te non recedet, & fœdus pacis meæ non nutabit dixit miſerator tuus IEHOVAH Pauper cula in turbine verſans Non accepit conſolationem, en ego iacere faciam in Carbunculis lapides tuos, & fundabo te in ſapphiris,

Et ponam ex Margaricis feneſt. tuas, & portas tuas ex lapidibus carbunculi, Atque omnem terminum tuum lapidibus deſiderabilibus. Vniuerſi quoque filij tui erunt docti à Iehouah, & Multiplicauit pacem filiorum tuorum. In iuſtitia fundaberis, longe aberis ab oppreſſione, quia Non timebis eam. & a contritione quia non appropinquabit tibi. En congregando congregabit ſe populus contra te abſque Me. quiſquis congregauerit ſe tecum contra te cadet.

Eſt & hæc meliorű nota Litera Currens Ciceroniana, artifice Roberto Granlone Gallo prodita, vulgo Scolaſticalis dicta, Ea in lineas arctior componitur operarumq; magis patiens exiſtit.

Romain Garamond de Garamond — Græc. Garamond de Granlon — Curſ. Garamond ou Immortel de Granlon

Ecce ego creaui fabrum ferrarium ſufflantem in igne pr[illegible], & proferentem inſtrumentum in opus ſuum, & ego inquam creaui vaſtatorem ad perdendum. Omne inſtru[[illegible]] mentum quod formabitur contra te, in proſperabitur, & OMNEM linguam quæ ſurget contra te in iudicium condemnabis, hæc eſt hæreditas ſeruorum Iehouah, & iuſtitia vorum a me eſt, dixit Iehouah.

Omnes ſitientes venite ad aquas etiam vos quibus non eſt argentum. Venite inquam. Venite abſque argento & [illegible] abſque vllo pretio vinum & lac. Quare appenditis argentum in non panem & laborem veſtrum in id quod ſaturare non poteſt. audite audiende me. & comedite bonum. delectabiturque in pinguedine anima veſtra. Inclinate aurem veſtram. & venite ad me. audite. & viuet anima veſtra.

Rom. Galliard de Granlon — Rom. Non parel. — Curſ. Petit Text de G. — Curſ. Non parel.

Ecce teſtem populis dedi eum, ducem & præceptorem populis. En gen[illegible] tem quam non noſti vocabis ad te & gentes quæ non nouerunt te, current propter Iehouam Deum tuum, & Sanctum Iſraelis. quia glorifica

[illegible]

Non enim cogitationes meæ. cogitationes veſtræ. nec via veſtra via mea. dixit Iehoua. [illegible] Nam quanto altiores ſunt cœli quam terra. tanto altiores ſunt viæ meæ. quam via veſtra. & cogitationes meæ Quam cogitationes veſtræ.

[illegible]

Prob vnd Abtruck Der fürnemiſten vnd allerſchönſten Schrifften ſo jemals an tag kommen mit groſſer mühe vnd koſten anfangs durch Weiland Chriſtian Egenolffs erſten Buchtruckers zu Franckfurt ſelbſten/vnd dann ſeine witib. Nachmahls aber durch deren ſelben Erben/als nemlich Jacob Sabon vnd Conrad Berner mit allem fleiß zuſammen gebracht/ vnd zu beförderung aller deren ſo ſich der ſelbigen gebrauchen. fürnemlich aber zu beſonderem vortheil den Autoribus der Exemplarien publiciert/ darinnen ſie ſich zuerſehen/mit was Schrifften ihre werck mögen verfertiget werden/ ſo wol auch allen vnd jeden Schrifftgieſſern/ vnd Buchdruckern nützlich/nachzuſehen was in eines jeden Truckerey vnnd bereitſchafft dienſtlich ſein möge. Nach dem aber die Teutſchen vnd Hebraiſchen Schrifften nicht ſonderlich hoch geachtet/ als ſindt die ſelbige wie dann auch etliche andere Latiniſche nicht hieher geſetzt/ob wol auch die beſten vnder den ſelbigen In Copia verhanden. Vnd man findet alſerh and Abſchläge Teutſch Latiniſch Griechiſch vnd Hebraiſch zu vberlaſſen vnd zuuerkauffen/zugieſſen aber ſind ſie ſampftlich verhanden. Verfe[illegible]get durch Conrad Berner Anno 1592

Figure 1: The Konrad Berner type specimen, Frankfurt, 1592. See note (3).

2 Size

Variation of size within a text is one of typography's signal contributions to the art of literacy. Size variation was not unknown before typography—it can be found in Latin and Greek alphabetic manuscripts, Egyptian hieroglyphic inscriptions, and Chinese logographic brush writing and block printing, but it has developed in typography to a greater degree than in those chirographic, epigraphic, or xylographic traditions.

Multiple sizes of type in a text are not strictly necessary, either for sense—typewritten texts composed in a single size of type have served authors, editors, teachers and students well for over a century—or for beauty—certain incunabula composed only in one size of type are still regarded as paragons of typography. Yet, variation of type size makes text clearer, more dynamic, and more engaging. Size variation is a mainstay of typographic design. Without it, the modern book would be dull, and the modern newspaper impossible.

Size is so important to typography that the creation of a spectrum of different sizes of type occupied the careers of the greatest punchcutters of the golden age of typography. After Claude Garamond cut his first definitive romans, like the St. Augustin size (approximately 14 point) in Jacques DuBois' *In Linguam Gallicam Isagoge* printed by Robert Estienne in 1531, three generations of punchcutters labored to create additional sizes in Garamond's idiom. The Konrad Berner Foundry type specimen of 1592 (fig. 1) shows the Garamond style of roman face available in a series of sizes from Canon (approximately 48 point) to Nonpareil (approximately 6 point), cut variously by Garamond himself, Robert Granjon, Pierre Haultin, and Jacques Sabon. Christopher Plantin's folio type specimen of 1585 shows a similar range by most of the same hands, with yet larger sizes of roman by Hendrik van den Keere (though in a style noticeably different from Garamond's). Toward the end of the sixteenth century, Guillaume Le Bé I and Jacques de Sanlecque also cut sizes of roman in the Garamond style. Moreover, Granjon in his long and prolific career cut many sizes of italic faces in his own distinctive styles, examples of which are also shown in the Berner and Plantin specimens. (3)

Why is size so useful? Tschichold's comparison of typography to music suggests analogies that may illuminate the role of size in typography. Large letters are used for emphasis in text, as loud notes are used for emphasis in music or loud voices in conversation. Hence, size in type is somewhat like loudness or dynamics in music. Just as different parts of a musical composition are loud or soft (in musician's terms, forte or piano) different parts of a text are large or small.

Type size is more closely analogous to musical pitch, but in a spatial rather than a temporal dimension. In typography, spatial frequency is the number of black and white alternations per unit of distance, just as musical frequency is the number of acoustic vibrations per unit of time. In a given length of line, a small size of type fits more letters, and hence more alternations of black stems with white counters and spaces, than does a large size; hence, smaller type has a higher spatial frequency. Here we see chains of "minimumu" at different sizes:

minimumuminim

minimumuminimumumi

minimumuminimumuminimumuminimimuminim

A typeface that is available in a range of sizes is like a musical instrument which can produce a scale of notes of different pitch. A typographic composition that includes different type sizes thus comprises "notes" of different visual pitch, just as a musical composition comprises notes of different audible pitch. When a reader views a whole page, different type sizes are perceived simultaneously, thus constituting a kind of spatial harmony; when one reads a linear text, the types are perceived sequentially, thus constituting a kind of spatial melody.

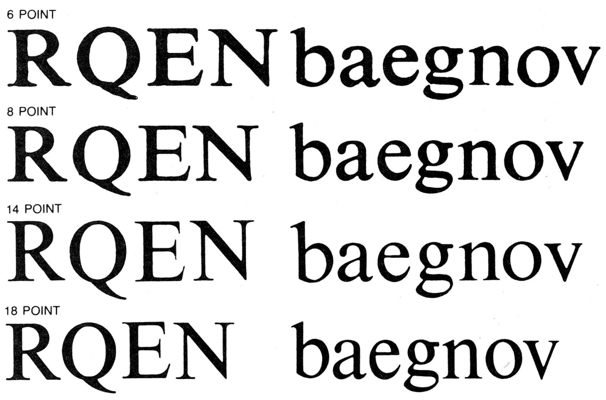


Figure 2: Times Roman fonts of different design sizes, scaled to the same x-height to show proportional changes according to scale. Courtesy of Kris Holmes.

An analogy can also be drawn between the visual qualities of the typographic notes produced by a typeface, and musical "timbre". Typographic timbre is a complex visual sensation resulting from the interaction of the proportions, details, and spacing of the typeforms, just as musical timbre is a complex acoustic sensation that results from the sum of the harmonics (partials or overtones) produced by the shape and design of an instrument. A typeface family that includes related variations, such as roman, italic, and bold (and perhaps sans-serif as well) permits the typographer to adjust visual timbre somewhat independently of spatial frequency or size. (4)

A concept related to size is "scale". A scale is a fixed series of measures, and hence to scale type is to enlarge or reduce it. When scaling leaves letter proportions unchanged, it is termed "linear". Linear scaling is standard in photographic and digital typography. Opposed to linear scale is non-linear or optical scale, adapted to the eye rather than the machine. Harry Carter, in his article "The Optical Scale in Typefounding", demonstrates that the traditional proportions of typeforms differ according to whether type is large, medium, or small (fig. 2). In particular, he shows that type designed for a small scale tends to be wider, have a larger x-height, thicker hairlines, and more exaggerated serifs, joins, and terminals than type for a large scale. (5)

Similarly, Daniel Berkeley Updike observes that, "A design for a type alphabet that may be entirely successful for the size for which it is drawn, cannot be successfully applied to all other sizes of the same series. Each size is a law unto itself…”. (6)

Traditional punchcutters and scribes made such proportional changes in order to optimize legibility. Recent research in visual perception suggests that such proportional changes are necessary because the human visual system has non-linear sensitivity to visual features of different spatial frequencies (fig. 3). (7)

Today, photographic and computer techniques can render almost any size of type with ease and precision. Type size has become a continuum instead of a sparse series of fixed sizes and proportions. Because innumerable fine gradations of size are now possible, the typographer must strive to understand the principles that govern the appearance of types at all sizes.

The range of sizes in text is commonly divided into three main scales: large, medium, and small. (8) These refer to apparent size more than physical size. Apparently small type may be physically small, e.g., 6 point type in a newspaper classified advertisement read at a distance of twelve inches, or physically large, e.g., two-foot high type on a billboard read at a distance of 288 feet, but both appear to be the same size because they subtend the same degree of visual angle at the retina of the eye, namely 0.4 degrees of visual angle in this case.

In general, large scale type is used for the display of particular words or phrases, as in titles; medium scale for the main or body text, and small scale for reference text, like footnotes in books or classified advertising in newspapers. However, the actual sizes depend on context. In a book, the medium scale text may be composed in 12 point and small scale footnotes in 9 point, whereas in a newspaper, the medium scale text may be in 9 point and the small scale classified advertising in 6 point. (8)

The textual significance of a given size of type is based on relative scale—the relationship of a given size of type to the other sizes of type on the page. For example, a small amount of large type positioned above a large amount of smaller type usually marks the former as a title or heading for the latter, whatever their actual sizes may be. Relative scale has meaning, whereas absolute size is merely a physical fact.

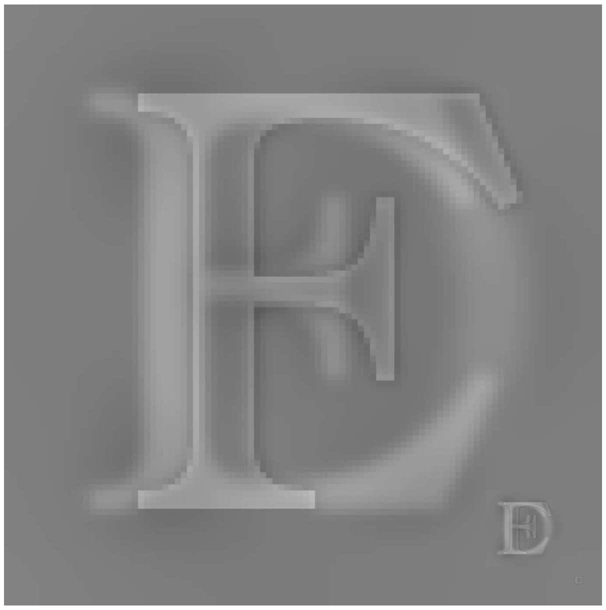


Figure 3: Superimposed letters F E D have been individually filtered into separate spatial frequency bands to which the human visual system is differentially sensitive. The large F is rendered within a band of high frequencies, mainly edges, that make the letter identifiable at a close viewing distance around one to two feet (30 to 60 cm). The large E is within a band of middle frequencies, mainly interior form, that make it more identifiable at a distance around three to four feet (90 to 120 cm). The large D lies within a band of low frequencies, mainly basic shape, that make it most visible at around eight to twelve feet (240 to 360 cm). *TUGboat* readers may experiment to find optimal visibilities by varying viewing distances. Identifiability of the letters may also depend on viewer eyesight and the ambient illumination.

At lower right, the superimposed small letters—one eighth the size of the large ones—are rendered within the same frequency bands as the large ones, respectively, but because the spatial frequencies reaching the eye are increased by smaller size or greater distances, followed by filtering out of high frequencies in the visual system, *TUGboat* readers may find the E or D more visible at normal reading distances, and the D most visible at greater distances. It would be intriguing to read a text composed in a font like this, if one is possible, which reveals different texts depending on reading distances. The authors of this study state, “Thus, large letters (and coarse squarewaves) are identified by their edges; small letters (and fine squarewaves) are identified by their gross strokes.”

See note (7) with reference to Majaj et al. for further explanation. Image courtesy of Denis Pelli.

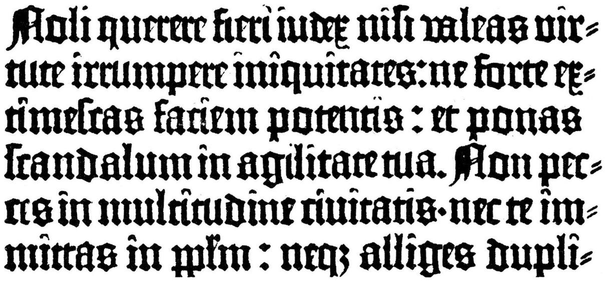

Noli querere fieri iudex nisi valeas vir⸗
tute irrūpere iniquitates: ne forte ex⸗
timescas faciem potentis: et ponas
scandalum in agilitate tua. Non pec⸗
ces in multitudine civitatis· nec te im⸗
mittas in plēm: neq; alliges dupli⸗

Figure 4: Text composed in type based on the Textura script, so named because of its woven appearance. From the Gutenberg Bible, ca. 1455, the first European book printed from movable type. Courtesy of the RIT Cary Graphic Arts Collection.

3 Combination

In text composition, size is always associated with another fundamental typographic principle, combination, as a direct consequence of the nature of language.

As a medium of communication, typography is twice removed from its content. At the first remove, writing is a visual representation of language, and at the second, typography is an industrialized representation of writing. To the reader, the letters that compose a text are recognized in passing but are not themselves objects of contemplation; rather, the words, and behind them, the ideas expressed by the text are of primary interest. As if to acknowledge these two aspects, the word "text" has dual meanings. First it is the printed artifact—the perceptual object, and second it is the linguistic construction—the conceptual object. Although text is used mainly in the latter sense today, its etymology suggests the former, as the modern word is derived from Latin *textus*, a weaving, referring to the woven pattern created by written letters arrayed on a page. (9) (See fig. 4.)

Because the art of weaving involves the creation of a two-dimensional fabric from a one-dimensional thread, an obvious analogy with typography can be drawn, for speech is a one-dimensional string in time woven by typography into a two-dimensional plane in space. As the revolutionary Russian typographer El Lissitzky observed, "We have two dimensions for the word. As a sound it is a function of time, and as a representation it is a function of space." (10)

Typographic weaving is composition, the repetition and recombination of a small number of letterforms into strings and the assembly of those strings into masses of text. It reflects what the French linguist André Martinet has called the "double articulation" of language. ("Articulation" here being itself a *double entendre*, meaning both segmentation into components and pronunciation.) Though apparently of infinite variety, the utterances of a language are constructed from a finite set of meaningful segments—words or "morphemes"—which constitute the first articulation. The words themselves are constructed from a much smaller set of sound units—"phonemes"—which constitute the second articulation. A language may contain myriads of words but will have fewer than a hundred distinct sounds. English, for example, has some forty-five phonemes, and at least several hundreds of thousands of words. Since there are so few individual phonemes in relation to the large number of words, each phoneme is repeated many times in many combinations. (11)

To represent language in a graphic medium, typography likewise utilizes the repetition and recombination of elements. In alphabetic typography, the graphic signs or "graphemes" are letters. In logographic typography (used for Chinese, Japanese, and Korean in varying degrees) the graphemes are characters. Individual letters signify sounds or phonemes (the second articulation), and combinations of letters (or single characters in a logographic script) signify words or morphemes, the first articulation. As is true for the phoneme, the single letter generally has no meaning by itself; its significance lies in its differentiation from the other letters, and its combinations with them to produce higher-level meaningful segments.

4 The typographic image

The interaction of size and combination creates three levels of the typographic image: form, pattern, and texture. Each level contains a range of sizes in varying degrees of combinatory complexity. Although size is a quantitative aspect of type, the emergent levels of the typographic image are qualitative.

Form

The design of a letter is a study in form (fig. 5). The letterform is a dualistic rendering: black and white, intaglio and relief, figure and ground, on and

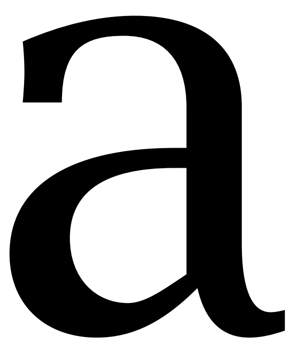


Figure 5: The ‘a’ of Lucida Book, font size 216pt.

off. The contour that separates and defines the polarities of the letter image creates the interaction between letterform and counterform, interior and exterior, positive and negative space. The resultant perception includes lines, such as fair curves, straight edges, smooth joins, and sharp corners, as well as shapes, such as solid regions, empty hollows, delicate taperings, and abrupt terminations.

At the level of form, the letter is viewed at a large scale. In the large letter, interior area dominates contour line because of a geometrical relationship known to the Greeks: the area of a form increases in proportion to the square of the size, whereas the length of the contour that defines the form increases in direct proportion to the size. When a letter is large, the area of the interior is large in proportion to the line of the contour, and much of that region is relatively far from the contour. Hence, the mass (or void) tends to dominate the image. However, this tendency is partly counteracted by mechanisms in the human visual system, such as lateral inhibition, that extract edges from images and de-emphasize monotonous surfaces. (12)

At its largest perceptual size, the letterform is isolated. Extracted from the context of the alphabetic system, the isolated letter becomes an object of contemplation, not meaning. It is pure form, its semiological role vacated because alone the letter has no significance. It is an abstraction. As Eric Gill wrote, “Letters are not pictures or representations. They are more or less abstract forms.” (13)

Form invites abstract analysis. Renaissance humanists and artists analyzed the shapes of letterforms with the compass and straightedge of Euclidean geometry (fig. 6). Enlightenment academicians used the grids of Cartesian geometry. Today’s computer scientists use the mathematical formulae of splines and conic curves.

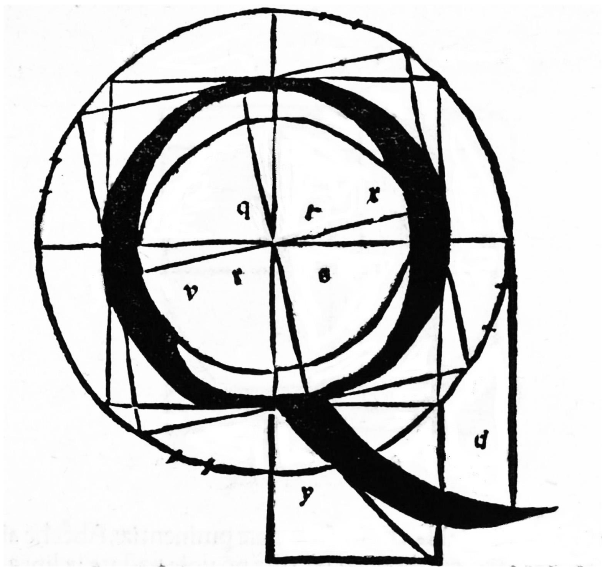


Figure 6: Capital Q letter construction by Sigismondo Fanti, *Theorica et practica … de modo scribendi fabricandique omnes literarum species*, Venice, 1514.

Because letterforms can easily be scaled to any size by modern typographic technology, the finest details of the forms, formerly examined only by experts, can now be appreciated by everyone. The letter under the lens of photographic and digital typography is like the work of art under the lens of the camera, as discussed by Andre Malraux in *The Voices of Silence* (1978):

> In an album or art book the illustrations tend to be of much the same size. Thus works of art lose their relative proportions; a miniature bulks as large as a full-size picture, a tapestry or a stained-glass window… In this way reproduction frees a style from the limitations which made it appear to be a minor art.
>
> Indeed, reproduction (like the art of fiction, which subdues reality to the imagination) has created what might be called “fictitious” arts, by systematically falsifying the scale of objects; by presenting oriental seals the same size as the decorative reliefs on pillars and amulets like statues… Sometimes the reproductions of minor works suggest to us great styles which have passed away—or which “might have been”. (14)

At a large scale, the letterform ceases to be a minor art, and takes its place with the forms of painting, sculpture, and even architecture as objects of study and contemplation. The letter as form is displayed, as Beatrice Warde said above, "as an expression of beauty for its own sake and for the delectation of the senses".

The effect that the forms of letters will produce when combined into text is often impossible to predict because many of the characteristics of an alphabet design emerge only *en masse*. The next level of the typographic image, the level of forms in combination, is the level of pattern.

Pattern

Combined into words and lines of text, the forms and counterforms of individual letters become elements of a periodic structure. The shapes of stems, bowls, serifs, and other features of a letter relate to similar features of other letters, as the forms of counters inside a letter relate to the counters of other letters and to the spaces between letters. Because there are only a few different letterforms in an alphabet, each form is repeated many times in many combinations, and the relationships between forms and counterforms extend beyond near neighbors to entire lines and columns. The sum of these relationships is a pattern.

The level of pattern occurs at a medium scale, where the interior area of a form no longer dominates the contour line. Instead, these two different aspects of a geometric figure tend toward equilibrium. Because much of the interior area of a form lies close to the contour, the interaction of contours with areas, edges with surfaces, gives the text image an active quality.

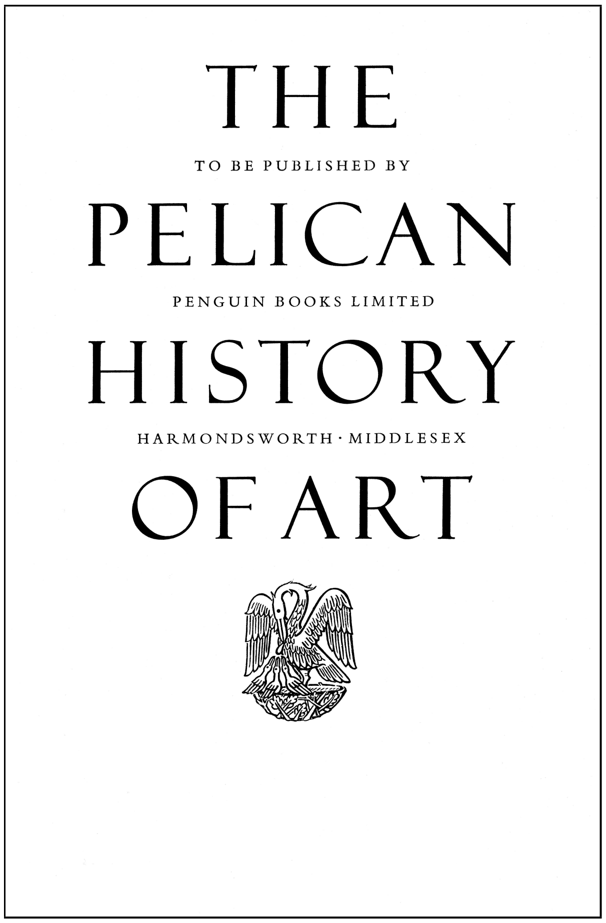
A text pattern is complex because it results not only from the repetition of whole letters, but also of letter parts. Letters are constructed from more primitive graphic elements. In handwriting, these elements are kinesthetic movements—gestures and strokes—which leave graphic traces; in type design, they are graphic features which can be consistently combined, such as stems, bowls, diagonals, cross bars, hairlines, joins, serifs, and terminals. Because the letters share a small number of elemental features, the structure of the text pattern is derived from symmetries and transformations of repeating letter parts as well as from repetitions of fully formed letters.

This constructive, systematic nature of typefaces is a result of the formal interactions of the letters during a long, common history. The development of the alphabet shows a transformation of originally iconic or pictorial signs, linked by relationships of resemblance to the objects signified, into abstract, symbolic shapes which have stronger formal linkages to each other than to the things they signify. The alphabet represents a system of sounds, not a collection of isolated entities, and thus no letter exists in isolation. Graphically, each letter must be unique in order to carry its particular significance, but it must also be fashionable by the same means as the others, writable by the same hands and tools, constructible from the same elements.

Pattern begins with the word and comes to full flower in the line and column. The line of type is a one-dimensional pattern, like a frieze, based on the repetition of geometric figures in a line. For efficient packing, the continuous line of type is cut into segments which are arrayed in columns, creating a two-dimensional type page which, like wallpapers, tilings, and fabrics, is based on repetition in a plane (fig. 7). This structural relationship of type to pattern has been familiar to typographers for centuries. Robert Granjon in the sixteenth century and Pierre Simon Fournier in the eighteenth excelled in the creation of floral forms or fleurons that could be combined into ornamental patterns to accompany their typeforms combined into text. (15)

The patterns that emerge from text are nevertheless different from those of ornamental friezes and tilings because the latter, though often beautiful and intricate, convey relatively little information despite multiplicity of forms. The mathematical principles that govern the tiling of the plane make it possible to predict exactly when and where a given geometric element will occur in a pattern (fig. 8). Hence, there is little new information derived from each repetition of the pattern. (16)

A third dimension of pattern is found in the codex form of book, which, like a crystal, is composed of parallel text planes in space. The codex book is, however, a cognitive more than a visual structure, since it relies on the reader's memory of the patterns on successive pages rather than on simultaneous perception of them. The third dimension of book structure is often emphasized in modern books created as art objects or experiments.


THE

TO BE PUBLISHED BY

PELICAN

PENGUIN BOOKS LIMITED

HISTORY

HARMONDSWORTH · MIDDLESEX

OF ART

Figure 7: Page of prospectus designed by Jan Tschichold, 1947. The individual forms of the large capitals are evident, and the beginnings of patterns are seen in the word combinations. The lines of smaller capitals exhibit stronger patterns, while individual forms are less evident.

So-called "hyper-texts", used for reading and accessing complex computer data bases, have structures that rely on the computer's memory to keep track of page sequence, which can be arbitrarily complex and convoluted, and seldom follows the regular, linear sequencing of the codex. Yet, so far, the basic designs of the typographic pages of hypertexts remain repetitive and book-like.

Typographic tilings are only partially predictable. The periodicity of a typographic pattern is approximate rather than exact, flexible rather than rigid, surprising rather than predictable, because the text constantly changes. The occurrence of a given letter or word space in a particular position is determined not by rules of geometry but by rules of language and the idiosyncratic choices of an author. (17)

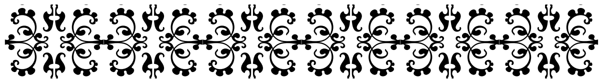


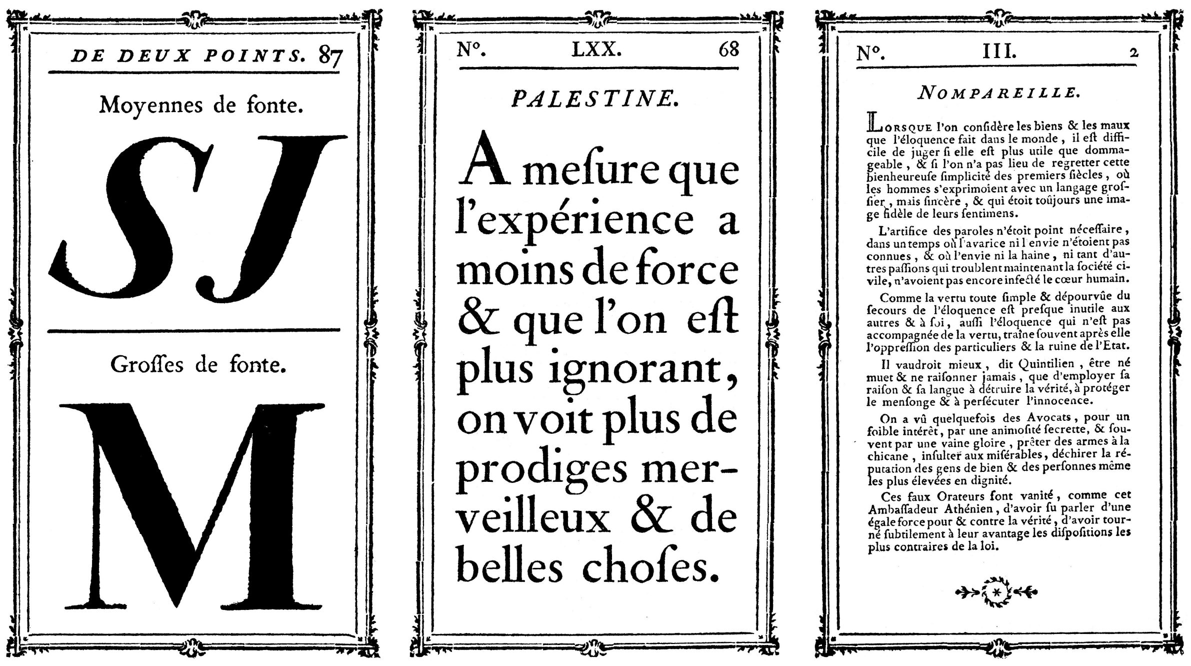
Figure 8: Arabesque after Granjon, from *Kleines Spiel mit Ornamenten* by Max Caflisch (Berne: Angelus-Drucke, 1965), reconstructed by Jacques André (*Petits jeux avec des ornenents*, `jacques-andre.fr/ed/caflisch-jeux.pdf`, 2009, p. 63).

The pattern made by a typeface is greatly influenced by the amount and distribution of space between lines and between letters. The typical text column makes a striped pattern which the typographer can augment or diminish by using more or less leading—interline spacing. Within the line, visually even letterspacing is often held to be ideal. A regular spatial frequency of alternating dark stems and white spaces creates a smooth rhythm with a look of stability and repose (as in the earlier "minimum" example). Techniques for the regular spacing of capitals have been described by Tschichold and for the even fitting of lower case by Walter Tracy. (18)

Perfectly even spacing is difficult to achieve in practice because the arbitrary shapes of letters inevitably cause some degree of irregularity of fitting in standard typographic technology. Not all letter combinations seem equally spaced. But some degree of irregularity in letterforms and spacing may be preferable to monotonous regularity, just as in music, where slight inharmonicity of partials, or overtones, creates the complex, wavering quality that makes a piano tone "warm", whereas precisely harmonic partials produce bland sounds. (19)

In contemporary advertising typography, the unstable, restless patterns of tight letterspacing are preferred. The busy, frenetic effect of so-called "sexy" spacing, in which the letters tend to rub up against each other, are common in the typography of mass market persuasion, where arresting, staccato patterns draw the reader's attention to texts that might otherwise be ignored. (19A)

Tight packing of letters also magnifies the logographic aspect of typography. Reduction of space between letters within a word emphasizes by contrast the space between words, thus articulating the text into an archipelago of word islands, rather than continuous strings of letters. The semi-crystalline lattice of letters on the page is thus interrupted by holes, but holes that have a purpose and meaning. In typography, the word space has developed as



DE DEUX POINTS. 87

Moyennes de fonte.

SJ

Groſſes de fonte.

M

N°. LXX. 68

PALESTINE.

A meſure que l'expérience a moins de force & que l'on eſt plus ignorant, on voit plus de prodiges mer-veilleux & de belles choſes.

N°. III. 2

NOMPAREILLE.

LORSQUE l'on conſidère les biens & les maux que l'éloquence fait dans le monde, il eſt diffi-cile de juger ſi elle eſt plus utile que domma-geable, & ſi l'on n'a pas lieu de regretter cette bienheureuſe ſimplicité des premiers ſiècles, où les hommes s'exprimoient avec un langage groſ-ſier, mais ſincère, & qui étoit toûjours une ima-ge fidèle de leurs ſentimens.

L'artifice des paroles n'étoit point néceſſaire, dans un temps où l'avarice ni l'envie n'étoient pas connues, & où l'envie ni la haine, ni tant d'au-tres paſſions qui troublent maintenant la ſociété ci-vile, n'avoient pas encore infecté le cœur humain.

Comme la vertu toute ſimple & dépourvûe du ſecours de l'éloquence eſt preſque inutile aux autres & à ſoi, auſſi l'éloquence qui n'eſt pas accompagnée de la vertu, traîne ſouvent après elle l'oppreſſion des particuliers & la ruine de l'Etat.

Il vaudroit mieux, dit Quintilien, être né muet & ne raiſonner jamais, que d'employer ſa raiſon & ſa langue à détruire la vérité, à protéger le menſonge & à perſécuter l'innocence.

On a vû quelquefois des Avocats, pour un foible intérêt, par une animoſité ſecrette, & ſou-vent par une vaine gloire, prêter des armes à la chicane, inſulter aux miſérables, déchirer la ré-putation des gens de bien & des perſonnes même les plus élevées en dignité.

Ces faux Orateurs ſont vanité, comme cet Ambaſſadeur Athénien, d'avoir ſu parler d'une égale force pour & contre la vérité, d'avoir tour-né ſubtilement à leur avantage les diſpoſitions les plus contraires de la loi.

Figure 9: Three pages from Fournier's *Manuel Typographique*, from left to right, exemplars of form, pattern, and texture. Scaled to approximately 80% of original size.

Although Fournier invented a precursor of our current point system, his *Manuel* uses names for type sizes, as was common in his era. In modern points, the sizes are approximately: Grosses de fonte ('M' in first image) = 96 points high; Palestine (second image) = 24 point body size; Nompareille (third image) = 6 point body size.

a representation of a psychological rather than an acoustic reality, since there are rarely gaps or pauses between words in continuous speech. (20) The importance of the blank space as a logographic mark in English text is indicated by its frequency, greater than e, the most frequent letter.

Further, close letterspacing creates characteristic word images by emphasizing the irregularity of a given sequence of features in a word, thus distinguishing its shape from that of other words. By subdividing the letter pattern into characteristic chunks and gaps instead of a continuous flow, alphabetic typography takes on some of the qualities of logographic Chinese writing.

Texture

The realm of texture is the habitat par excellence of the serious reader, where the text reaches its greatest mass and density and the ultimate visual qualities of the literate image emerge. At the level of texture, line dominates area; forms are obscured; patterns become aggregates. Small, the letters are seen as though at a distance, through an intervening atmosphere, resembling more the attenuated figures sculpted by Alberto Giacometti than the forms and volumes shaped by Henry Moore.

When letters are seen at a small size, it is difficult if not impossible to discern the exact forms of fine features such as serifs, joins, and terminals, though these are obvious at a large size. Linearity replaces interiority. The area of the interior of the small letter is small compared to the length of the contour, and most of the interior area lies along the contour. At the level of texture, the letter is mainly line, an aspect intensified by the edge-detection mechanisms of the human visual system. Textures are complexes of edges.

Patterns evident at a medium scale become so dense at the small scale that statistical qualities emerge; the density or "color" of the text, its granularity, and its weave can be seen directly. Out of the myriad interactions of features and spaces, texture emerges. That a basic quality of the text emerges from multiplicity was known both to traditional and to modern typefounders. Pierre Simon Fournier observes in his *Manuel Typographique* (fig. 9):

One letter measured singly may seem neither

Typography is closely allied to the fine arts, and types have always reflected the taste or feeling of their time. The charm of the early Italian types has perhaps never been equalled

Typography is closely allied to the fine arts, and types have always reflected the taste or feeling of their time. The charm of the early Italian types has perhaps never been equalled

Figure 10: A quote from D. B. Updike's *Printing Types*, set in Helvetica (above) and Syntax (below), illustrating an observation from poet Heinz Peyer.

> appreciably too big nor too small, but ten thousand composed into printed matter repeat the error ten thousand times over, and, be this never so small, the effect will be the opposite of what was intended. The same trouble also occurs when a stroke is made either too thick or too thin relative to its length, which makes a letter look clumsy and faulty, with out the reason for it being always easy to find out. (21)

One memorable observation on typographic texture was made by Heinz Peyer, a Swiss poet, who said that reading a text composed in Helvetica was like walking through a field of stones, whereas reading a text in Syntax was like walking through a field of flowers (fig. 10). (23)

Chauncey Griffith, designer of twentieth-century news faces, found the ready analogy between textiles and text typography:

> But the individual piece of type is like a thread. A single thread might be dyed crimson, scarlet, or pink and the human eye would find the difference hard or impossible to detect. But once that thread is woven into cloth, the color is very apparent. So type must be judged after it is woven into the texture of a paragraph or a page. (22)

Although the level of texture is where most reading takes place, the vocabulary of texture is the least developed of the three levels of the typographic image. At the level of form, terms like line, space, and mass, commonly applied to drawing, painting, and sculpture, can equally apply to type. At the level of pattern, notions of symmetry, homology, and periodicity, commonly applied to tessellations and mosaics, can also apply to type. But at the level of texture, few standard terms are available. Typographers use “color” for achromatic density—the darkness or lightness—of printed text, or a few ad hoc expressions like “spikey”, “wormy”, or “stolid”, depending on one's feeling for metaphor. In the realm of texture, poets may need to come to the aid of printers by providing words to describe the images of the “black art”.

Form is often susceptible to logical analysis, and pattern somewhat so, but texture evades precise description because its repetitions are so numerous, its features so small, and its interactions so refined, that the multifarious complexity of the emergent image resists orderly analysis. Texture requires a holistic more than an analytic understanding. This is an aspect of a deeper and larger philosophical difficulty stated by Pascal in his famous comparison of the intuitive mind to the geometrical mind:

> These (principles) can be seen only with difficulty, they are sensed more than seen, and it is infinitely difficult to make them known to those who do not sense them for themselves. These things are so delicate and so numerous that a sense of great delicacy and precision is necessary to perceive them and to judge correctly and accurately from the perception, and in most cases it is not possible to prove the judgment logically as in geometry, because not all the necessary principles are available and it would be an infinite undertaking to gather them. It is necessary to see the whole thing all at once, in a single glance, and not by progressive reasoning… (24)

As a consequence of the complexity and refinement of texture, psycho-physical and mathematical studies of its perception have used statistical analyses and formalized notions of clustering, orientation, and brightness. However, the visual elements and arrangements used in such perceptual studies are simple compared to the complexity of letterforms in actual text, and hence such studies, though suggestive, have so far been minimally relevant to typography. (25)

For various purposes, typefounders, telegraphers, cryptographers, and information theorists have made statistical measures of the frequencies of letters and letter combinations in various languages. (26)

When coupled with knowledge of the forms

Quo usque tandem abutere, Catilina, patientia nostra? quam diu etiam furor iste tuus nos eludet? quem ad finem sese effrenata iactabit audacia? Nihilne te nocturnum praesidium Palati, nihil urbis vigiliae, nihil timor populi, nihil

Typography is closely allied to the fine arts, and types have always reflected the taste or feeling of their time. The charm of the early Italian types has perhaps never been equalled; and the like is true of the Renaissance manuscripts

nugwagímx̣ łgá dáyaxbt, aga dánmax̣ wílxba díq̓əlpxix. łúxwan dáwax wakáyim, łúxwan agúnax̣ alátxwida, łá::niwa iłk̓álak̓iya łdímamt. gałə́kim, ‹‹adî::! kiníkštx̣ náyka. nagəlgát adáłutk. idmílxam á::nga wakádaču ikdúdina.››

Figure 11: Texts in Latin (from Cicero's first oration against Catiline), English (from D.B. Updike's *Printing Types*), and Clackamas Chinook (from Jacobs' and Howard's Clackamas-Chinook Texts) show different textures resulting from different letter frequencies. Composed in Syntax-Antiqua.

of the letters, such statistics can partially indicate texture. For example, Updike notes that differences in letter frequencies change the appearance of a type page. He favorably compares Latin text, with its frequent u's, m's, and n's, infrequent diagonally stroked y's, and infrequent descenders, to English text, with its greater frequency of diagonals and descenders. (27)

Updike's personal preference for the texture of Latin was, however, a matter of taste more than objective judgment. The letter frequencies of Latin and English actually seem rather similar when compared to those of non-Indo-European languages. Had Updike broadened his literary horizons beyond Europe and New England to native American texts published by his contemporary Franz Boas, he might have noticed, for example, that literary texts in the Chinookan languages of the Pacific Northwest have a plenitude of diagonals and descenders that the texture of English seems staid, and Latin dull in comparison. (28) (See fig. 11.)

When the typographic image is understood to comprise distinct levels of different aesthetic and functional qualities, the opposing arguments of Warde and Tschichold can both be seen to be true. At the level of form, typography is a fine art. Its works are accessible to the aesthetic sensibility of the viewer as well as to the intellectual analyses of the art historian, and its shapes are susceptible, at least to some degree, to the logical analyses of the mathematician and scientist. At the level of texture, typography is a utilitarian craft. Its forms are aesthetically transparent to the reader, and its emergent visual qualities, though obvious to the beholder as a holistic image, are resistant to articulate analysis, as its perceptual workings remain for the most part mysterious to the scientist. Typography as pattern articulates form with texture, presenting a bivalent image—formal yet functional, ornamental yet informational—leading on the one hand toward the isolated shape and on the other toward the emergent image. As a whole, then, typography can be seen both as a tool for thought and as an object of contemplation, a conveyor of sense and a delight to the senses.

5 Notes

(1) Beatrice Warde, "The Crystal Goblet, or Printing Should be Invisible", originally an address entitled "Printing Should be Invisible", given to the British Typographers' Guild at the St Bride Institute, London, 1932.

Reprinted in *The Crystal Goblet: Sixteen Essays on Typography*. London: Sylvan Press, 1955; Cleveland and New York: The World Publishing Co., 1956. Also reprinted in *The Monotype Recorder*, vol. 44, no. 1, Autumn 1970. `readings.design/PDF/The%20Crystal%20Goblet.pdf`

Warde's famous essay enjoys an enduring place among the best literature of typography, and her phrase, "until printing itself hands its usefulness to some yet unimagined successor" was clairvoyant. Nine decades later, her "unimagined successor" to print is not only imaginable, it is nearly ubiquitous in the digital display of text on the screens of computers, tablets, e-readers, smart phones, and the like. The number of smart phones worldwide is estimated to be greater than six billion and the number of computers at least 2 billion. Ebooks in various forms have been estimated to constitute 14 to 21 percent of books published per year.

(2) Jan Tschichold, "Ton in des Töpfers Hand", in *Ausgewählte Aufsätze uber Fragen der Gestalt des Buches und der Typographie*. Basel: Birkhäuser, 1975.

English translation by Hajo Hadler, as "Clay in a Potter's Hand", in *The Form of the Book*, Robert Bringhurst, ed. Vancouver, Canada: Hartley &

Marks Publishers, 1991. (Hadler's translation differs somewhat from the one in this essay.)

(3) John Dreyfus, ed., "Specimen no. 2: Konrad Berner, Frankfurt 1592", in *Type Specimen Facsimiles*. John Dreyfus, general editor, with research by A.F. Johnson, Harry Carter, Matthew Carter, Netty Hoeflake, Mike Parker. London: Bowes & Bowes, 1963.

H.D.L. Vervliet, Harry Carter, eds., "Specimen no. 17: Plantin's Folio Specimen c. 1585" and "Specimen no. 18: The Le Bé–Moretus Collection of Fragments c. 1599" in *Type Specimen Facsimiles II*. Toronto: University of Toronto Press, 1972.

Nicolas Barker, "The Aldine Roman in Paris, 1530–1534". *The Library*, vol. s5-XXIX, no. 1, Mar. 1974, pp. 5–20. doi.org/10.1093/library/s5-XXIX.1.5

Harry Carter, ed., *Sixteenth Century French Type-founders: The Le Bé Memorandum.* Paris: André Jammes, 1967.

More recent research by Hendrik D.L. Vervliet indicates that the St. Augustin roman type used by Robert Estienne in the 1531 Isagoge and other books was cut not by Garamond but by a "Maitre Constantin" who cut five romans in the Aldine style for Estienne but of whom little else is known.

Hendrik D.L. Vervliet, *French Renaissance Printing Types: A Conspectus*. New Castle, DE, USA: Oak Knoll Press, 2010. Simultaneously published in London by the Bibliographical Society and the Printing Historical Society, p. 36.

(4) Charles Bigelow, Kris Holmes, "The design of Lucida", in *Text Processing and Document Manipulation*, J.C. van Vliet, ed. Cambridge: Cambridge University Press, 1986, pp. 1–17.

Charles Bigelow, Kris Holmes, "Science and history behind the design of Lucida". *TUGboat*, vol. 39, no. 3, pp. 204–211, 2018. tug.org/TUGboat/tb39-3/tb123bigelow-lucida.pdf

(5) Harry Carter, "The optical scale in typefounding", in *Typography* 4, pp. 144–148. London: The Shenval Press, 1937.

Reprinted as "Optical scale in type founding", *The Printing Historical Society Bulletin*, vol. 13, 1984, pp. 144–148. London: St Bride Institute. issuu.com/letterror/docs/harry_carter_optical_scale_in_typefounding

Harry Carter's influential essay is well known to type designers; see, for instance: Tim Ahrens and Shoko Mugikura, *Size Specific Adjustments to Type Designs*. Garching, Germany: Just Another Foundry, 2013.

The first scientific study of the effect of optical scale on legibility, to my knowledge is:

Kevin Larson, Matthew Carter, "Sitka: A collaboration between type design and science", in *Digital Fonts and Reading*, Mary C. Dyson, Ching Yee Suen, eds., pp. 37–53. Singapore: World Scientific, 2016. microsoft.com/en-us/research/publication/sitka-a-collaboration-between-type-design-and-science/

The authors conclude that type designs intended to optimize legibility at small sizes are also optimally legible at large sizes, while designs intended for large sizes are artistically pleasing at large sizes:

> The size-specific adjustments made for large sizes do not increase legibility for large sized text. If we want increased legibility at large sizes, we are better served using a small size-specific design. If our goal is instead some level of elegance or personality, then a large size-specific design is appropriate.

A review of a century of research on the effects of typeface features on legibility is:

Charles Bigelow, "Typeface features and legibility research". *Vision Research*, vol. 165, Dec. 2019, pp. 162–172.

(6) Daniel Berkeley Updike, *Printing Types: Their History, Forms, and Use, A Study in Survivals.* Cambridge, Massachusetts: Harvard University Press, 1937.

(7) Najib J. Majaj, Denis G. Pelli, Peri Kurshan, Melanie Palomares, "The role of spatial frequency channels in letter identification". *Vision Research*, vol. 42, no. 9, Apr. 2002, pp. 1165–1184. (Source for figure 3.)

Since the late 1970s, there have been several studies of how (postulated) frequency sensitive channels in the human visual system encode features of letters and text for recognition by readers. Evidence has emerged that different sizes of letters are detected and encoded by visual channels sensitive to different frequencies, although the findings are neither as clear-cut nor as simple as I believed when I wrote the present essay, in 1989.

Reading researcher Gordon Legge, summarizing findings by several researchers, including in his own laboratory, has written:

> These empirical results imply that letters of large angular size are identified by channels encoding edge features or other high-frequency components of the letters' spectra. Identification of tiny letters depends on channels that encode coarser features (lower frequencies in units of cycles per letter).

Gordon E. Legge, *Psychophysics of Reading in Normal and Low Vision*. Mahwah, NJ: Lawrence Erlbaum Associates, 2007, pp. 60–65.

The implication that large sized letters are recognized by their high-frequency (sharp) edges, while small sized letters are recognized by their low-frequency (soft) components reminds us of claims made by Harry Carter in "Optical Scale" (1937):

> Legibility is all that matters in 6- to 10-point types; so that their successful design is a technical, and not in the ordinary sense an artistic, achievement… In the design of founts from 20- to 72-point the artist comes into his element. The eye dwells on big letters instead of hurrying from one to another as quickly as it can make out their meaning, as it does in reading text-sized types. Every letter must therefore be worth looking at for its own sake… There is no technical virtuosity about the fact of cutting a 24-point letter: the problem is an artistic one. The pleasure given by a fine large type comes from the beauty of the design and the beauty of the workmanship.
>
> The whole problem of adapting type-design to optical susceptibilities is a fascinating and a very difficult one. It is only possible to nibble at it without having proper experimental apparatus and ample time.

Quantitative nibblings of spatial frequencies, or optical susceptibilities, in the recognition of letters, principally of type, include:

Arthur P. Ginsburg, "Visual information processing based on spatial filters constrained by biological data". Report No. AMRL-TR-78-129-VOL-1/2, Air Force Aerospace Medical Research Lab, Wright-Patterson AFB, Ohio, USA, 1978.

Robert Morris, "Spectral font signatures". Technical Report of the Department of Mathematics and Computer Science. Boston: University of Massachusetts, 1989.

Robert Morris, "Image processing aspects of type", in *Document Manipulation and Typography*, J.C. van Vliet, ed. Cambridge: Cambridge University Press, 1988.

Charles Bigelow and Donald Day, "Digital typography". *Scientific American*, vol. 249, no. 2, Aug. 1983, pp. 106–119.

Charles Bigelow, "On Type: Optical letter spacing for new printing systems". *Fine Print*, vol. 4, no. 3, Oct. 1977. Reprinted in *Visible Language*, vol. 11, no. 3, 1977, pp. 325–329, `visiblelanguage.herokuapp.com/issue/43`

(8) Division into three levels of typographic scale probably reveals a culturally ingrained preference rather than a perceptual spectrum. Indo-European languages and cultures, including English, often divide phenomena into three parts, as in the grammatical partitioning of adjectives into positive, comparative, and superlative forms (big, bigger, biggest), or the story of Goldilocks and the three bears, or Caesar's division of all Gaul into three parts. Had Caesar been a book designer instead of a general, he might have written, "Typographia est omnis divisa in partes tres."

A scientific analysis based on precise measures of visual size and reading speeds suggests there are two important divisions of typographic scale. In studies beginning in the 1980s, Gordon Legge and co-researchers have discovered a "critical print size" (CPS) below which reading speed decreases precipitously as type size decreases. Above the CPS, reading speed does not appreciably increase but instead reaches a plateau even as size increases. CPS is measured by the visual angle that a font subtends at the retina, which depends on two factors: the physical x-height of a typeface in print or on screen and the distance at which it is read.

As an example, assuming a reading distance of 16 inches (40 centimeters), the CPS of Times Roman (or Times New Roman, or any similar-enough font) is 9 point, giving a CPS of approximately 0.2 degrees of visual angle.

A review of the relationship of critical print size to typography:

Gordon E. Legge, Charles A. Bigelow, "Does print size matter for reading? A review of findings from vision science and typography". *Journal of Vision*, vol. 11, no. 5, Aug. 2011. `doi.org/10.1167/11.5.8`

(9) Latin text- and Greek tex- (the "tech-" of "technology" and "technique", as well as "TEX", and the "-tect-" of "architect" and "tectonic") can both be traced back to a reconstructed Proto-Indo-European root, *teks-, meaning weaving and fabrication. The use of computer technology to weave text reunites aspects of an ancient craft.

A striking conjunction of typography with weaving is in two books woven in silk by a nineteenth century weaving firm in Lyon, France. A small, 20 page book of a poem, "Les Laboureurs", extracted from a larger work by Alphonse de Lamartine, was digitized and woven in silk on a Jacquard loom by the lace-making firm of J.-A. Henry in Lyon, France. It was exhibited at the 1878 Paris Exposition. The body size of the digitally woven text is small, approximately 8.5 point. "Les Laboureurs" was followed in 1886 by a 50 page book (*Livre de Prières tissé …*) woven by the same methods, and also elaborately ornamented and illustrated, which won a prize at the 1889 Universal Exposition in Paris, where the newly completed Eiffel Tower was a celebrated phenomenon.

(10) El (Lazar Markovich) Lissitzky, "Our Book", in *Gutenberg Jahrbuch* 1926 7. Mainz: Gutenberg-Gesellschaft. English translation by Helene Aldwinkel, in *El Lissitzky*, Sophie Lissitzky-Küppers, ed. London: Thames and Hudson, 1980.

(11) Andre Martinet, *Éléments de Linguistique Generale*. Paris: Librairie Armand Colin, 1967. Martinet follows an observation made by Ferdinand de Saussure in *Cours de linguistique generale*, Charles Bally and Albert Sechehaye eds., 1916. English edition: *Course in General Linguistics*, Wade Baskin, trans. New York: The Philosophical Library, 1959.

Articulation into words is a notion familiar to typographic literates because words are separated by blank spaces in typography (though they were not so in classical Greek and Latin manuscripts). "Morpheme" denotes an elementary meaningful unit of language, which may be a word or a significant part of a word. An English noun such as "bird" is a morpheme, as is the plural suffix represented by the letter "-s" in the plural noun "birds". Written morphemes may or may not be separated by word spaces, depending on the orthography of a language.

"Phoneme" denotes a psychologically distinguishable sound of speech. Unlike a word, a phoneme generally has no meaning in itself; its role is to be different from other phonemes and to make meaning through its interactions and combinations with them. Articulation into phonemes is familiar to readers of alphabetic scripts because many, though not all, phonemes are uniquely represented by a single letter. In English orthography, for example, the letters b, d, p, and t, among others, represent consonantal phonemes. Most orthographies are not, however, perfectly phonemic. English vowel phonemes, as a notorious example, do not have simple, one-to-one correspondences with the letters of written English. Few languages have more than 100 phonemes, excepting some African Khoisan languages which have a rich repertoire of click phonemes.

(12) Floyd Ratliff, "Contour and Contrast". *Scientific American*, vol. 226, no. 6, June 1972, pp. 9–20.

David H. Hubel, *Eye, Brain, and Vision*. New York: Scientific American Library, 1988.

See notes (7), (8), and (25) for findings that letter recognition depends on different channels of spatial frequencies, not simply on edges.

(13) Eric Gill, *An Essay on Typography*. London: Sheed & Ward, 1936. Reprinted, Boston: David R. Godine, 1988.

(14) Andre Malraux, "Museum Without Walls", in *The Voices of Silence*. Stuart Gilbert, trans. Princeton: Princeton University Press, 1978.

(15) John Dreyfus, *French Eighteenth Century Typography*. Cambridge, U.K.: The Roxburghe Club, 1982.

(16) A.V. Shubnikov and V.A. Koptsik, *Symmetry in Science and Art*. G.D. Archard, trans. New York: Plenum Press, 1974.

Branko Grünbaum and G.C. Shephard, *Tilings and Patterns*. New York: W.H. Freeman, 1986.

Hermann Weyl, *Symmetry*. Princeton: Princeton University Press, 1952.

(17) John R. Pierce, *An Introduction to Information Theory: Symbols, Signals and Noise*. New York: Dover Publications, 1980.

(18) Jan Tschichold, *A Treasury of Alphabets and Lettering*. New York: Reinhold, 1966.

Walter Tracy, *Letters of Credit*. Boston: David R. Godine, 1986.

(19) John R. Pierce, *The Science of Musical Sound*. New York: Scientific American Library, 1983.

(19A) Several studies have contradicted the popular hypothesis that decreasing letter spacing increases legibility by merging letters such that words are recognized as whole shapes instead of by their component parts, i.e., letters. A few representative articles:

Susana T.L. Chung, "The Effect of Letter Spacing on Reading Speed in Central and Peripheral Vision". *Investigative Ophthalmology & Visual Science*, vol. 43, no. 4, 2002, pp. 1270–1276.

Using monospaced Courier as the test font, the author found that letter spacing less than that of standard Courier did not increase reading speed, and, on average, apparently decreased it when the spacing was less than approximately 0.9 of standard spacing. Spacing greater than the standard value also failed to increase reading speed.

Denis G. Pelli, Bart Farell, Deborah C. Moore, "The remarkable inefficiency of word recognition". Letter to Nature, *Nature*, vol. 423, no. 6941, June 12, 2003, pp. 752–756.

The authors state: "Our results indicate that, rather than directly recognizing complex familiar objects, such as words, our visual system detects smaller components—letters or perhaps features of letters—and only then recognizes the object specified by these components."

Kevin Larson, "The Science of Word Recognition". Lecture at Association Typographique Internationale, Sept. 2003. `docs.microsoft.com/en-us/typography/develop/word-recognition`

The author summarizes: "Word shape is no longer a viable model of word recognition. The bulk of scientific evidence says that we recognize a word's component letters, then use that visual information to recognize a word."

Relevant discussion is found in:

Gordon E. Legge, *Psychophysics of Reading in Normal and Low Vision*. Mahwah, NJ: Lawrence Erlbaum Associates, 2007, pp. 94–96.

(20) Andre Martinet, "The Word". *Diogenes* 51, 1965.

(21) Pierre Simon Fournier, *Manuel Typographique*. Paris: Fournier (and Barbou), 1764–66; Harry Carter, trans. *Fournier on Typefounding*. New York: Burt Franklin, 1973.

First edition: Pierre Simon Fournier, Harry Carter, *Fournier on Typefounding. The Text of the Manuel Typographique, 1764–1766*. Translated and edited with notes by Harry Carter. [With a Portrait.] London: Soncino Press, 1930.

(22) Chauncey Griffith, quoted by Edmund C. Arnold in *Functional Newspaper Design*. New York: Harper & Brothers, 1956.

(23) Related by Hans Ed. Meier, personal communication. The design philosophy of Syntax is analyzed by Erich Schulz-Anker, "Syntax-Antiqua, a Sans Serif on a New Basis". *Gebrauchsgraphik* no. 8, 1970, pp. 49–56.

Syntax is a sans-serif based on proportions and letter forms of Renaissance humanist minuscule and classical Roman inscriptional letterforms. Released in 1968 under the name Syntax-Antiqua, it joined Edward Johnston's London Underground lettering and Eric Gill's Gill Sans as a "Humanist" sans-serif. In its narrow sense, the typographic term "Antiqua" means Humanist types of the Renaissance, also called "Garalde" (Garamond + Aldus) in a common type classification. In a broad sense, such as in the German DIN classification, Antiqua can mean nearly any style of seriffed typeface. The "Humanist sans-serif" has gained popularity since Syntax and has become an expanded genre within the sans-serif class.

Helvetica, designed by Max Miedinger, is a sans-serif typeface in the grotesque style cut in the mid-nineteenth century but refined in 1957 for modernist typography. Figure 11 shows the two typefaces for comparison.

(24) Blaise Pascal, Serie XXII, 512, in *Pensées*. Paris: Editions du Seuil, 1962.

(25) In this essay on typography, "texture" refers to the visual property of small sizes of text in which

the letter shapes are too small to be easily appreciated as forms on their own, but are nevertheless big enough to be recognized in normal reading. Different textures are perceptible aesthetic features of different typefaces.

Popular science accounts of the visual perception of texture include:

Bela Julesz, "Texture and Visual Perception", *Scientific American*, vol. 212, no. 2, Feb. 1965, pp. 38–48; and "Experiments in the Visual Perception of Texture", *Scientific American*, vol. 232, no. 4, Apr. 1975, pp. 34–43.

Some of Julesz's claims have been disproven by Persi Diaconis and David Freedman, "On the Statistics of Vision: The Julesz Conjecture". *Journal of Mathematical Psychology*, vol. 24, no. 2, 1981.

A more recent discussion and definition of texture, including its relation to reading:

Denis G. Pelli, Katherine A. Tillman, "The uncrowded window of object recognition". *Nature Neuroscience*, vol. 11, no. 10, Oct. 2008, pp. 1129–1135.

The authors state:

> We suggest that one might define "texture" as what one can see without object recognition.
>
> If we cannot recognize things in this part of our vision [the periphery], what do we see? We see stuff (unnamed texture) and perceive space (the shape of the scene we are in). With an effort, observers can name and describe texture, but this rarely happens.

(26) Pierce (1980).

(27) Updike, op. cit.

(28) Franz Boas and Charles Cultee (narrator), Chinook Texts. Washington: Smithsonian Institution, Bureau of Ethnology, 1894.

Melville Jacobs and Victoria Howard (narrator), Clackamas-Chinook Texts. Bloomington: Indiana University Research Center in Anthropology, Folklore, and Linguistics, 1958.

Dell Hymes, "Victoria Howard's 'Gitskux and his Older Brother': A Clackamas Chinook Myth", in Smoothing the Ground: Essays on Native American Oral Literature. Berkeley, CA: University of California Press, 1983.

Acknowledgements

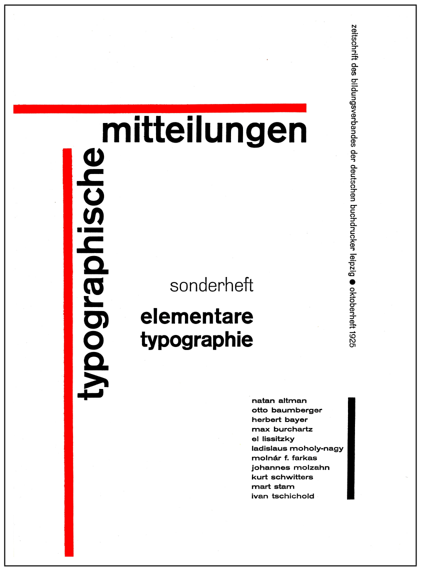
This essay is based on "The Literate Image: Form, Pattern, and Texture in Typeface Design", a talk given in the Stanford University Lectures in Art series, November 20, 1985, and on "New Technology of Type", a talk given to the Monotype Typographic Seminar, University College, University of London, September 27, 1984.

I am grateful to the late Jack Stauffacher for bringing Jan Tschichold's essay to my attention, to the late William Bright for discussions of the relation of typography to language, and especially to Kris Holmes for collaboration on the Lucida family of typefaces, which provided the opportunity to consider these matters in practice as well as theory.

I am also grateful to the late Sandra Kirshenbaum, founder, editor, and publisher of *Fine Print*, in which this article first appeared in print. It was my pleasure to serve as an associate editor for typography and as an occasional writer for *Fine Print* over several years.

◇ Charles Bigelow
`https://lucidafonts.com`

Postscript: A final bonus image, a cover of a special issue of the printing trade journal "typographische mitteilungen", October 1925, designed by Jan Tschichold with sans-serif types. This definitive introduction to typographic modernism by its leading practitioner shows texture and pattern, while form is implied by the rectilinear structure of vertical and horizontal title words aligned with geometric rules. Compare with Figure 7, a title page in classical style designed by Tschichold 22 years later with seriffed types, also showing form and pattern.



Arabic text justification using LuaLaTeX and the DigitalKhatt OpenType variable font

Amine Anane

Abstract

Arabic script is a cursive script where the shape and width of letters are not fixed, but vary depending on the context and justification needs. A typesetter must consider these dynamic properties of letters to achieve high-quality text comparable to Arabic calligraphy.

This article presents a proof-of-concept implementation of Arabic text justification, by varying letter shapes and widths, as a first step towards such high-quality Arabic typesetting. It uses LuaLaTeX and the DigitalKhatt OpenType variable font produced from a METAFONT-based dynamic font.

1 Introduction: Justification in Arabic

Due to the cursive nature of Arabic script, where the letters are connected, a letter can have several distinct forms and width depending on its position in the word and the letters that surround it. For example, Figure 1 shows fourteen possible forms of the letter ب (beh), among which are extensible forms.

Therefore, unlike the Latin script where justification is mainly based on the distribution of whitespace, Arabic calligraphy takes advantage of this dynamic nature of letters to justify text using three main techniques, as explained below: kashida extension, wider form substitution, and ligature composition and decomposition.

1.1 Kashida extension

The curvilinear stroke that connects letters is called kashida or tatweel. The width of the kashida is variable and can be stretched or compressed to justify text. Table 1 shows kashida extension examples.

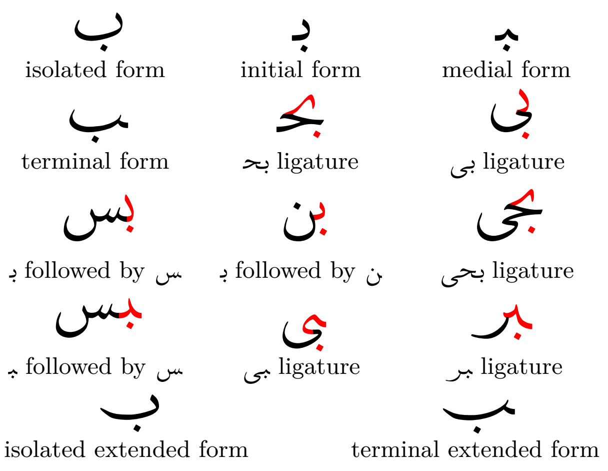


Figure 1: letter ب (beh) shapes

Table 1: Kashida extension

Min width	Natural width	Max width
حه	حه	حـــه
صا	صا	صـــا
حد	حد	حـــد
سو	سو	ســو
با	با	بـــا

1.2 Wider form substitution

Several Arabic letters have extended forms whose width can vary continuously within an interval. To stretch a line, a wider form is substituted. Table 2 shows some examples of wider forms.

Table 2: Wider form substitution

Original letter	Wider form	
	Min width	Max width
ب	ب	ب
س	س	س
ن	ن	ن
ف	ف	ف
ك	ك	ك

1.3 Ligature composition/decomposition

Arabic script makes extensive use of ligatures [2]. To stretch a line, an optional ligature can be decomposed to its constituent letters. Conversely, a less common ligature can be composed to shrink a text line. Table 3 shows some examples of ligature decomposition.

doi.org/10.47397/tb/42-3/tb132anane-variable

Table 3: Ligatures

Ligature	Constituent letters	
	Natural width	Max width
بح	بح	بـــح
لح	لح	لـــح
هم	هم	هـــم

To implement justification using the three techniques above, a dynamic font having variable glyph width and shape can be designed. By applying the justification rules provided by the font, the justification process will determine the width and shape of the glyphs to stretch or shrink a line of text. A few previous works [1, 2, 3, 12, 14] have been interested in Arabic justification using dynamic fonts. Unfortunately, these works still have significant challenges to overcome and they do not appear to be any further along in years.

On the other hand, most standard font formats, such as Apple Advanced Typography (AAT), Graphite and OpenType, provide mechanisms to control justification of text lines. However, Arabic justification has not benefited much from these solutions. For instance, the idea of using a variable font to stretch glyphs to do justification dates from TrueType GX, currently specified in AAT by the means of the `just` table. Unfortunately, this feature cannot be used since it is not supported by the Apple CoreText layout engine.

So, the approach adopted in this project is to start from the widely used OpenType standard and extend it to support Arabic justification using dynamic fonts, while drawing inspiration from existing solutions.

This article presents a proof-of-concept implementation of Arabic text justification by applying the three techniques above, using LuaLaTeX and the DigitalKhatt OpenType variable font. Section 2 introduces METAFONT and the VisualMETAFONT editor, used to design the DigitalKhatt font. Section 3 presents the OpenType layout engine, the justification algorithm and rules that have been developed. Section 4 shows an overview of OpenType variable fonts and explains how such a font is generated from the METAFONT. Section 5 gives the results obtained using LuaLaTeX and compares them with a handwritten Quranic text. Section 6 concludes with future work needed to further improve the results.

2 METAFONT

To design the glyph of an extensible letter, a function must be defined. This function produces shapes representing the extensible letter at different widths. At runtime, the justification process will determine the appropriate function arguments to justify text lines by stretching or shrinking some glyphs.

To design such parametric glyphs, the METAFONT [11] language[1] is used, which was specifically designed to deal with parametric fonts. Here are some of the advantages of METAFONT.

- It supports macros to extend the language with new syntax and operations.
- It supports deducing smooth cubic Bézier curves from high-level constructs such as direction, tension, curl, without having to specify each control point of the cubic curve.
- It supports a declarative style using linear equations.

For example, from the following METAFONT code

```
draw (0,0) .. (10,0) .. cycle
```

the following curve is generated: . METAFONT deduces the control points in such a way to have a smooth curve as close as possible to a circle. This default behavior can be changed by giving the tension or direction at some points. The following code

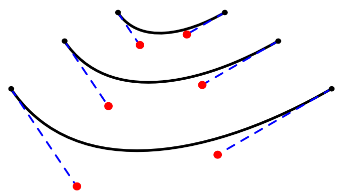
```
draw (0,0) .. tension 1.5 ..
     (10,0) .. tension 1.5 .. cycle;
```

applies some tension to the end points to obtain an ellipse-like shape: . This feature of automatically calculating the control points can be useful to define an extensible kashida. As an example to illustrate this, the following code

```
draw (0,0){dir -56}
  .. {dir 30} (50 + extension,0);
```

produces the following shapes by setting 0, 50 and 100 to the variable `extension`. METAFONT has automatically shifted the control points linearly with respect to the variable `extension`, allowing it to keep the same curve shape.

[1] Specifically, the METAPOST program [7, 8, 9] was used, an altered version of METAFONT which produces vector graphic commands instead of raster images. The term 'METAFONT' is kept since it is used for font and meta-font design (i.e., functions generating functions).



In addition to the kashida having variable width, it can have several shapes depending on the connecting letters. To simplify the design of the font, a step-by-step approach has been adopted. As a first step, a generic stretchable kashida has been used to join most of the connecting letters. So a functional parametric font can be obtained as quickly as possible, from which justification experimentation can be started. Subsequently a gradual refinement of the kashidas will be undertaken, as mentioned in section 6.

To design such a generic kashida a new operator `join` is added. The following METAFONT code uses this new operator to define the glyph `behshape.medi`.

```
defchar(behshape.medi,-1,-1,-1,-1);
%%beginparams
z1 = (219.986,47.4628);
z3 = (134.906,31.769);
%%beginverbatim
leftExtRatio := 8;
rightExtRatio := 10;
righttatweel_const := (120,-y1);
z2 = z1 + (penwidth,0) rotated 70;
z4 = z3 + (penwidth,0) rotated 83;
%%beginpaths
fill (181.269,99.7718) {dir -116.055} .. z3
    join z4 .. tension 0.932615 and 2.98102 ..
    (182.631,170.321) .. tension 1.7264 and
    1.23542 .. {right}(195.366,180.062) ..
    tension 1.7264 and 1 .. z2 join z1 .. {curl
    1} cycle;
%%endpaths
enddefchar;
```

The `join` operator can accept arguments to have more control over the form of the kashida. For example, the `leftExtRatio` and `rightExtRatio` variables above influence the curvature of the left and right kashidas. The `defchar` command generates the "glyph function"

⟨*glyph_name*⟩`_(leftExt,rightExt)` (1)

which produces the glyph shape as a function of the argument values provided. Here is the result of the function `behshape.medi_` called with different arguments to stretch or shrink the kashidas: . In addition, the `join` operator defines the left and right anchors used to specify the OpenType cursive attachment connections between glyphs.

Another advantage of METAFONT is the possibility of using elliptical and polygonal pens to describe curves. However, this feature was not used, in order to easily generate from the METAFONT font an OpenType version which can be used elsewhere. So all the glyph shapes are described using only cubic curve outlines.

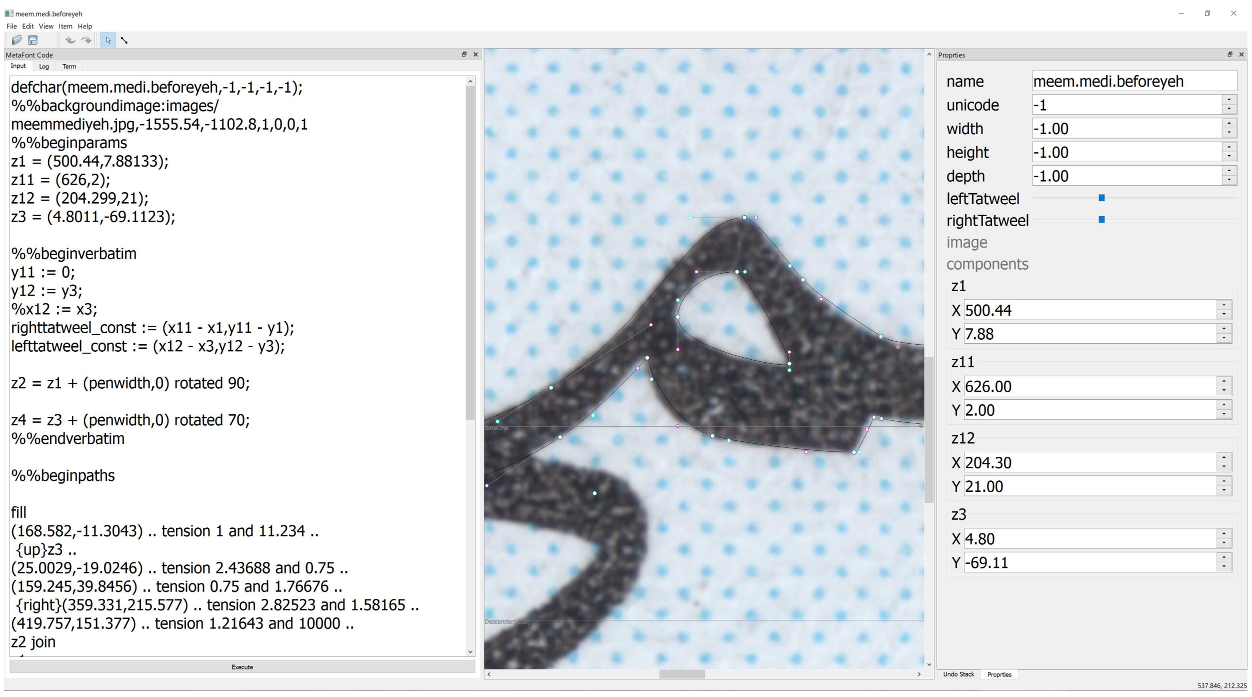
On the other hand, a major drawback of METAFONT is that a font designer has to write code to design a glyph. To validate that the glyph is correct, it must be visualized by executing the code and generating the glyph image with some labeling points to help debug the code and fine-tune the result. This process repeats until getting the desired result.

This fine-tuning process becomes more tedious if there are several parameters, since it is important to see how the glyph behaves by changing the parameter values. Also, if the font design is based on tracing of handwritten letters, as it is the case in this project, the task becomes even slower.

So, a visual graphic editor was developed to overcome this limitation, called VisualMETAFONT. This editor is based on Qt, a C++ software development framework. VisualMETAFONT uses the `mplib` library [9], a project supported by the LuaTeX team in order to turn the METAPOST interpreter into a system library that can be used by other applications. It is also based on HarfBuzz, a text-shaping engine supporting OpenType, AAT, and Graphite used in many applications and devices such as LuaLaTeX, Android, Chrome, Firefox, XeTeX and Qt.

Figure 2 shows the VisualMETAFONT window for glyph design. It is composed of three panes. The left pane is the METAFONT code of a single glyph. Each glyph starts with a `beginchar` or `defchar` command which provides its name, Unicode code point and glyph dimensions, if applicable. The code for a glyph can have five sections. The first section is a path to an image file that will be imported, to be traced around. The second section defines the parameters that will be controlled graphically. The third section is free METAFONT code to add other variables and define the relationships between them. The fourth section defines the paths that will be manipulated graphically. The fifth and last section is free METAFONT code. Therefore, there is no limit on what can be used in the METAFONT code. On the other hand, there are limits to what can be managed graphically, although features can be added to the GUI as needed.

The outline resulting from the execution of the METAFONT code is shown in the middle pane. The points that can be manipulated graphically are shown with different colors depending on their functions



```
defchar(meem.medi.beforeyeh,-1,-1,-1,-1);
%%backgroundimage:images/
meemmediyeh.jpg,-1555.54,-1102.8,1,0,0,1
%%beginparams
z1 = (500.44,7.88133);
z11 = (626,2);
z12 = (204.299,21);
z3 = (4.8011,-69.1123);

%%beginverbatim
y11 := 0;
y12 := y3;
%x12 := x3;
righttatweel_const := (x11 - x1,y11 - y1);
lefttatweel_const := (x12 - x3,y12 - y3);

z2 = z1 + (penwidth,0) rotated 90;

z4 = z3 + (penwidth,0) rotated 70;
%%endverbatim

%%beginpaths

fill
(168.582,-11.3043) .. tension 1 and 11.234 ..
 {up}z3 ..
(25.0029,-19.0246) .. tension 2.43688 and 0.75 ..
(159.245,39.8456) .. tension 0.75 and 1.76676 ..
 {right}(359.331,215.577) .. tension 2.82523 and 1.58165 ..
(419.757,151.377) .. tension 1.21643 and 10000 ..
z2 join
```

name	meem.medi.beforeyeh
unicode	-1
width	-1.00
height	-1.00
depth	-1.00
leftTatweel	
rightTatweel	
image	
components	
z1	X 500.44
	Y 7.88
z11	X 626.00
	Y 2.00
z12	X 204.30
	Y 21.00
z3	X 4.80
	Y -69.11

Figure 2: VisualMETAFONT glyph window

(i.e., inline point, left and right control points, parameter of type point). Each change in the position of one of these points generates new METAFONT code which is reflected immediately in the visual pane.[2]

The right pane displays the glyph parameters which can also be controlled graphically using slider and spin widgets, especially for numeric parameters without a graphic representation in the middle pane. For instance, by sliding the widgets for `leftTatweel` and `rightTatweel`, the user can view how the glyph width and shape behave for different values.

Once the dynamic glyphs are designed, a font designer needs to define the OpenType and justification rules that govern how a Unicode-encoded text will be presented using these glyphs. The next section presents these rules and the algorithms behind them.

3 Justification

This section starts by introducing the OpenType layout engine and its mechanism of lookups and rules. Next, the justification process is presented; it inherits the same notions of lookups and rules.

The OpenType layout engine takes as input an ordered list of lookups given a language and a set of features. Each lookup contains an ordered list of rules having the same type of action, which is either glyph substitution or glyph positioning. A substitution action replaces one sequence of glyphs by another. It is used for different purposes such as composition, decomposition, ligature, and alternates. A positioning action alters the advance width and offset position of a glyph. It is used for kerning, cursive attachment, mark to base position or mark to mark positioning. An OpenType rule can be considered as a simple form of pattern matching with the following general form:

$$\textit{backtrack input lookahead} \rightarrow \textit{action}$$

If the input, backtrack and lookahead sequences match the glyph string at the current glyph then the layout engine applies the action of the rule to the input sequence and advances the current glyph to the glyph after the input sequence. The layout engine does the substitution lookups first, then the positioning lookups, as shown in Algorithm 1.

The same principle of lookups and rules is used for justification. Indeed, by applying ligature compositions and decomposition, glyph alternates, kerning, and cursive attachment it is possible to shrink or extend a line until desired width is achieved. This technique of justification has been used in the Oriental TeX Project [6].

The current implementation uses the same idea. Two sets of steps can be defined: one used to shrink text lines and one used to stretch text lines. Each step contains a set of lookups that will be applied in

[2] Due to the speed of METAFONT and contemporary machines, the visual design is very smooth and no slowness or lagging is experienced.

Algorithm 1: OpenType layout algorithm

```
input  : Unicode text
output : List of glyph id, advance widths,
         offset positions
glyphRun ← map Unicode text to glyph id;
foreach stage of {substitution, positioning} do
  foreach lookup of stage's lookups do
    foreach currentGlyph in glyphRun do
      foreach rule of lookup do
        if rule matches at currentGlyph then
          apply rule's action to input sequence;
          currentGlyph ← glyph after input seq;
          break;
        end
      end
    end
  end
end
```

the order given until reaching the desired line width. Listing 1 shows an example justification table, which will be discussed in section 5.

Listing 1: Justification table

```
table(just) {
  stretch {
    lookup expa.ligadecomp;
    lookup expa.kafalternate;
    lookup expa.alternates;
    lookup expa.seensad;
    lookup expa.taamar_haa_dal;
    lookup expa.alef_tah_lam_kaf;
    lookup expa.behshape_yeh_reh;
    lookup expa.waw_ain_qaf_fa;
    lookup expa.others_fina;
  }
  shrink {
    lookup shr1.ligatures;
    lookup expa.shrinkspace;
    lookup shr1.kaf;
    lookup expa.shrink;
    lookup shr1.kern;
    lookup shr1.dalcursive;
  } };
```

To take into account the dynamic aspect of the glyphs, a new lookup format is added. The idea is to assign for each expandable glyph two attributes defining its ability to stretch and to shrink, as with TEX glue. (As future work, it would be interesting to study the possibility of including these attributes in breaking of paragraphs into lines by TEX's line-breaking algorithm.) Listing 2 shows an example of this new format of justification lookup. The class `@haslefttatweel` in this example contains all the glyphs having an extensible kashida. If one of these glyphs is followed by a `heh.fina` or `dal.fina` then the justification attributes defined by the lookup `expa.haa.l1` are used.

Listing 2: Justification lookup

```
lookup expa.taamar_haa_dal {
  feature sch1;
  lookupflag IgnoreMarks;
  lookup expa.haa.l1 {
    sub behshape.init expfactors 0 5 0 0;
    sub behshape.medi expfactors 0 5 0 0;
    ...
    sub heh.fina expfactors 0 0 0 2;
    sub dal.fina expfactors 0 0 0 2;
  } expa.haa.l1;
  ...
  sub @haslefttatweel'lookup expa.haa.l1
     [heh.fina dal.fina]'lookup expa.haa.l1;
} expa.taamar_haa_dal;
```

So, depending on the context, the lookup specifies which glyph will be substituted by another, if any, in addition to the maximum desired stretch or shrink for the left and right kashidas. For example, Listing 2 specifies that `behshape.init` and `behshape.medi` can stretch by an argument value of `leftExt` that cannot exceed 5 and `heh.fina` and `dal.fina` can stretch by an argument value of `rightExt` that cannot exceed 2.

The justification proceeds in two phases. The first phase constitutes the execution of the lookup, as given by Algorithm 1. The result of this execution is the set of glyphs to be substituted, including their extension attributes. The second phase calculates the stretch/shrink values and applies them to the glyph string, after substituting the affected glyphs.

Even though there are different justification techniques, as given in section 1, from the point of view of the justification algorithm all these techniques constitute substitutions. Indeed, the kashida technique is a special case of justification by alternates, which is a substitution of one glyph by another — only in this case it may or may not be the same glyph. The same is true for ligature composition and decomposition, which is a substitution of one sequence of glyphs with another, and thus represents a more general case than the two previous techniques. The algorithm takes into account this general case.

Here is the problem stated more formally. As the result of the first phase, we will have m sequences

S_k to be substituted, where each sequence contains one or more glyphs. Let Δw_i be the amount that will be added to, or subtracted from, the width of the line, if the glyph i is substituted, and let e_i its maximum stretchability or shrinkability. Let $\Delta W_k = \sum_{i\in S_k} \Delta w_i$. Let Δd the amount by which we want to stretch or shrink a text line. So the problem of the second phase is to find the set of sequences R and the extensibility ratio r such as

$$\sum_{k\in R}(\Delta W_k + \sum_{i\in S_k} e_i * r) \leq \Delta d$$

If we want to find the solution which maximizes the above sum in order to minimize the gap Δd of the line, the problem becomes NP-hard. Indeed, if we ignore the term $\sum_{i\in S_k} e_i * r$ from the inequality above, the problem becomes an optimization problem of the subset sum problem, which is NP-hard.

We can think of some heuristics to solve this problem, or we can use an exact algorithm, when there are only a few substitutions to apply. In addition to minimizing the gap Δd, other criteria can be considered. For example, we may prefer to distribute the substitutions throughout the line rather than being concentrated at the beginning or at the end of the line. A deeper study of the rules of justification will dictate the way forward.

Currently, the algorithm implemented iterates over the sequences and chooses a sequence if its gap ΔW_k is less than the remaining line gap Δd, otherwise it skips to the next sequence. We can consider ordering the sequences such that the glyphs to be justified are at the left or the right of the line or distributed over the entire line. Algorithm 2 gives the steps in more detail.

For Algorithm 2 to work, it is necessary that the width difference Δw_i for each glyph i be linear in its arguments `leftExt` and `rightExt`, in the intervals between their current values and their maximum values specified by the justification rule.

Algorithm 2: Justification lookup algorithm

```
input  : $m$ sequences $S_k$ and $\Delta d$
output : a set of sequences $R$ and a ratio $r$
totExpansion $\leftarrow 0$;
$R \longleftarrow \emptyset$;
for $k \leftarrow 1$ to $m$ do
  if $\Delta W_k \leq \Delta d$ then
    $\Delta d = \Delta d - \Delta W_k$;
    totExpansion $\leftarrow$ totExpansion $+ \sum_{i\in S_k} e_i$;
    $R \longleftarrow R + \{S_k\}$;
  end
end
if totExpansion $> \Delta d$ then
  $r \leftarrow \Delta d/$totExpansion;
  $\Delta d \leftarrow 0$;
else
  $r \leftarrow 1$;
  $\Delta d \leftarrow \Delta d -$ totExpansion;
end
```

4 OpenType variable fonts

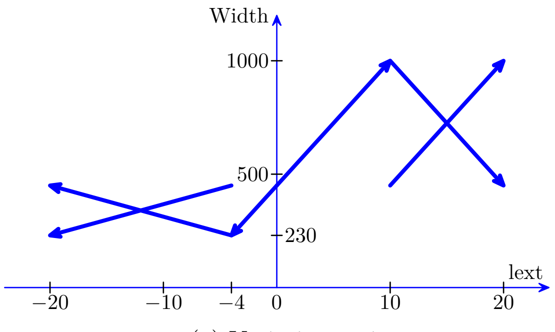
Variable fonts originated in Apple's TrueType GX and was adapted by OpenType in 2016. As variable fonts are used more and more, it is interesting to show that the justification technique already implemented with METAFONT also works for variable fonts.

Initially, the goal of using variable fonts was to allow continuous variations along design axis such as weight, width, slant, italic and optical size. An axis has a minimum, maximum and default values. For a given axis value, the layout engine use interpolation over regions of the variation space to calculate variable quantities such as glyph outline points, anchor position, and glyph advance width.

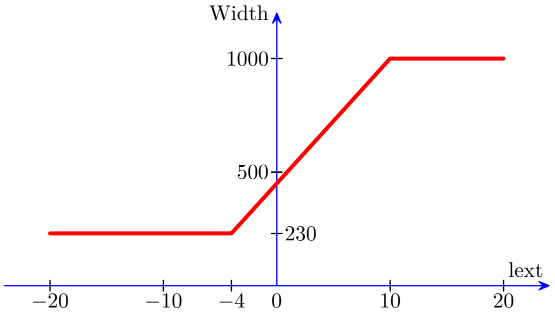
The same idea can be applied to glyph extension. In our case, two extension axes are defined, `lext` and `rext` corresponding to the `leftExt` and `rightExt` parameters. Each axis has a default value 0. The minimum and maximum values of the axes are chosen arbitrarily, and each glyph can decide how the axes' values are mapped to the argument values of the corresponding parameters.

To simplify the explanation below, we will assume that the minimum value of an axis is the minimum possible argument value of its corresponding parameter, and likewise for the maximum. A mapping that only requires two regions by glyph quantity is to return the minimum value of a quantity when the minimum value of the axes are given, and likewise for the maximum. However, this mapping can make the width difference Δw_i of the previous section non-linear in the interval containing 0 (i.e., the functions are discontinuous at 0) and therefore the justification rules should be adapted to this situation, and probably also Algorithm 2.

So, to get linear mapping functions all along the axis, the minimum values are returned when the minimum argument value of the glyph is given, and likewise for the maximum. For any axis value less than the minimum argument value, its minimum values are returned, and conversely for the maximum. This second mapping which preserves linearity was used to generate the OpenType variable font. So each glyph is represented with a maximum of six regions for each axis, three for the negative axis and three for the positive axis.



(a) Variation regions



(b) Resulting width function

Figure 3: Variation example

As an example, suppose that the minimum value for the `lext` axis is −20 and the maximum value is 20. Further suppose we have a glyph with a natural width of 450. Its maximum width is 1000, obtained when the argument value is 10, and the minimum width is 230, obtained when the argument value is −4. Figure 3a shows the 6 regions that should be defined for this glyph. Since the results of the regions in the same segment add up, we get the resulting width function in Figure 3b which becomes constant when it reaches its maximum or minimum value.

At this point, the following question may arise: What is the advantage of using METAFONT if variable fonts can meet the need of a dynamic font to apply justification? As highlighted in section 2, METAFONT remains an excellent tool to design such a variable font, thanks to its power and flexibility. In addition, if METAFONT could be used during the layout process then its full power can be harnessed. Nevertheless, designing a dynamic glyph using interpolation can be easier than trying to adjust the direction and tension of the curve points, especially when there are multiple points to define. Fortunately, METAFONT can design a glyph as an interpolation function between two shapes. Also, it is often not possible to find the glyph with the desired maximum width. In this case, extrapolation can be used to deduce the maximum shape.

5 Experimental results

A prototype implementation was presented at the TUG 2019 meeting. The implementation is a patched version of HarfBuzz and `mplib` and accepts as input a METAFONT font. That implementation was compiled to WebAssembly and a demonstration was published at `www.digitalkhatt.org`.

This section presents the result of a prototype implementation based on LuaLaTeX which uses as input the DigitalKhatt OpenType variable font presented in the previous section. All the examples presented here are compiled with this patched version of LuaLaTeX.

LuaLaTeX supports OpenType fonts through the `luaotfload` package, which is an adaptation of ConTeXt modules. This package has three OpenType processing modes: `base`, `node` and `harf`. The `base` mode supports only OpenType features that can be mapped to traditional TeX ligature and kerning mechanisms, and the `node` mode uses the OpenType layout engine implemented in ConTeXt.

The `harf` mode [10] uses the HarfBuzz library linked in LuaHBTeX. Since HarfBuzz is already part of LuaHBTeX, the integration of the justification algorithm becomes much easier. Indeed, the justification algorithm is patched in HarfBuzz and linked with LuaTeX to generate a patched version of LuaHBTeX. The Lua interface to the HarfBuzz engine is also modified to include the new functionality.

Unlike `luaotfload`'s `node` mode, `harf` mode does not support OpenType variable fonts. So the `node` modules supporting variable fonts are adapted to `harf` mode. In the `node` mode, variable glyphs are treated as LuaTeX virtual characters with special commands. During PDF generation, these commands are inserted each time the character is referenced, potentially leading to large file sizes. To remedy this, the virtual characters are converted to real characters of Type 3 fonts, created dynamically during PDF generation. This conversion also has the advantage of treating glyphs like characters, thus enabling PDF text functionality.

As a case study, the justification technique proposed by Microsoft to extend CSS2 and implemented in Internet Explorer (IE) is adapted here using only justification rules and therefore shows that the justification algorithm is general enough to support multiple justification scenarios.[3] Here is a description of the technique, as specified in [13].

1. Find the priority of the connecting opportunities in each word.

[3] This technique has been abandoned and to date there is no new CSS proposal for Arabic justification.

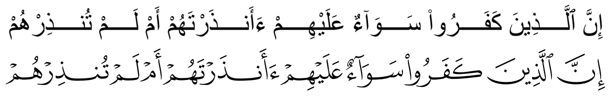
2. Add expansion at the highest priority connection opportunity.
3. If more than one connection opportunity has the same highest value, use the opportunity closest to the end of the word.

Table 4 defines the priority for connection opportunities and where expansion occurs. Since the algorithm deals with automatic justification, priority 1 for manual insertion of the kashida is ignored.

The IE implementation requires one kashida per word. To simplify the justification rules, this constraint is reduced to one kashida per module (i.e., cluster of connecting letters). IE also uses a CSS attribute `kashida-space` which defines the amount of justification given to kashidas in ratio to spaces. In our case, the amount of stretch is specified by the justification rules. When there is no more stretching possible, spaces will be used.

The CSS specification deals only with kashidas and does not mention the other justification techniques such as glyph alternates and ligature decomposition. In this case study, these techniques are added according to the priorities specified in the justification table of Listing 1. Optional ligature decomposition has the highest priority, followed by Kaf alternate, then by the other alternates.

Afterwards, kashida justification rules are defined according to the priorities presented in Table 4. These rules consider that the glyphs with a given priority participate in the justification process only after glyphs with higher priorities have exhausted their maximum stretchability or shrinkability. The following example compares the IE result (top) to the LuaLaTeX result (bottom), using the Times New Roman font after applying kashida justification. The line is stretched about 1.4 times its natural width.



IE justifies text by inserting flat kashidas unlike the curvilinear stretching of the current solution, which applies kashidas by substituting a wider glyph. Figure 4 shows how the text line stretches and shrinks, changing the line width by 5pt each time. Line 6 is set to the natural width of the text which gives a non-justified text. So the first five lines correspond to the shrinking case. Lines 5 and 4 are shrunk by decreasing the space between words due to the lookup `expa.shrinkspace`. Lines 3 and 2 shrink kashidas due to the lookup `expa.shrink`. The result of the shrink is similar to that obtained by using the pdf/LuaTeX font expansion feature. In this case, however, the expansion factor is a continuous value whose limit can depend on the context and the priority level. Line 1 uses in addition the lookup `shr1.dalcursive` to decrease the space before ينَ in the second word.

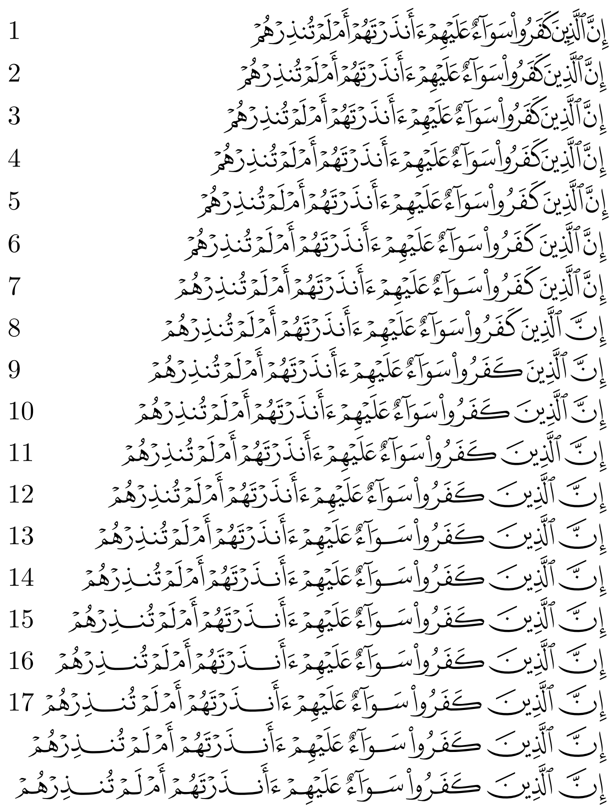


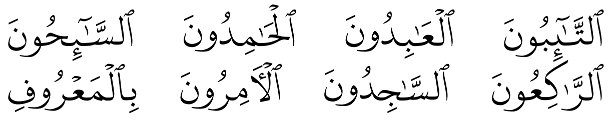
Figure 4: Continuous expansion

Line 7 shows two types of justifications. The first is the decomposition of the ligature هُمۡ in the last word, and the second is the kashida stretching of the letter seen in سَوَآءٌ , having priority 2 in Table 4. Line 8 uses the alternate wider glyph نَّ of the letter noon نَّ of the first word. However, the previous letter سَـ is no longer stretched. This is because, in line 7, there is not enough space to accommodate the wider نَّ, while in line 8 there is enough space, and since the alternates have priority over kashidas, the wider نَّ is chosen. In line 9, the wider alternate of letter kaf كَـ is used. Since it has priority over نَّ, the wider noon نَّ has shrunk a little. At line 10, the second alternate نَّ of the second word is substituted. From line 10 to line 13, the widths of نَّ and نَّ increase continuously until they reach their limits; at this moment, the stretching of the letter سَـ comes into play again. At line 14 starts the stretching of نـذ and نذ, of priority 3, until reaching their limits at line 18 where begins the stretching of kashidas of priority 7, which continues to the last line 19.

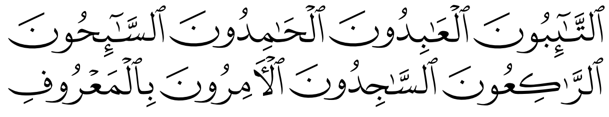
Table 4: Priority for kashida opportunities in CSS/IE proposal

Pri	Glyph	Condition	Kashida location
1	User-inserted Kashida	The user entered a Kashida in a position.	After the user inserted kashida
2	Seen, Sad	Connecting to the next character (initial or medial form).	After the character.
3	Taa Marbutah, Haa, Dal	Connecting to previous character.	Before the final form of these characters.
4	Alef, Tah, Lam, Kaf, Gaf	Connecting to previous character.	Before the final form of these characters.
5	Ra, Ya, Alef Maqsurah	Connected to medial Baa	Before preceding medial Baa
6	Waw, Ain, Qaf, Fa	Connecting to previous character.	Before the final form of these characters.
7	Other connecting characters	Connecting to previous character.	Before the final form of these characters.

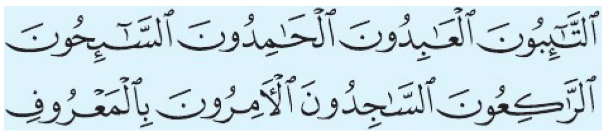
Figure 5 shows an example which compares the justification result of the current solution with the same handwritten text from which the font was designed. In the second line the wider alternates are not distributed evenly. For instance, the final noon ن of the second word has not been substituted by a wider noon ں, unlike the other two letters noons. Instead, there is some stretching before the final Reh ر of the third and fourth words which has not been considered among the justification cases of Table 4.


ٱلتَّٰٓئِبُونَ ٱلۡعَٰبِدُونَ ٱلۡحَٰمِدُونَ ٱلسَّٰٓئِحُونَ
ٱلرَّٰكِعُونَ ٱلسَّٰجِدُونَ ٱلۡأٓمِرُونَ بِٱلۡمَعۡرُوفِ

(a) justified text using interword space


ٱلتَّٰٓئِبُونَ ٱلۡعَٰبِدُونَ ٱلۡحَٰمِدُونَ ٱلسَّٰٓئِحُونَ
ٱلرَّٰكِعُونَ ٱلسَّٰجِدُونَ ٱلۡأٓمِرُونَ بِٱلۡمَعۡرُوفِ

(b) justified text using letter stretching


ٱلتَّٰٓئِبُونَ ٱلۡعَٰبِدُونَ ٱلۡحَٰمِدُونَ ٱلسَّٰٓئِحُونَ
ٱلرَّٰكِعُونَ ٱلسَّٰجِدُونَ ٱلۡأٓمِرُونَ بِٱلۡمَعۡرُوفِ

(c) handwritten text

Figure 5: Comparison with handwritten text

Figure 6 shows another example illustrating the difference between the handwritten text and LuaLaTeX.

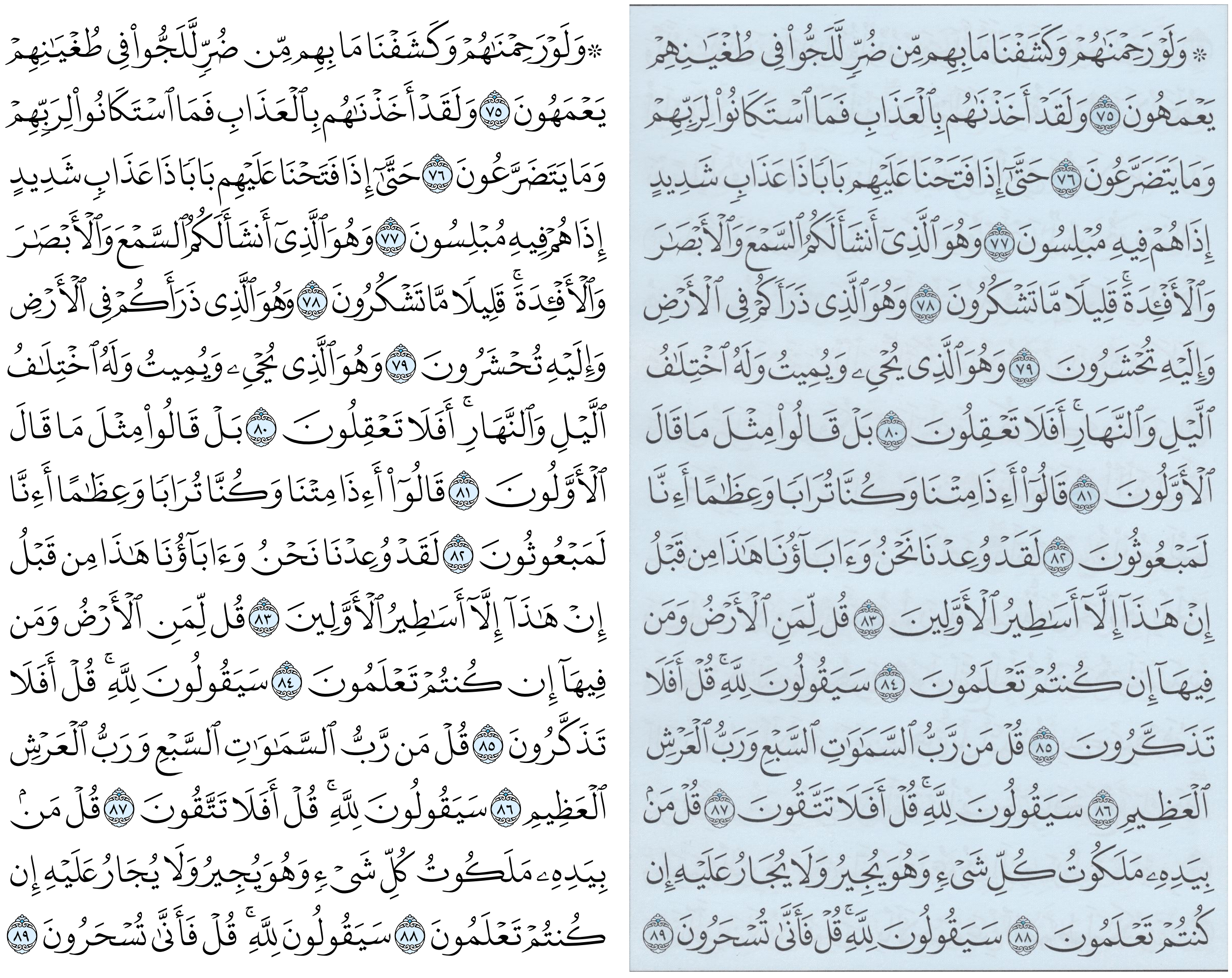
So far, all the examples presented are typeset in TeX's restricted horizontal mode using `\hbox`. Figure 7a shows an example of justified text using the current technique after TeX breaks paragraph into lines. Figure 7b shows the same text not justified. Some kashidas remain to be improved; however, this justification technique based on the continuous extension of letters conforms to Arabic calligraphy principles, allowing for better quality of Arabic text.

6 Future work

This article has presented a proof-of-concept implementation of Arabic justification based on letter extension techniques used in Arabic calligraphy, using LuaLaTeX and the DigitalKhatt variable font, which has achieved very promising results. This section presents some future work in the short and medium-term to continue the journey towards a high quality Arabic typesetter.

As a first step it is important to do more analysis and experiments of the different calligraphic rules used for Arabic justification to answer some important questions, such as: Which justification approach is preferable over another and in what context? If there are many justification choices with same preference, how should the space be distributed among them? Some works [2, 4, 6] have studied these rules and it would be interesting to implement them and compare them with the handwritten text.

Secondly, the DigitalKhatt font needs to be improved, especially at the kashida level. For the moment, each connecting glyph starts and/or finishes at



۞وَلَوْ رَحِمْنَٰهُمْ وَكَشَفْنَا مَا بِهِم مِّن ضُرٍّ لَّلَجُّوا۟ فِى طُغْيَٰنِهِمْ
يَعْمَهُونَ ٧٥ وَلَقَدْ أَخَذْنَٰهُم بِٱلْعَذَابِ فَمَا ٱسْتَكَانُوا۟ لِرَبِّهِمْ
وَمَا يَتَضَرَّعُونَ ٧٦ حَتَّىٰٓ إِذَا فَتَحْنَا عَلَيْهِم بَابًا ذَا عَذَابٍ شَدِيدٍ
إِذَا هُمْ فِيهِ مُبْلِسُونَ ٧٧ وَهُوَ ٱلَّذِىٓ أَنشَأَ لَكُمُ ٱلسَّمْعَ وَٱلْأَبْصَٰرَ
وَٱلْأَفْـِٔدَةَ ۚ قَلِيلًا مَّا تَشْكُرُونَ ٧٨ وَهُوَ ٱلَّذِى ذَرَأَكُمْ فِى ٱلْأَرْضِ
وَإِلَيْهِ تُحْشَرُونَ ٧٩ وَهُوَ ٱلَّذِى يُحْىِۦ وَيُمِيتُ وَلَهُ ٱخْتِلَٰفُ
ٱلَّيْلِ وَٱلنَّهَارِ ۚ أَفَلَا تَعْقِلُونَ ٨٠ بَلْ قَالُوا۟ مِثْلَ مَا قَالَ
ٱلْأَوَّلُونَ ٨١ قَالُوٓا۟ أَءِذَا مِتْنَا وَكُنَّا تُرَابًا وَعِظَٰمًا أَءِنَّا
لَمَبْعُوثُونَ ٨٢ لَقَدْ وُعِدْنَا نَحْنُ وَءَابَآؤُنَا هَٰذَا مِن قَبْلُ
إِنْ هَٰذَآ إِلَّآ أَسَٰطِيرُ ٱلْأَوَّلِينَ ٨٣ قُل لِّمَنِ ٱلْأَرْضُ وَمَن
فِيهَآ إِن كُنتُمْ تَعْلَمُونَ ٨٤ سَيَقُولُونَ لِلَّهِ ۚ قُلْ أَفَلَا
تَذَكَّرُونَ ٨٥ قُلْ مَن رَّبُّ ٱلسَّمَٰوَٰتِ ٱلسَّبْعِ وَرَبُّ ٱلْعَرْشِ
ٱلْعَظِيمِ ٨٦ سَيَقُولُونَ لِلَّهِ ۚ قُلْ أَفَلَا تَتَّقُونَ ٨٧ قُلْ مَنۢ
بِيَدِهِۦ مَلَكُوتُ كُلِّ شَىْءٍ وَهُوَ يُجِيرُ وَلَا يُجَارُ عَلَيْهِ إِن
كُنتُمْ تَعْلَمُونَ ٨٨ سَيَقُولُونَ لِلَّهِ ۚ قُلْ فَأَنَّىٰ تُسْحَرُونَ ٨٩

۞وَلَوْ رَحِمْنَٰهُمْ وَكَشَفْنَا مَا بِهِم مِّن ضُرٍّ لَّلَجُّوا۟ فِى طُغْيَٰنِهِمْ
يَعْمَهُونَ ٧٥ وَلَقَدْ أَخَذْنَٰهُم بِٱلْعَذَابِ فَمَا ٱسْتَكَانُوا۟ لِرَبِّهِمْ
وَمَا يَتَضَرَّعُونَ ٧٦ حَتَّىٰٓ إِذَا فَتَحْنَا عَلَيْهِم بَابًا ذَا عَذَابٍ شَدِيدٍ
إِذَا هُمْ فِيهِ مُبْلِسُونَ ٧٧ وَهُوَ ٱلَّذِىٓ أَنشَأَ لَكُمُ ٱلسَّمْعَ وَٱلْأَبْصَٰرَ
وَٱلْأَفْـِٔدَةَ ۚ قَلِيلًا مَّا تَشْكُرُونَ ٧٨ وَهُوَ ٱلَّذِى ذَرَأَكُمْ فِى ٱلْأَرْضِ
وَإِلَيْهِ تُحْشَرُونَ ٧٩ وَهُوَ ٱلَّذِى يُحْىِۦ وَيُمِيتُ وَلَهُ ٱخْتِلَٰفُ
ٱلَّيْلِ وَٱلنَّهَارِ ۚ أَفَلَا تَعْقِلُونَ ٨٠ بَلْ قَالُوا۟ مِثْلَ مَا قَالَ
ٱلْأَوَّلُونَ ٨١ قَالُوٓا۟ أَءِذَا مِتْنَا وَكُنَّا تُرَابًا وَعِظَٰمًا أَءِنَّا
لَمَبْعُوثُونَ ٨٢ لَقَدْ وُعِدْنَا نَحْنُ وَءَابَآؤُنَا هَٰذَا مِن قَبْلُ
إِنْ هَٰذَآ إِلَّآ أَسَٰطِيرُ ٱلْأَوَّلِينَ ٨٣ قُل لِّمَنِ ٱلْأَرْضُ وَمَن
فِيهَآ إِن كُنتُمْ تَعْلَمُونَ ٨٤ سَيَقُولُونَ لِلَّهِ ۚ قُلْ أَفَلَا
تَذَكَّرُونَ ٨٥ قُلْ مَن رَّبُّ ٱلسَّمَٰوَٰتِ ٱلسَّبْعِ وَرَبُّ ٱلْعَرْشِ
ٱلْعَظِيمِ ٨٦ سَيَقُولُونَ لِلَّهِ ۚ قُلْ أَفَلَا تَتَّقُونَ ٨٧ قُلْ مَنۢ
بِيَدِهِۦ مَلَكُوتُ كُلِّ شَىْءٍ وَهُوَ يُجِيرُ وَلَا يُجَارُ عَلَيْهِ إِن
كُنتُمْ تَعْلَمُونَ ٨٨ سَيَقُولُونَ لِلَّهِ ۚ قُلْ فَأَنَّىٰ تُسْحَرُونَ ٨٩

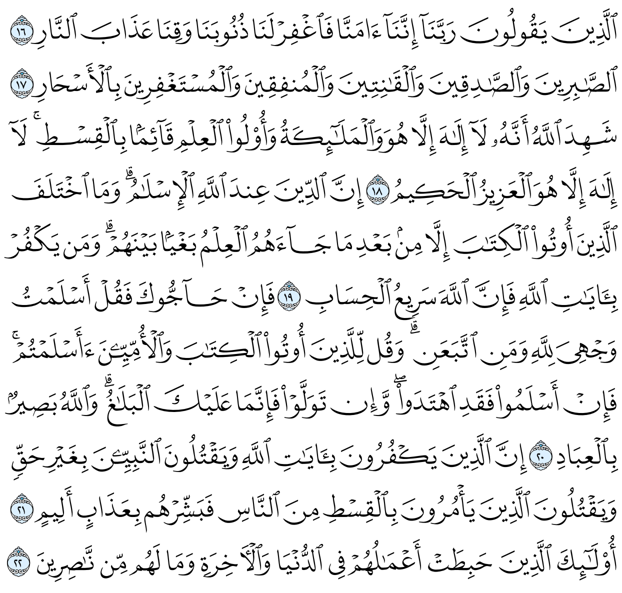
(a) LuaLaTeX (b) Page 347 of Madina Mushaf

Figure 6: Difference between LuaLaTeX and handwritten text

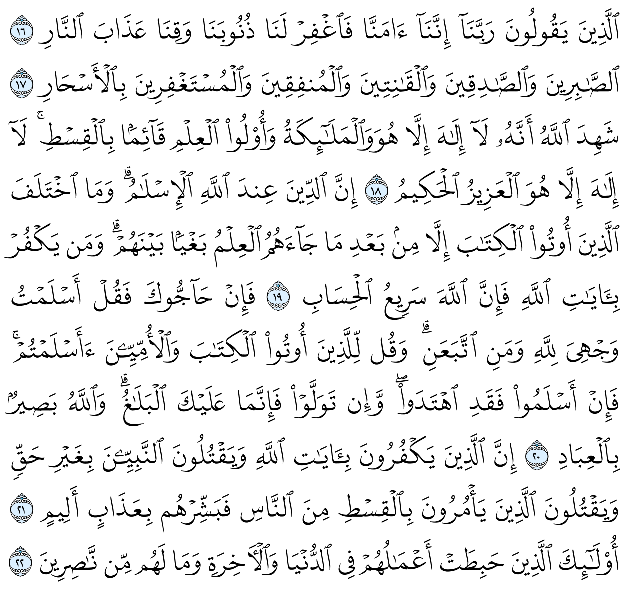
the minimum of the kashida. It is up to the justification rule to specify how much width the left curve of the kashida has and how much width the right curve has. A better approach would be to design connecting letters as a single stroke containing the whole kashida, and then the justification rule specifies the width of the whole kashida. Using METAFONT, the kashida will be split at specific points to produce each glyph part.

Another important work to consider is to embed code within the font in order to support custom rules. For instance, the width of some marks depends on the width of other glyphs, and custom rules are needed to express this. In a recent presentation [5], the author of HarfBuzz has proposed using WebAssembly to embed code within a font, since current technology allows it, which would bring great flexibility to the font developer. For instance, if this happens, it will be possible to embed the justification algorithm based on METAFONT within the font, thus allowing to take full advantage of METAFONT and to use it in any platform supporting this technology. So, as future work, WebAssembly will be considered to implement custom lookups and actions instead of using a specific scripting language. Also, to deal with complex Arabic calligraphic rules, adding custom lookups supporting the full power of regular expression and finite state machines could be necessary.

When typesetting the Quran by respecting the handwritten Mushaf, page by page and line by line, the difficulty will be more in terms of line shrinking than line stretching. Indeed, the calligrapher can visually optimize the space between letters and words so that some letters can appear above others and some marks can change their positions horizontally and vertically to fill the space, while avoiding some marks or letters being too close. Implementing this using


ٱلَّذِينَ يَقُولُونَ رَبَّنَآ إِنَّنَآ ءَامَنَّا فَٱغْفِرْ لَنَا ذُنُوبَنَا وَقِنَا عَذَابَ ٱلنَّارِ ١٦
ٱلصَّٰبِرِينَ وَٱلصَّٰدِقِينَ وَٱلْقَٰنِتِينَ وَٱلْمُنفِقِينَ وَٱلْمُسْتَغْفِرِينَ بِٱلْأَسْحَارِ ١٧
شَهِدَ ٱللَّهُ أَنَّهُۥ لَآ إِلَٰهَ إِلَّا هُوَ وَٱلْمَلَٰٓئِكَةُ وَأُو۟لُوا۟ ٱلْعِلْمِ قَآئِمًۢا بِٱلْقِسْطِ ۚ لَآ
إِلَٰهَ إِلَّا هُوَ ٱلْعَزِيزُ ٱلْحَكِيمُ ١٨ إِنَّ ٱلدِّينَ عِندَ ٱللَّهِ ٱلْإِسْلَٰمُ ۗ وَمَا ٱخْتَلَفَ
ٱلَّذِينَ أُوتُوا۟ ٱلْكِتَٰبَ إِلَّا مِنۢ بَعْدِ مَا جَآءَهُمُ ٱلْعِلْمُ بَغْيًۢا بَيْنَهُمْ ۗ وَمَن يَكْفُرْ
بِـَٔايَٰتِ ٱللَّهِ فَإِنَّ ٱللَّهَ سَرِيعُ ٱلْحِسَابِ ١٩ فَإِنْ حَآجُّوكَ فَقُلْ أَسْلَمْتُ
وَجْهِىَ لِلَّهِ وَمَنِ ٱتَّبَعَنِ ۗ وَقُل لِّلَّذِينَ أُوتُوا۟ ٱلْكِتَٰبَ وَٱلْأُمِّيِّـۧنَ ءَأَسْلَمْتُمْ ۚ
فَإِنْ أَسْلَمُوا۟ فَقَدِ ٱهْتَدَوا۟ ۖ وَّإِن تَوَلَّوْا۟ فَإِنَّمَا عَلَيْكَ ٱلْبَلَٰغُ ۗ وَٱللَّهُ بَصِيرٌۢ
بِٱلْعِبَادِ ٢٠ إِنَّ ٱلَّذِينَ يَكْفُرُونَ بِـَٔايَٰتِ ٱللَّهِ وَيَقْتُلُونَ ٱلنَّبِيِّـۧنَ بِغَيْرِ حَقٍّ
وَيَقْتُلُونَ ٱلَّذِينَ يَأْمُرُونَ بِٱلْقِسْطِ مِنَ ٱلنَّاسِ فَبَشِّرْهُم بِعَذَابٍ أَلِيمٍ ٢١
أُو۟لَٰٓئِكَ ٱلَّذِينَ حَبِطَتْ أَعْمَٰلُهُمْ فِى ٱلدُّنْيَا وَٱلْءَاخِرَةِ وَمَا لَهُم مِّن نَّٰصِرِينَ ٢٢

(a) justified text using letter stretching


ٱلَّذِينَ يَقُولُونَ رَبَّنَآ إِنَّنَآ ءَامَنَّا فَٱغْفِرْ لَنَا ذُنُوبَنَا وَقِنَا عَذَابَ ٱلنَّارِ ١٦
ٱلصَّٰبِرِينَ وَٱلصَّٰدِقِينَ وَٱلْقَٰنِتِينَ وَٱلْمُنفِقِينَ وَٱلْمُسْتَغْفِرِينَ بِٱلْأَسْحَارِ ١٧
شَهِدَ ٱللَّهُ أَنَّهُۥ لَآ إِلَٰهَ إِلَّا هُوَ وَٱلْمَلَٰٓئِكَةُ وَأُو۟لُوا۟ ٱلْعِلْمِ قَآئِمًۢا بِٱلْقِسْطِ ۚ لَآ
إِلَٰهَ إِلَّا هُوَ ٱلْعَزِيزُ ٱلْحَكِيمُ ١٨ إِنَّ ٱلدِّينَ عِندَ ٱللَّهِ ٱلْإِسْلَٰمُ ۗ وَمَا ٱخْتَلَفَ
ٱلَّذِينَ أُوتُوا۟ ٱلْكِتَٰبَ إِلَّا مِنۢ بَعْدِ مَا جَآءَهُمُ ٱلْعِلْمُ بَغْيًۢا بَيْنَهُمْ ۗ وَمَن يَكْفُرْ
بِـَٔايَٰتِ ٱللَّهِ فَإِنَّ ٱللَّهَ سَرِيعُ ٱلْحِسَابِ ١٩ فَإِنْ حَآجُّوكَ فَقُلْ أَسْلَمْتُ
وَجْهِىَ لِلَّهِ وَمَنِ ٱتَّبَعَنِ ۗ وَقُل لِّلَّذِينَ أُوتُوا۟ ٱلْكِتَٰبَ وَٱلْأُمِّيِّـۧنَ ءَأَسْلَمْتُمْ ۚ
فَإِنْ أَسْلَمُوا۟ فَقَدِ ٱهْتَدَوا۟ ۖ وَّإِن تَوَلَّوْا۟ فَإِنَّمَا عَلَيْكَ ٱلْبَلَٰغُ ۗ وَٱللَّهُ بَصِيرٌۢ
بِٱلْعِبَادِ ٢٠ إِنَّ ٱلَّذِينَ يَكْفُرُونَ بِـَٔايَٰتِ ٱللَّهِ وَيَقْتُلُونَ ٱلنَّبِيِّـۧنَ بِغَيْرِ حَقٍّ
وَيَقْتُلُونَ ٱلَّذِينَ يَأْمُرُونَ بِٱلْقِسْطِ مِنَ ٱلنَّاسِ فَبَشِّرْهُم بِعَذَابٍ أَلِيمٍ ٢١
أُو۟لَٰٓئِكَ ٱلَّذِينَ حَبِطَتْ أَعْمَٰلُهُمْ فِى ٱلدُّنْيَا وَٱلْءَاخِرَةِ وَمَا لَهُم مِّن نَّٰصِرِينَ ٢٢

(b) justified text using interword space

Figure 7: Text justification after TeX's line-breaking algorithm

rules will be very difficult due to the large number of possible cases. An algorithm which analyzes the available space to move marks and word modules and prevent collisions would be very useful.

Finally, it is interesting to study how to extend TeX's line-breaking algorithm in order to consider glyph stretching and shrinking properties before breaking paragraph to lines.

Acknowledgments

I would like to thank Barbara Beeton and Karl Berry for improving this article with their English and technical editing.

References

[1] A. Bayar, K. Sami. How a font can respect basic rules of Arabic calligraphy. *Intl. Arab. J. e-Technol.* 1(1):1–18, 2009. `cs.uwaterloo.ca/~dberry/HTML.documentation/KeshidePapers/HowFontCanRespectArabicCalligraphy.pdf`

[2] M.J.E. Benatia, M. Elyaakoubi, A. Lazrek. Arabic text justification. *TUGboat* 27(2):137–146, 2006. `tug.org/TUGboat/tb27-2/tb87benatia.pdf`

[3] D.M. Berry. Stretching letter and slanted-baseline formatting for Arabic, Hebrew, and Persian with ditroff/ffortid and dynamic PostScript fonts. *Softw. Pract. Exp.* 29:1417–1457, 1999. `doi.org/10.1002/(SICI)1097-024X(19991225)29:15%3C1417::AID-SPE282%3E3.0.CO;2-F`

[4] M. Elyaakoubi, A. Lazrek. Justify just or just justify. *J. Elect. Pub.* 13(1), Winter 2010. `doi.org/10.3998/3336451.0013.105`

[5] B. Esfahbod. Better-engineered font formats. `www.youtube.com/watch?v=fG1QEcl3yks&ab_channel=BehdadEsfahbod`

[6] H. Hagen, I.S. Hamid. Oriental TeX: optimizing paragraphs. *MAPS* 45:128–154, 2012. `www.ntg.nl/maps/45/12.pdf`

[7] J.D. Hobby. A METAFONT-like system with PostScript output. *TUGboat* 10(4):505–512, Dec. 1989. `tug.org/TUGboat/tb10-4/tb26hobby.pdf`

[8] J.D. Hobby. *A User's Manual for MetaPost*, 1994. `www.tug.org/docs/metapost/mpman.pdf`

[9] T. Hoekwater, L. Scarso. *MPlib API documentation, version 2.00*, 2018. `mirror.ctan.org/systems/doc/metapost/mplibapi.pdf`

[10] K. Hosny. Bringing world scripts to LuaTeX: The HarfBuzz experiment. *TUGboat* 40(1):38–43, 2019. `tug.org/TUGboat/tb40-1/tb124hosny-harfbuzz.pdf`

[11] D.E. Knuth. *The METAFONTbook*. Addison-Wesley Longman Publishing Co., Inc., USA, 1986.

[12] S. Mansour, H. Fahmy. Experiences with Arabic font development. *TUGboat* 33(3):295–298, 2012. `tug.org/TUGboat/tb33-3/tb105mansour.pdf`

[13] P. Nelson. Justifying text using cascading style sheets (CSS) in Internet Explorer 5.5/6.0. `web.archive.org/web/20030813215144/http://www.microsoft.com/middleeast/Arabicdev/IE6/KBase.asp`

[14] A. Sherif, H. Fahmy. Meta-designing parameterized Arabic fonts for AlQalam. *TUGboat* 29(3):435–443, 2008. `tug.org/TUGboat/tb29-3/tb93sherif.pdf`

◇ Amine Anane
`ananeamine (at) gmail dot com`
`https://github.com/DigitalKhatt`

Rendering open street maps

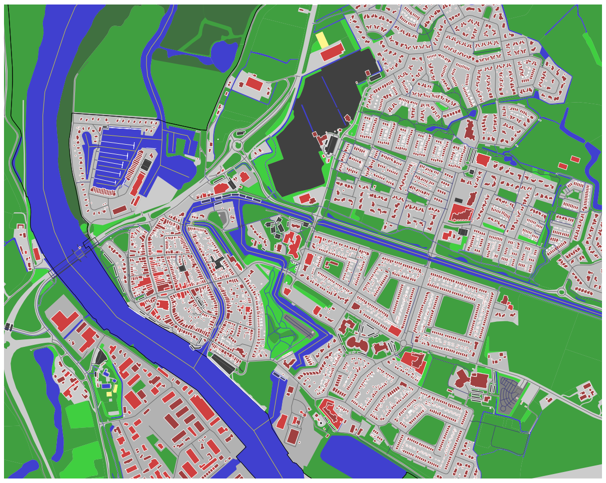
Hans Hagen

1 Introduction

At the 2021 ConTEXt meeting I did a presentation about rendering so-called open street maps with MetaPost. These are maps available on the web and in some mobile applications made from contributions by (public) organizations, governments, and volunteers. On the web these maps are rendered efficiently from cached tiles (bitmaps).

If you just want to render a map in ConTEXt and not be bothered with how this is accomplished, here is a recipe:

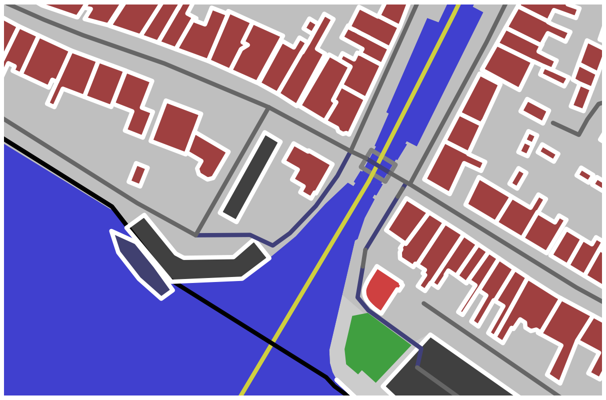
```
\usemodule[m-openstreetmap]
\startMPpage
  draw lmt_openstreetmap [
    filename = "hasselt.osm"
  ] ;
\stopMPpage
```



If you are interested in the details you can read on. I will roughly describe what it takes and show some TEX, MetaPost, Lua, and XML. You can find the code in the files `m-openstreetmap.mkxl` and `m-openstreetmap.lmt`. These (in the usual spacey-coded way) files take some 50 KB which demonstrates that modules that produce impressive graphics don't need to be large. Of course we fall back on plenty that is available in the ConTEXt code base.

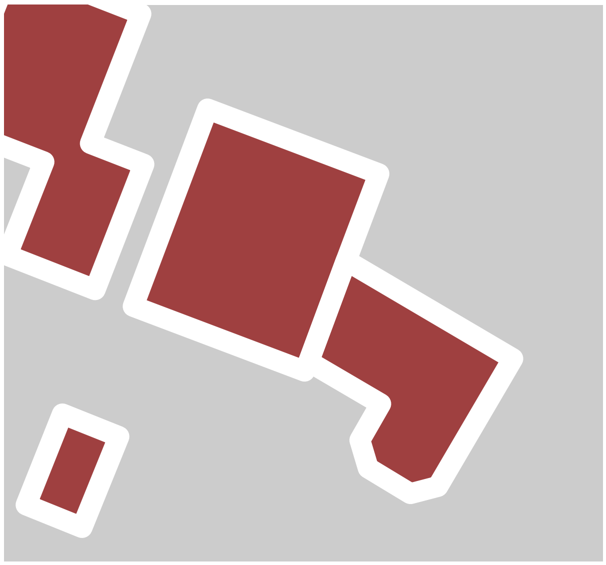
2 The XML files

In the web interface (`www.openstreetmap.org`) you can export a selection. There are some pointers to other exports. The `osm` file is an XML file that has been exported from the web interface. These files can become pretty large. The file used here describes my hometown and is some 12 MB, and when we add a bit of the surroundings it becomes 24 MB.[1] The small file results in this map:



Although we generate an outline, rendering is still pretty fast. The colors that I use are rather primary but at the ConTEXt meeting Hraban [Ramm] promised to come up with less primary colors. These maps are quite detailed so you can zoom in a lot, so for practical reasons I will use a smaller section.

In this smaller map you see the buildings as outlines. You cannot see how large the lot is that belongs to a house, but normally that's not how you use these maps.



The reason for coming up with this rendering is that on the mailing list a user wanted to know if we had ways to render a country's outline. I had already looked into that decades ago and a little browsing showed me that there were still no free high quality outlines available. At a BachoTEX meeting Mojca Miklavec and I had spent some time on rendering

[1] I found out that some sites have limitations on the total amount that you can export in a period of time. Others have slightly different export formats (for instance using coordinates instead of latitudes and longitudes).

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6">
 <bounds minlat="52.58941" minlon="6.09082" maxlat="52.58967" maxlon="6.09128"/>
 <node id="263682438" visible="true" lat="52.5895903" lon="6.0910379">
  <tag k="addr:city" v="Hasselt"/> <tag k="addr:housenumber" v="27"/>
  <tag k="addr:postcode" v="8061GH"/> <tag k="addr:street" v="Ridderstraat"/>
  <tag k="source" v="BAG"/> <tag k="source:date" v="2014-01-22"/>
 </node>
 <node id="2636834867" visible="true" lat="52.5894257" lon="6.0908799"/>
 <node id="2636834886" visible="true" lat="52.5894363" lon="6.0908368"/>
 ...
 <way id="258306565" visible="true">
  <nd ref="2636835112"/> <nd ref="2636835038"/> <nd ref="2636835025"/>
  ...
  <tag k="building" v="yes"/> <tag k="ref:bag" v="1896100000000287"/>
  <tag k="source" v="BAG"/> <tag k="source:date" v="2014-01-22"/>
  <tag k="start_date" v="1951"/>
 </way>
 <way id="258307233" visible="true">
  <nd ref="2636835038"/> <nd ref="2636835112"/> <nd ref="2636835042"/>
  ...
  <tag k="building" v="house"/> <tag k="ref:bag" v="1896100000004701"/>
  <tag k="source" v="BAG"/> <tag k="source:date" v="2014-01-22"/>
  <tag k="start_date" v="1951"/>
 </way>
</osm>
```

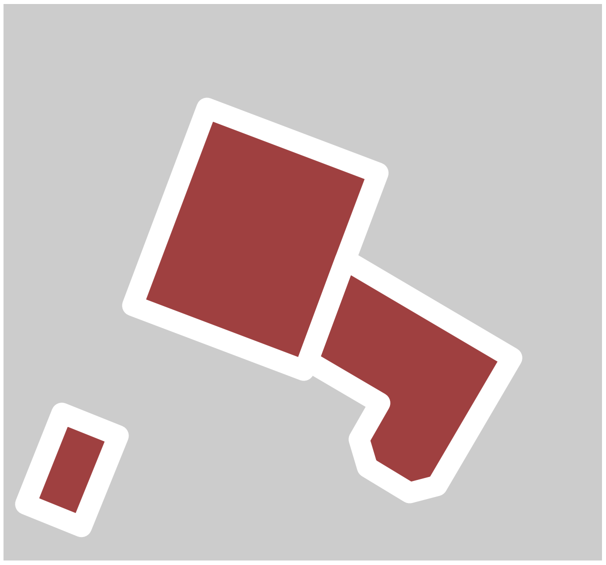
Figure 1: Stripped (and abridged) OpenStreetMap XML for three buildings in Hasselt

the open street map of that conference park in the woods with MetaPost, so I wondered if we could use that for country outlines. That code worked well but of course only dealt with what we encountered in the open street map XML at that time.

So, on a sunny (but too hot to stay outside) weekend I sat down in my attic (which overlooks the waterway you see in the rendering) to see what could be done. It was only after I finished the code that I found out that the country borders in these maps are not that usable. For instance, for the Netherlands the borders run through the North Sea, because part of it belongs to the Netherlands. This, although correct, doesn't give the view one is accustomed to. Because doing this is often a trial and error effort, it makes sense to use data that you are familiar with, which is why I choose Hasselt.

In the smallest map above, part of a neighboring house is shown. In the map below this has been removed, simply by editing the XML (shown in the code above). I also removed the `changeset`, `timestamp`, `user` and `uid` attributes because these bloat the output (and we don't need them).

When looking at the data I noticed the year `1951` which I always thought should be `1952`. Although this data likely comes from a government database I won't be surprised if it has errors. A while ago I



found out that the lot is actually multiple combined lots and some were registered in a peculiar way, but in general it can be revealing.

The `node` and `way` elements are what define the map. What exactly is present in the way is determined by the `tag` attribute. Once you look into the large file it becomes clear that the fact that numerous volunteers create the content on the one hand results in completeness but on the other hand also brings inconsistencies. It doesn't look like there is any periodic cleanup, so applications that deal with the data have to apply heuristics.

I am only interested in the graphics, not in the text. If there is demand I could look into it, in which case we probably also need some more control, because the additional (textual) information can be anything users add. The house is tagged as:

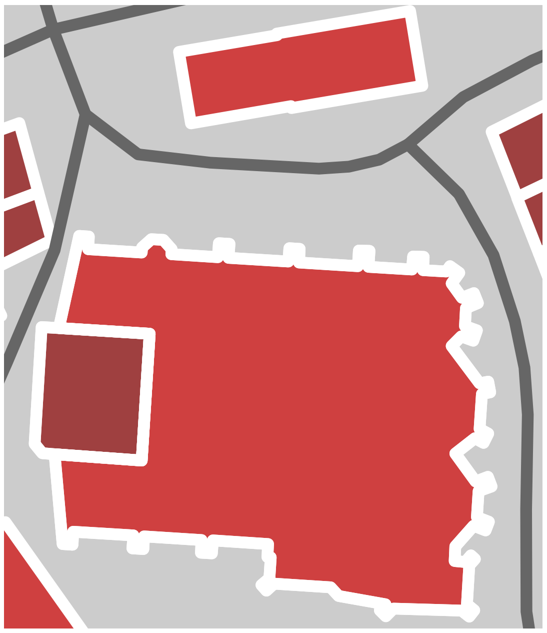
```
<tag k="building" v="yes"/>
```

The large old church close by (in the center of town) has tags:

```
<tag k="amenity" v="place_of_worship"/>
<tag k="building" v="yes"/>
<tag k="denomination" v="protestant"/>
<tag k="name" v="Grote- of Sint Stephanus kerk"/>
<tag k="ref:bag" v="1896100000001276"/>
<tag k="religion" v="christian"/>
<tag k="source" v="BAG"/>
<tag k="source:date" v="2018-08-08"/>
<tag k="start_date" v="1380"/>
```

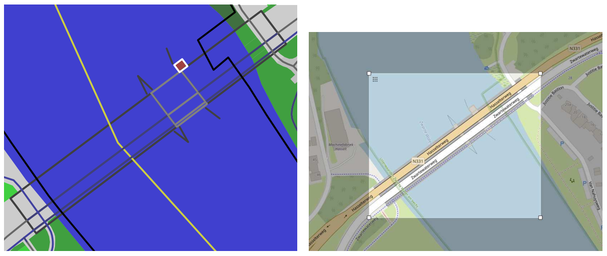
In order to determine what kind of building we have we need to look at `building` and/or `amenity` and their values. There is often some inconsistency in what gets assigned. The problem that arises from that is that one has to apply heuristics to determine the stacking order. For instance, ships float on water, buildings are on top of meadows and streets, bridges span water ways.

Here's another close-up:



The outline of the church has nice details and the tower can be handled separately because it is another `way`. The old town hall to the north lacks detail, but that's okay because the details are in the third dimension.

Next example: these bridges are tricky. There is information but my impression is that there is some clever combination of road information and bridge properties needed. The problem is not so much in recognizing the shapes but in the intended stacking order. For now, just to be sure, I use outlines but I probably need to spend some more time on this in the future.



3 The TEX and MetaPost files

It's now time to look into the (mid-2021) implementation and we start top down, with the MetaPost macro that one uses. Because we use ConTEXt LMTX, we can use the parameter driven interface:

```
\startMPdefinitions
  presetparameters "openstreetmap" [
    filename = "test.osm",
  % grid     = "dots",
    griddot  = 1.5,
  ] ;

  def lmt_openstreetmap = applyparameters
         "openstreetmap" "lmt_do_openstreetmap"
  enddef ;

  vardef lmt_do_openstreetmap = image (
      lua.mp.lmt_do_openstreetmap() ;
  ) enddef ;
\stopMPdefinitions
```

The `presetparameters` macro registers a parameter set (at the Lua end) and the `applyparameters` macro uses Lua to scan following parameters from a key/value list given between square brackets. When that is done the macro `lmt_do_openstreetmap` will be expanded. The parameter list is scanned using Lua functions that themselves use MetaPost scanning operations.

The `filename` parameter gets a string assigned because the scanner sees a string (expression) and `griddot` gets a number because a numeric expression is seen. From this you can deduce that we can also pick up pairs, colors, transforms, paths, pens, and we can also pick up a hash table (of parameters) between square brackets and an indexed table (array) by wrapping between curly braces. If you look at how these scanners are implemented you will be surprised how complex it is, simply because of the way MetaPost interprets its input. It is a mix of picking up tokens, symbols, known values, expressions with occasional lookahead and push back. This scanning interface is definitely more complex than the TEX scanners but the good news is that we have it all wrapped up in helpers like the ones mentioned here.[2]

The `lmt_do_openstreetmap` macro renders an `image` (which is a MetaPost macro returning a picture) by calling a Lua function. Here we use the `lua.mp` interfacing method; a more efficient variant would be to register a function and calling it by reference via `runscript` but that doesn't pay off here.

In the ConTEXt distribution there are plenty of examples where parameters are accessed from the MetaPost end but here we don't need that. We handle all at the Lua end:

```
function mp.lmt_do_openstreetmap()
  local specification = metapost.getparameterset
                                 ("openstreetmap")
  return openstreetmap.convert(specification)
end
```

At the TEX end surprisingly little happens: we only define some colors, for instance (from a set of 27):

```
\definecolor [osm:building] [r=.50]
\definecolor [osm:boat]     [b=.25]
\definecolor [osm:water]    [b=.75]
\definecolor [osm:forest]   [g=.75]
\definecolor [osm:sand]     [y=.75]
```

which then gets referenced in more detail, for instance (from a set of 173):

```
\definecolor[osm:amenity:hospital]
            [osm:building:special]
\definecolor[osm:amenity:townhall]
            [osm:building:special]

\definecolor[osm:barrier:gate]
            [osm:barrier]
\definecolor[osm:barrier:wall]
            [osm:barrier]

\definecolor[osm:boat:yes]
            [osm:boat]

\definecolor[osm:building:cathedral]
            [osm:building]
\definecolor[osm:building:residential]
            [osm:building]
\definecolor[osm:building:townhall]
            [osm:building]
```

[2] A further complication is that we can have multiple MetaPost instances so an implementation has to deal with that too.

From this you can conclude that much more detail in coloring is possible. On the web you can find CSS files with specifications, assuming some kind of order, but I didn't look into those much (I'm not going to set up a large rendering farm); I leave that to others.

4 The Lua file

The real work happens at the Lua end. Here we start by reading in the XML file using the parser built into ConTEXt. A 25 MB XML file loads reasonably fast but takes a bit of memory (because we store files in a round-trip way, prepared for filtering in and rendering with TEX). At some point I exported Fort Collins (the place where Alan Braslau, a MetaPost companion, lives) which gave a 125 MB file. It paid off to first strip the versioning information from the file after loading the blob, but that (rather trivial bit of) code is not shown here. In the following explanation some other code has been left out also, just to save paper and avoid confusion.

We start with some data tables. I mention these lists because they give an idea of what one has to deal with. The way objects get stacked is of relevance. We omit objects that make no sense and end up with:

```
local order = {
  "landuse", "leisure", "natural", "water",
  "amenity", "building", "barrier", "man_made",
  "bridge", "historic", "military", "waterway",
  "highway", "railway", "aeroway", "aerialway",
  "boundary",
}
```

We also need to determine what objects are polygons. There is a bit of back and forth involved here. For instance it makes sense in theory to add bridges here but that doesn't work out for Hasselt. Watch the mix of main categories and subcategories:

```
local polygons = tohash {
  "abandoned:aeroway", "abandoned:amenity",
  "abandoned:building", "abandoned:landuse",
  "abandoned:power", "aeroway", "allotments",
  "amenity", "area:highway", "craft",
  "building", "building:part", "club", "golf",
  "emergency", "harbour", "healthcare",
  "historic", "landuse", "leisure", "man_made",
  "military", "natural", "office", "place",
  "power", "public_transport", "shop",
  "tourism", "water", "waterway", "wetland",
}
```

Another piece of information is the stacking order. When we have a highway we have some 25 subcategories. For instance a `track` gets a value of `110`, a `path`, `footway` and `cycleway` use 100, and `steps` come on top with 190. There are of course more subcategories and categories to cover. Because all is in Lua tables, all can be tweaked and updated easily.

```
local stacking = {
  highway = {
    ...
    track          = 110,
    path           = 100,
    footway        = 100,
    cycleway       = 100,
    steps          = 190,
    ...
  },
  ...
}
```

What gets colored is also specified in tables. Here we show the subtable for boundaries. This `boundary` table demonstrates that there is some arbitrary tagging going on: there is `aboriginal_lands` but there are no tags for other lands (at least not that I could find now).

```
local colors = {
  amenity = {
    ...
  },
  boundary = {
    aboriginal_lands = true,
    national_park    = true,
    protected_area   = true,
    administrative   = true,
  },
  ...
}
```

We can fill areas but sometimes we need to force outlines, so we have a registry for this:

```
local forcedlines = {
  golf      = { "cartpath", "hole", "path" },
  emergency = { "designated", "destination",
                "no", "official", "yes" },
  historic  = { "citywalls" },
  leisure   = { "track", "slipway" },
  man_made  = { "breakwater", "cutline",
                "embankment", "groyne",
                "pipeline" },
  natural   = { "cliff", "earth_bank",
                "tree_row", "ridge", "arete" },
  power     = { "cable", "line", "minor_line" },
  ...
}
```

Normally we either draw or fill but sometimes we have to do both:

```
local lines = {
  amenity  = true,
  building = true,
  man_made = true,
  boat     = true,
}
```

Again, by looking at these tables you get an idea of the curious mix of tags. I was told (at the meeting) that anyone can add tags so I suppose that over time more has to be added to these tables. It's a bit like permitting any TEX user to add anything to a macro package without being strict with respect to how and where.

The conversion from XML data to MetaPost can be seen in `m-openstreetmap.lmt` and is not that complex. It is a typical example of "Sit down and just stepwise implement" with some testing as one progresses. For me the most time goes into the look and feel and having clean code, and here I also had to figure out the specification (and heuristics). Some safeguards and small extras (like drawing a grid on top) are not shown here.

The `f_` functions are what we call 'formatters' in ConTEXt which are variants of `string.format` that offer more features. We could use the `..` (string concatenation) which is probably faster but I prefer the formatters. The collected option can be used to either combine the path or output them separately. Combined paths permit transparency because crossing lines are not treated twice (strings are broken for *TUGboat* presentation):

```
local formatters = string.formatters

local f_draw = formatters['D %--t W "%s";']
local f_fill = formatters['F %--t--C W "%s";']
local f_both = formatters['P := %--t--C;'
          .. ' F P W "%s"; D P W "white" L 2;']
local f_draw_s = formatters['D %--t W "%s" L %s;']
local f_fill_s = formatters
                         ['F %--t--C W "%s" L %s;']
local f_both_s = formatters['P := %--t--C;'
          .. ' F P W "%s"; D P W "white" L %s;']

local f_nodraw     = formatters['ND %--t;']
local f_nofill     = formatters['NF %--t--C;']
local f_nodraw_s   = formatters['ND %--t;']
local f_nofill_s   = formatters['NF %--t--C;']

local f_background
 = formatters['F %--t -- C W "osm:background";']
local f_clipped
 = formatters['clip currentpicture to %--t--C'
             .. ' withstacking (0,250);']
```

The MetaPost wrapping blobs come first. We use short commands so that we don't have to pipe too much from Lua to MetaPost. The `no*` and `do*` commands are used to construct large paths instead of small snippets. This is similar to drawing font shapes. The resulting PDF is smaller and rendering can be faster. These commands are built into Metafun and use some of the magic available in the

MetaPost library. The shortcuts are defined in the preamble:

```
local beginmp = [[
  begingroup ;
  pickup pencircle scaled 1 ;
  save P ; path P ;
  save D ; let D = draw ;
  save F ; let F = fill ;
  save C ; let C = cycle ;
  save W ; let W = withcolor ;
  save L ; let L = withstacking ;
  save ND ; let ND = nodraw ;
  save DD ; let DD = dodraw ;
  save NF ; let NF = nofill ;
  save DF ; let DF = dofill ;
]]
```

The `L` shortcut expands to `withstacking` which is a native MPlib (3.0) extension.[3] When writing this summary I realized that for clipping a more advanced stacking method was needed, which is why `f_clipped` shown before specified the range to which the clip applies. Just for the record, the stacking property is just that: a property. It is the backend that does the ordering based on these properties.

We end the graphics definitions with:

```
local endmp = [[ endgroup; ]]
```

Between these two snippets we will make the graphic. The graphic operators are collected and flushed in one go. This all happens in the converter that we define next. Reporting, tracing and checking has been removed here but is of course present in the real code. First, we load the file and determine the bounds.

```
function openstreetmap.convert(specification)
 local root   = xml.load(specification.filename)
 local bounds = xml.first(root,"/osm/bounds")
```

Users can overload colors by providing a table in the parameter set (at the MetaPost end). Or instead one can just overload the TeX definitions shown before or use palettes.

```
local usercolors
  = specification.used -- from the parameter set
local usedcolors
  = table.copy(colors) -- preserve the originals

if usercolors then
  for k, v in next, usercolors do
    local u = usedcolors[k]
    if not u then
      -- error
    elseif v == false then
      usedcolors[k] = false
    elseif type(v) == "string" then
      for k in next, u do
        u[k] = v
      end
    elseif type(v) == "table" then
      for kk, vv in next, v do
        if vv == false then
          u[kk] = false
        elseif type(vv) == "string" then
          u[kk] = vv
        end
      end
    end
  end
end
```

[3] It could be implemented using `withprescript` and some backend filtering but a native mechanism is more efficient and permits restacking.

We do need to convert from `lat` (latitude) and `lon` (longitude). This helper used conversion code that Mojca (who is far more capable in math than I am) gave to me for the BachoTeX park graphic.

```
local minlat     = bounds.at.minlat
local minlon     = bounds.at.minlon
local maxlat     = bounds.at.maxlat
local maxlon     = bounds.at.maxlon
local midlat     = 0.5 * (minlat + maxlat)
local deg_to_rad = math.pi / 180.0
local scale      = 3600
                   -- vertical scale: 1" = 1cm

local f_f_pair = formatters["(%.3Ncm,%.3Ncm)"]

local function f_pair(lon, lat)
  return f_f_pair((lon - minlon) * scale
                 * cos(midlat * deg_to_rad),
                 (lat-minlat) * scale)
end
```

First we collect relevant data in tables. We need to do this because the stacking order is not the same as the order in the file. We could resolve everything via XML path lookups, but limiting the passes saves time. The real code is a bit more optimized. We could check for bad and redundant paths but it's not worth the effort.

Most of the parsing action is driven by the `xml.collected` iterators that filter the relevant elements. Much has to do with determining if something should be drawn (which can be specified), what color should be applied to a fill or outline, and where the object sits in the stacking order.

```
local insert      = table.insert
local rendering   = table.tohash(order)
local coordinates = { }
local ways        = { }
local result      = { }
local layers      = { }
local areas       = { }
```

```
for c in xml.collected(root,"/osm/node") do
  local a = c.at
  coordinates[a.id] = a
end
for c in xml.collected(root,"/osm/way") do
  ways[c.at.id] = c
end
for c in xml.collected(root,"tag[@k='area']") do
  areas[c] = c.at.v
end
for c in xml.collected(root,"tag[@k='layer']") do
  layers[c] = c.at.v
end
```

Although normally filtering is fast enough not to bother about performance, collecting nodes, ways, areas and layers is cheaper than filtering them from the (possibly huge) file each time. Most entries go into the nodes table.

As mentioned we can combine paths to save some space (not much). Another advantage is that it works better with transparency when a path crosses itself. This is what the do* and no* formatters are for: piecewise build a path and flush it afterwards. This is not native MetaPost but handled in the back-end where we go from the graphic output (in Lua tables) to PDF.

```
local function drawshapes(what,order)
  function xml.expressions.osm(k)
    return usedcolors[k]
  end

  local function getcolor(r)
    local t = xml.first(r,"/tag[osm(@k)]")
    if t then
      local at = t.at
      local v  = at.v
      if v ~= "no" then
        local k  = at.k
        local col = usedcolors[k][v]
        if col then
          return k, col, lines[k], stacking[k][v],
                 forcedlines[k][v]
        end
      end
    end
  end

  local function addpath(r, p, n)
    for c in xml.collected(r,"/nd") do
      local coordinate = coordinates[c.at.ref]
      if coordinate then
        n = n + 1 p[n] = f_pair(coordinate.lon,
                                coordinate.lat)
      end
    end
    return p, n
  end

  local function checkpath(parent,p,n)
    local what, color, both, stacking,
          forced = getcolor(parent)
    if what and rendering[what] then
      if not polygons[what] or forced
         or areas[parent] == "no" then
        insert(result,stacking
                      and f_draw_s(p,color,stacking)
                      or f_draw(p,color))
      elseif both then
        insert(result,stacking
                      and f_both_s(p,color,stacking)
                      or f_both(p,color))
      else
        insert(result,stacking
                      and f_fill_s(p,color,stacking)
                      or f_fill(p,color))
      end
    end
  end
```

There are ways and relations. Relations can have members that point to ways but also to relations. My impression is that we can stick to way members so I'll deal with more when needed.

```
for c in xml.collected(root,f_pattern(what)) do
  local parent = xml.parent(c)
  local tag = parent.tg
  if tag == "way" then
    local p, n = addpath(parent, { }, 0)
    if n > 1 then
      checkpath(parent,p,n)
    end
  elseif tag == "relation" then
    if xml.filter(parent,"xml://tag[@k='type'
         and (@v='multipolygon' or @v='boundary'
                or @v='route')]") then
      local what, color, both, stacking,
            forced = getcolor(parent)
      if rendering[what] then
        local p, n = { }, 0
        for m in xml.collected(parent,
              "/member[(@type='way')
                      and (@role='outer')]") do
          local f = ways[m.at.ref]
          if f then
            p, n = addpath(f,p,n)
          end
        end
        if n > 1 then
          checkpath(parent,p,n)
        end
      end
    end
  else
    for m in xml.collected(parent,
                 "/member[@type='way']") do
      local f = ways[m.at.ref]
      if f then
```

```
          local p, n = addpath(f, { }, 0)
          if n > 1 then
            checkpath(parent,p,n)
          end
        end
      end
    end
  end
end
```

Now we can wrap up. We add a background first and clip later. There can be substantial bits outside the clip path (like rivers) because they are defined as one `way`, but because paths are not that detailed we don't waste time on building a cycle. We could check if points are outside the bounding box and then use the MetaPost `buildpath` macro, at least if it works at all on these kinds of paths. It's not worth the trouble and probably would introduce errors too.

```
  local boundary = {
    f_pair(minlon,minlat),
    f_pair(maxlon,minlat),
    f_pair(maxlon,maxlat),
    f_pair(minlon,maxlat),
  }

  insert(result,beginmp)
  insert(result,f_background(boundary))

  for i=1,#order do
    local o = order[i]
    if usedcolors[o] then
      drawshapes(o,i)
    end
  end

  insert(result,f_clipped(boundary))
  insert(result,endmp)

  return concat(result)
end -- of drawshapes function
```

5 Running

This document only uses a few maps, a large one and some smaller. On my 2013 Dell Precision laptop processing this file gives this on the console. Observe how we use `scaled` mode. For larger maps it probably makes sense to use a `double` instance.

```
metapost > initializing instance 'metafun:1'
           using format 'metafun' and method 'default'
metapost > loading 'metafun' as 'metafun.mpxl'
           using method 'default'
metapost > initializing number mode 'scaled'
metapost > trace > This is MPLIB for LuaMetaTeX,
                   version 3.11, running in scaled mode.
metapost > trace > loading metafun for lmtx, including
                   the plain 1.004 base definitions
```

And:

```
openstreetmap > processing file 'hasselt.osm'
openstreetmap > original size 12352168 bytes,
                stripped down to 6232386 bytes
openstreetmap > 1599441 characters metapost code,
                    preprocessing time 2.433 seconds

openstreetmap > processing file 'hasselt-small.osm'
openstreetmap > original size 906573 bytes,
                stripped down to 453398 bytes
openstreetmap > 165132 characters metapost code,
                    preprocessing time 0.155 seconds

openstreetmap > processing file 'hasselt-tiny.osm'
openstreetmap > original size 7318 bytes,
                stripped down to 3790 bytes
openstreetmap > 1337 characters metapost code,
                    preprocessing time 0.000 seconds

openstreetmap > processing file 'hasselt-tiny-stripped.osm'
openstreetmap > original size 2875 bytes,
                stripped down to 2322 bytes
openstreetmap > 1111 characters metapost code,
                    preprocessing time 0.008 seconds

openstreetmap > processing file 'hasselt-church-cityhall.osm'
openstreetmap > original size 156921 bytes,
                stripped down to 123986 bytes
openstreetmap > 7601 characters metapost code,
                    preprocessing time 0.030 seconds

openstreetmap > processing file 'hasselt-bridge.osm'
openstreetmap > original size 1088541 bytes,
                stripped down to 568184 bytes
openstreetmap > 190043 characters metapost code,
                    preprocessing time 0.182 seconds
```

As you can see, we output some statistics that are not implemented in the code shown here. With standard compression, the `hasselt.osm` file, when processed standalone into `hasselt.pdf`, becomes a 951 KB file. It has quite a lot of detail so in the end that is not too bad for a file with the usual high-quality MetaPost outlines.

In the code above we had some code related to user specified colors. This is how that works:

```
\startMPpage
  draw lmt_openstreetmap [
    filename = "hasselt.osm"
  % collect  = true,
  % grid     = "dots",
  % griddot  = 1,
    used     = [
      natural   = "magenta",
      leisure   = "cyan",
      landuse   = "green",
      amenity   = false,
      boundary  = false,
      building  = false,
      ...
      aerialway = false,
    ]
  ] ;
\stopMPpage
```

Thus, you can drop objects and also force different colors. This one doesn't look pretty any more so it is not shown here. It should be clear that you have to know what objects are actually available, which is not something trivial. The commented options drive the collection in large paths and overlaying a dotted grid with a given dot size. It would not be visible here on the detailed map.

The last map (below) shows the location of the next ConTEXt meeting in Dreifelden (Germany). because that is less populated than Hasselt, we can show the grid. We use `griddot=2` here and from the log you can see that it is indeed a smaller map:

```
openstreetmap > processing file 'dreifelden.osm'
openstreetmap > original size 755190 bytes,
                stripped down to 398891 bytes
openstreetmap > 130304 characters metapost code,
                  preprocessing time 0.150 seconds
```

6 Conclusion

This started out as an experiment but as usual once you start you want to finish it. I admit that after writing the code I didn't really look at it before the 2021 ConTEXt meeting but I expect that once users are aware of this module, they might have demands. It is not hard to add features because after all it was quite trivial to implement this, at least if we forget about the guesswork and some fuzzy heuristics. But these are things that users can help with once they look at maps of places that they know well.

When wrapping up this document I decided to check how Don Knuth's university area comes out, and I was surprised to see that first of all the whole area turned red (a side effect of the area being tagged as an university amenity) but more strangely, quite a few buildings did not show up. When I looked in the file I saw lots of 'new' (hence unrecognized) tags for buildings and amenities. These two categories (tags) are used very inconsistently and in the long run I think that this should be fixed. After adding colors (and enablers) for additional building values:

`university` `barn` `bridge` `detached` `dormitory` `farm_auxiliary` `grandstand` `greenhouse` `kindergarten` `parking` `stable` `stadium` `toilets`

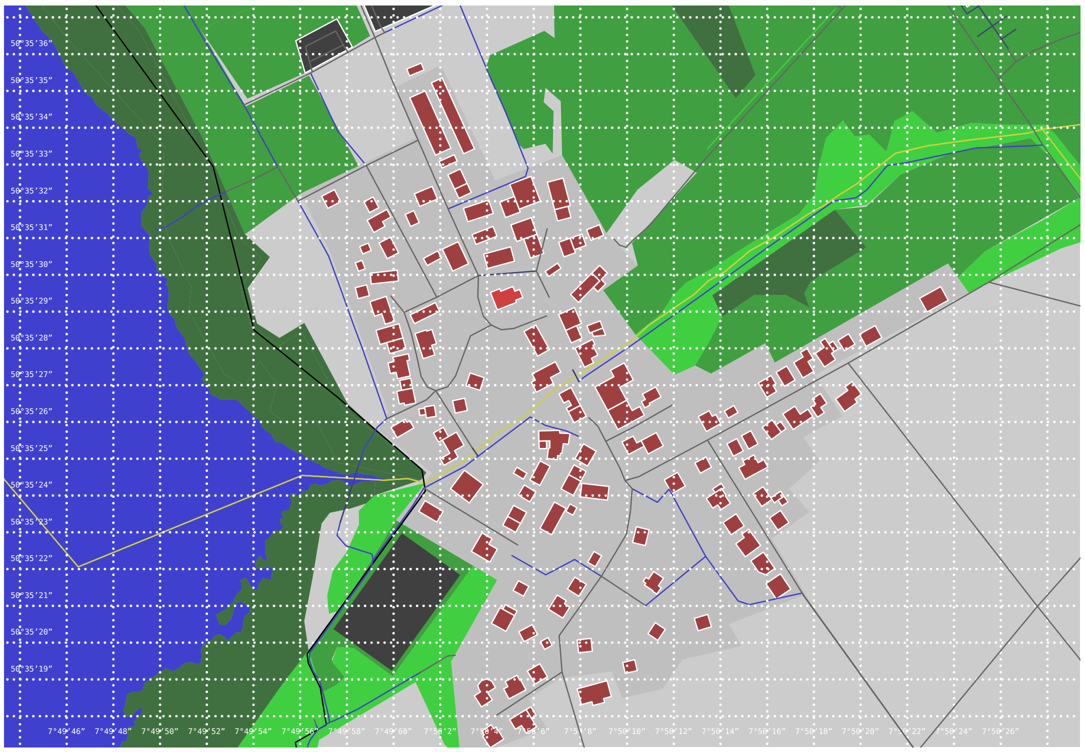
the output looked more reasonable. And, after adding a subset of the new amenities I saw

`bicycle_parking` `bicycle_repair_station` `cafe` `car_wash` `childcare` `clinic` `clubhouse` `college` `community_centre` `events_venue` `fast_food` (many :) `fire_station` `fountain` `fuel` `library` `mailroom` `pharmacy` `place_of_worship` `post_office` `recycling` `research_institute` `theatre` `wellness_centre`

and even `computer_lab` showed up, but there is plenty of work left (for potential users) to do. I probably will make some helper for identifying new tags and values.

In the end, this was one of the projects that makes working with TEX, Lua and especially MetaPost fun. It is also a good demonstration that some things are relatively easy in TEX and friends compared to typographical challenges, where one mixes all kinds of conflicting user demands and still expects perfect typeset outcomes.

◇ Hans Hagen
`http://pragma-ade.com`



Controlling captions, fullpage and doublepage floats: hvfloat

Herbert Voß

Abstract

The package hvfloat defines macros which place objects and captions of floats in different positions with different rotating angles for the object and caption. The object can fill a full column, a full page or full doublepage, with or without taking margins into account.

1 Introduction

The well-known floating environments like `figure` and `table` are easy to handle if there is only one object and one caption which fits into the current page text layout. If you want a caption rotated and beside the object (an image, tabular, ...) then you need some LaTeX knowledge or a package which does the rotation and the checking of the current page number if you want to place the rotated caption for a twocolumn document into the outer margin.

All this can be simplified by using the package `hvfloat` which has a variety of possible options for the floating object and caption. The package is loaded in the usual way:

```
\usepackage[options]{hvfloat}
```

The package has options `hyperref`, `nostfloats`, and `fbox`. The latter is only used for locating spacing problems in the document: objects and captions are framed, so unwanted whitespace can easily be seen. With `nostfloats` one can prevent the loading of the package `stfloats`, which allows bottom floats in a twocolumn document. This option is needed only in rare cases where a package conflict between `stfloats` and another package exists. With `hyperref` the package of that name is loaded.

If you would like to reset the default for the float position parameters to `htp` (here, top and page) (the default is `tbp`, top, bottom, and page), then you can load the helper package `hvfloat-fps`. It knows the optional arguments `table`, `figure`, and `all`. If you have a document with a large number of floats and relatively short text you can load the package with

```
\usepackage[all=!htb]{hvfloat-fps}
```

The exclamation allows LaTeX to ignore the internal parameter settings for the floats, e.g. the number of floats on one page [2].

Usually several LaTeX runs will be needed until `hvfloat` knows whether figures are on even or odd page and to get all the references correct. The usual warning '`Label(s) may have changed`' will be shown if another compilation is needed.

2 Dependencies

The following packages are loaded by default: `afterpage`, `caption`, `expl3`, `graphicx`, `ifoddpage`, `multido`, `picture`, `stfloats`, `subcaption`, `trimclip`, `xkeyval`.

3 The macros and optional arguments

The three main macros are `\hvFloat`, `\hvFloatSet`, and `\hvFloatSetDefaults`. The syntax for calling them is somewhat complex. Optional arguments are gray shaded:

```
\hvFloat * [options] +
    {float type}
    {floating object}
    [short caption] {long caption}
    {label}
\hvFloatSet{key=value list}
\hvFloatSetDefaults
```

The star version of `\hvFloat` is explained in section 4 on page 270 and the optional `+` is explained in section 7.2 on page 278.

The `\hvFloatSet` macro allows the global setting of parameters via the given keyword=value list, while `\hvFloatSetDefaults` sets all parameters to their default values, as shown in Table 2 on page 269.

If `\hvFloat` is given an empty second argument for *float type*, it switches by default to a nonfloat object and activates the option `onlyText` (see Table 2). The *short caption* is a second optional argument; if given, it specifies, as usual, the caption entry for the `\listof...`. All other arguments are mandatory but may be empty.

Some other macros are defined, mostly for use in the `hvfloat` implementation, but they can also be used for a user's own purposes. Only `\tabcaption` should be placed at the top of an object.

```
\figcaption [short caption] {long text}
\tabcaption [short caption] {long text}
\tabcaptionbelow [short caption] {long text}
```

They are used for the `nonFloat` keyword, where these macros write captions in the same way but outside of any float environment. The default caption cannot be used here. It is no problem to use the `\tabcaption` command to place a caption anywhere, for instance here in an inline mode:

Table 1: A caption with neither sense nor object.

In this case a label should be put inside the argument and not after the command `\tabcaption`,

doi.org/10.47397/tb/42-3/tb132voss-hvfloat

so that a reference to the nonexistent object Table 1 will still work. Source for this:

```
It is no problem to use the \verb|\tabcaption|
command ... here in an inline mode:
\tabcaption[The caption without sense ...]
  {A caption with neither sense nor object.%
   \label{dummy}}
In this case a label should be put inside the
argument ... so that a reference to the
nonexistent Table~\ref{dummy} will still work.
```

With the macro `\hvDefFloatStyle` one can define a style to be used instead of the individual setting. Internally the style is saved in a macro named `\hv@`*name*.

`\hvDefFloatStyle{`*name*`}{`*setting*`}`

The possible keywords are listed in the rotated and full page Table 2 on the next page. To make this table, we first save it in the predefined box `hvOBox` as a `tabularx` with the tabular width of the current textheight. A `tabularx` cannot be used as an argument to `\hvFloat`. This is the reason we use the intermediate box:

```
\begin{lrbox}{\hvOBox}\small
  \begin{tabularx}{\textheight}
    {@{} l>{\small\ttfamily}cX @{}}\toprule
   \emph Keyword & \emph Default ...
[...]
  \end{tabularx}
\end{lrbox}
```

Then, to typeset the table, we use the keyword `rotAngle`, which rotates object and caption together:

```
\hvFloat*[floatPos=p,rotAngle=90,
          capPos=top,capWidth=w,useOBox=true]
  {table}{}
  {The optional keywords for the
   \texttt{\textbackslash hvFloat} macro.}
  {tab:options}
```

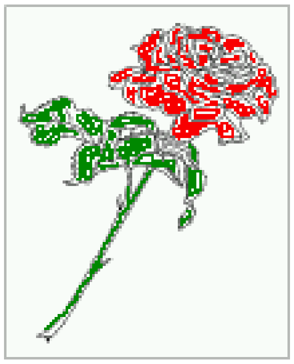
3.1 Caption positioning

By default the caption is set below the object and the macro `\hvFloat` behaves like the usual `figure` or `table` environment. With the keyword `capPos` and the value `before`, the caption can be placed beside the object. For small objects (smaller than a column/page), `before` is equivalent to `left`. Thus, here is the code for our first example:

```
\hvFloat[capPos=left]{figure}
  {\includegraphics{frose}}{A short caption
  beside a figure [...] without a label.}{}
```

If the caption is shorter than the possible width it is horizontally centered. The vertical position is by default also centered. This can be changed by the optional argument `capVPos`. The formatting can be modified by the optional arguments of the (already-loaded) package `caption`. They can be specified to `\hvFloat` via the optional argument `capFormat` (see Figure 2). The caption is also rotated by setting `capAngle=90`, which is a counter-clockwise rotation:

Figure 1: A short caption beside a figure object (`capPos=left`) without a label.



```
\hvFloat[capPos=right,
         capAngle=90,capWidth=h,
         capFormat={font=sf}]{figure}
  {\includegraphics{frose}}
  {A caption in sans [...],
   to the right [...],
   as wide as [...],
   and rotated by 90\textdegree [...]}{fig:1}
```

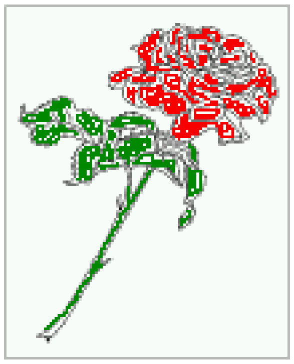


Figure 2: A caption in sans serif (capFormat={font=sf}), to the right of the object (capPos=right), as wide as the object (capWidth=h), and rotated by 90° (capAngle=90).

The caption's vertical position is controlled by the keyword `capVPos` which accepts the values `top`, `center`, and `bottom`. The `capPos=inner` setting is explained later (§5.2, p. 271). Typographically, a side caption for images should usually be at the bottom and for a table at the top of the object (Figure 3).

```
\hvFloat[capPos=inner,capVPos=bottom,
         objectAngle=180]{figure}
  {\includegraphics{frose}}
  {This caption is at the inner margin [...],
  and vertically at the bottom [...],
  and the object is rotated [...]}{fig:11}
```

3.2 The caption width

For a caption beside the object the horizontal justification is by default centered if the total width

Table 2: The optional keywords for the `\hvFloat` macro.

Keyword	*Default*	*Description*
`floatPos`	`tbp`	This is the same default placement setting as in standard LaTeX; maybe not always the best setting.
`rotAngle`	`0`	The value for the angle if both the object and the caption should be rotated together.
`capWidth`	`n`	The width of the caption. Can be `n` for a natural width given by the current linewidth, `w` for the width of the object, `h` for the height of the object, or a scale factor for `\columnwidth`.
`capAngle`	`0`	The integer value for the angle if the caption should be rotated. Positive is counter-clockwise.
`capPos`	`bottom`	The position of the caption relative to the object. Possible values: `before`: *always* before (left) from the object. `top`: *always* on top of the object. `left`: *always* before (left) from the object, but on the same page in twocolumn mode. `after`: *always* after (right) from the object. `bottom`: *always* on the bottom of the object. `right`: *always* after (right) from the object, but on the same page in twocolumn mode. `inner`: in twoside mode always typeset at the inner margin. `outer`: in twoside mode always typeset at the outer margin. `evenPage`: in twoside mode with fullpage objects always on an even page. `oddPage`: in twoside mode with fullpage objects always on an odd page.
`capVPos`	`center`	Only used when `capPos=left` | `right`; in these cases, the caption can be vertically placed at the `bottom`, `center` or `top`.
`objectPos`	`center`	Horizontal placement of the object relative to the document. Possible values are (**l**)eft, (**c**)enter, (**r**)ight.
`objectAngle`	`0`	Integer value for the angle if the object should be rotated. Positive is counter-clockwise.
`floatCapSep`	`5pt`	Additional space between the object and a left- or right-placed caption.
`useOBox`	`false`	Instead of passing the object as a parameter to `\hvFloat`, with `useOBox=true` the contents of the predefined box `\hvOBox` is used.
`onlyText`	`false`	The caption is printed as normal text with no entry in any list of . . .
`nonFloat`	`false`	The object isn't put in a floating environment, but printed as standard text with an additional caption. The float counter is increased as usual and can be referenced.
`wide`	`false`	The float can use `\textwidth` + `\marginparwidth` as horizontal width.
`objectFrame`	`false`	Put a frame with no separation around the float object.
`style`	*none*	Use a defined style.
`capFormat`	*none*	Define formatting options for `\caption`; see documentation of package `caption`.
`subcapFormat`	*none*	Define formatting options for `\subcaption`.
`fullpage`	`false`	Use a complete column in twocolumn mode.
`FullPage`	`false`	Use the full text area for the object.
`FULLPAGE`	`false`	Use the full paper width/height for the object.
`doublePage`	`false`	Use the text area on a doublepage with additional text.
`doublePAGE`	`false`	Use the text area on a doublepage without additional text.
`doubleFULLPAGE`	`false`	Use the paperwidth on a doublepage without additional text.
`vFill`	`false`	Put a `\vfill` between every two objects in a multi- or subfloat.
`sameHeight`	`false`	use the same text height on both pages for a `doublePage` object.

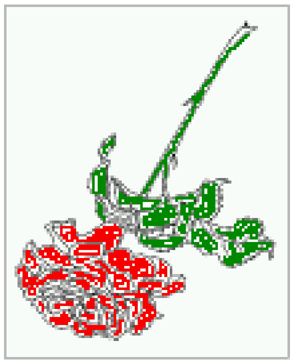
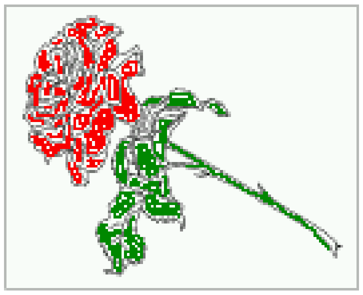


Figure 3: This caption is at the inner margin (`capPos=inner`, see p. 271), vertically at the bottom of the object (`capVPos=bottom`), and the object is rotated 180° (`objectAngle=180`).

of object and caption are less than the current column/line width. The caption width itself can be controlled by the keyword `capWidth`, which can be set to `n` (natural width), `w` (width of the object), `h` (height of the object), or a value by which to scale `\columnwidth`. Figure 2 on page 268 shows the use of `capWidth=h`, which is used for rotated captions beside the object and Figure 4 shows a caption above the object with the same width.

```
\hvFloat[capWidth=w,capPos=top,
     capAngle=180,objectAngle=90]{figure}
  {\includegraphics{frose}}
  {A 180\textdegree-rotated caption above
   [...] with the same width.}{fig:1a}
```

Figure 4: A 180°-rotated caption above a 90°-rotated object, with the same width.



4 The star version `\hvFloat*`

In `twocolumn` mode the floating environment can occupy both columns using the star version `\hvFloat*`. This is analogous to the environments `figure*` and `table*`.

If possible, the floating environment will be placed at the top of the following page or at the bottom of the current page. The latter needs the package `stfloats` which is loaded by `hvfloat` by default. (`stfloats` cannot place a float at the bottom of the first page of an article or chapter when using the core LaTeX document classes; these classes also include code that prevents placement of a float at the top of the first page.) Placing the float across both columns within the text area is not possible. Here is the code for the following example (Figure 5 on the next page):

```
\hvFloat*[capVPos=bottom,capPos=right]{figure}
 {\includegraphics{frose}
  \includegraphics[angle=180,origin=c]{frose}}
 {A caption to the right [...],
  It spans both columns [...]}{fig:2}
```

The same can be seen in Table 3 on the facing page, which also spans two columns (we'll discuss the content of that table later). Internally the number of possible floating objects on top of the page is controlled by the parameters `\topnumber` (in onecolumn mode) and `\dbltopnumber` (in twocolumn mode). They are preset for this documentclass (*TUGboat*) to 2 and 2 and differ for other document classes. For doublepage objects the values will temporarily be changed to 1.

5 Full column or fullpage objects

As mentioned in Table 2 there are three keywords for fullpage objects:

- `fullpage` for a complete column or page in a onecolumn mode,
- `FullPage` for a complete text area of a page or both columns in a twocolumn mode, and
- `FULLPAGE` for the complete paper area without leaving any margin.

This refers to the reserved space which `\hvFloat` will use when typesetting the object and caption. The object itself can be smaller than a full column or page. Package `hvfloat` defines five additional optional arguments for the package `graphicx` which can be used together with `\includegraphics` to make the code a bit shorter. They are listed in Table 3 on the next page. The so-called bind correction is additional free space at the inner margins of a twoside document.

In general, the interface is the same whether using the complete text area or the complete paper area for the floating object; the only difference is `fullpage` vs. `FULLPAGE`. By default, such a page will have no page number, no header, and no footer, and the pagestyle is `empty`.

Setting the keyword `keepaspectratio` to `false` only makes sense for images which have nearly the same ratio as the current height/width. Using a full column or page for an object implies to put the caption on the preceding or following column/page. For a twocolumn document this should *always* be the opposite column on the *same* page and for twoside documents the opposite page. Only for doublepage

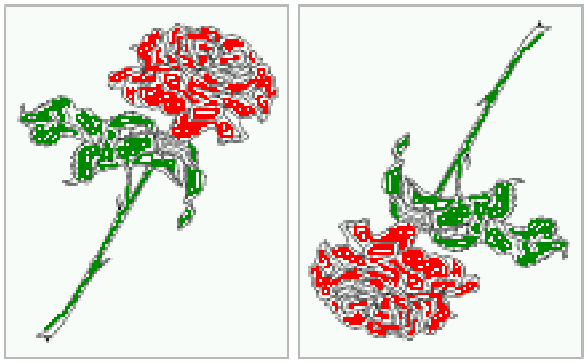


Figure 5: A longer caption to the right of the object (`capPos=right`), and vertically at the bottom of the object (`capVPos=bottom`). It spans both columns (`\hvFloat*`) and may be at the top or bottom of the page.

Table 3: Additional keywords for the `\includegraphics` macro.

name	`width=`	`height=`	`keepaspectratio=`
`fullpage`	`\columnwidth`	`\textheight`	`false`
`FullPage`	`\textwidth`	`\textheight`	`false`
`FULLPAGE`	`\paperwidth`	`\paperheight`	`false`
`doublefullPage`	`2\paperwidth-2in-2\evensidemargin`		`true`
`doubleFULLPAGE`	`2\paperwidth`	`\paperheight`	`false`
`doubleFULLPAGEbindCorr`	`2\paperwidth−2\bindCorr`	`\paperheight`	`false`

objects (left–right pages) the caption must be on the preceding or following column/page, by default at the bottom of that page or column.

A label defined via `\hvFloat` always points to the image, not to the caption. This makes no difference for the default floats, where the image and caption are on the same page. For fullpage or doublepage objects, however, the macro internally defines additional labels; one pointing to the caption (label ⟨*label*⟩`-cap`) and, if it is a doublepage object, another pointing to the second (right) part of the object (label ⟨*label*⟩`-2`).

All labels, the given one ⟨*label*⟩ and the internal ones ⟨*label*⟩`-cap` and ⟨*label*⟩`-2`, will point to the same object counter, but possibly to different page numbers. An example is shown in section 6, where Figure 13, defined with label `fig:dP`, has its caption on page 277 and its image on pages 276 and 277. The following table shows the behavior:

	`fig:dP`	`fig:dP-cap`	`fig:dP-2`
`\ref{...}`	13	13	13
`\pageref{...}`	276	277	277

5.1 Twoside and onecolumn mode

In a twoside document with onecolumn mode, a fullpage object and the corresponding caption should be on facing pages (left–right). This can be specified with the keyword `capPos` and the values `evenPage` or `oddPage`. To save space we show only the output of two example documents (Figure 6 on the following page). The upper pair of pages uses the following settings:

```
\hvFloat[fullpage, capPos=evenPage]
  {figure}
  {\includegraphics[fullpage]{frose}}
  {A caption of a \texttt{fullpage} object
   with \texttt{capPos=oddPage} ... for a
   long caption.}{fig:fullpage1}
```

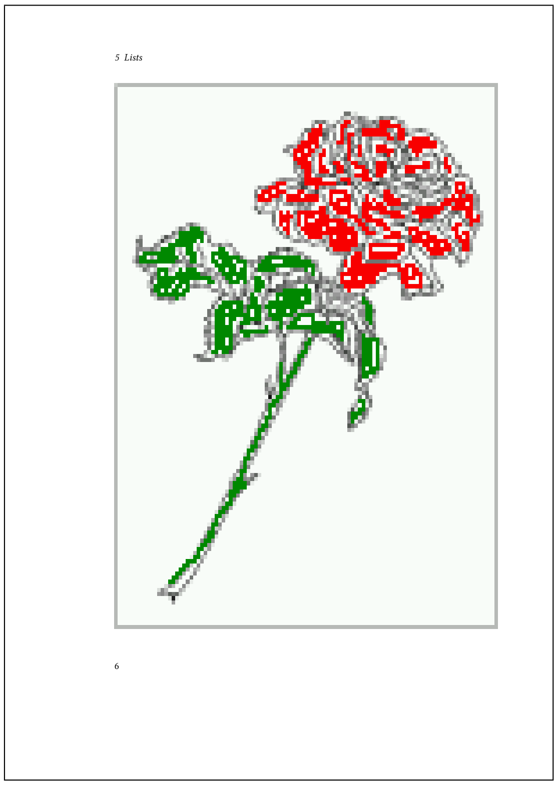
The lower two pages in Figure 6 are similar, except `capPos=evenPage` and the object is set as `FULLPAGE` instead of `fullpage`.

The captions here (and throughout) are typeset in red to make them more visible in the examples, which are often reduced in size. The complete code for all examples is on CTAN (`mirror.ctan.org/macros/latex/contrib/hvfloat/doc/examples`).

5.2 Twoside and twocolumn mode

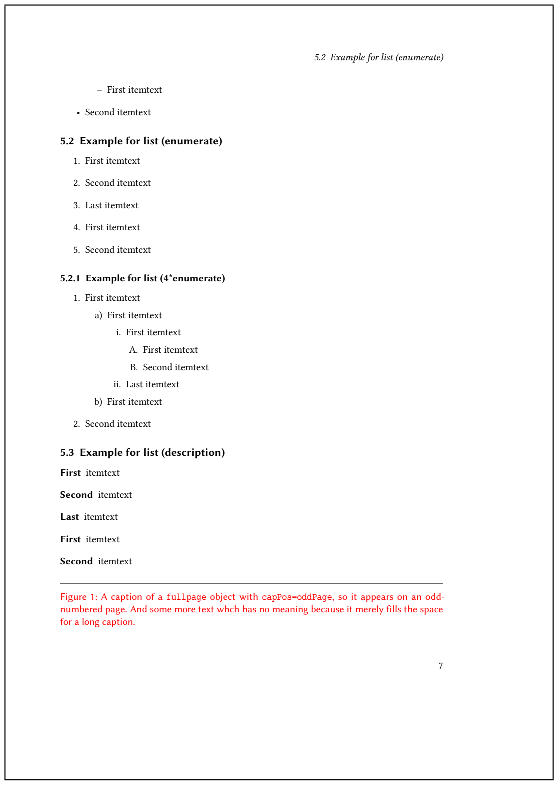
In contrast, in a twoside document in twocolumn mode, by default a caption appears *before* the fullpage or fullcolumn object, independent of an even or odd column or page. Figure 7 on page 273 shows the output of this example code:

```
\hvFloat[fullpage, capPos=inner]
  {figure}
  {\includegraphics[fullpage]{frose}}
  [A short caption for the LoF.]
  {A caption on the inner side of a twoside and
   twocolumn document (\texttt{capPos=inner}).
   This can be an even or odd page. And ...
   ... long caption.}{fig:full0}
```



5 Lists

6



5.2 Example for list (enumerate)

- – First itemtext
- • Second itemtext

5.2 Example for list (enumerate)

1. First itemtext
2. Second itemtext
3. Last itemtext
4. First itemtext
5. Second itemtext

5.2.1 Example for list (4*enumerate)

1. First itemtext
    a) First itemtext
        i. First itemtext
            A. First itemtext
            B. Second itemtext
        ii. Last itemtext
    b) First itemtext
2. Second itemtext

5.3 Example for list (description)

First itemtext

Second itemtext

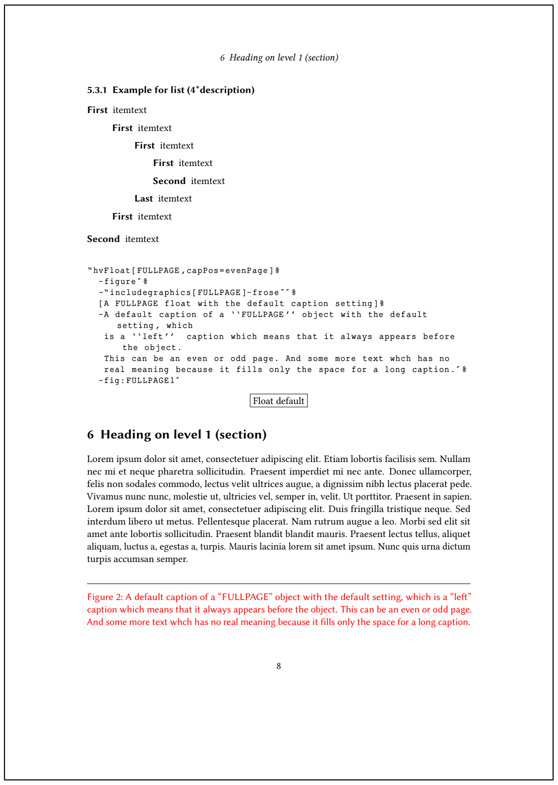
Last itemtext

First itemtext

Second itemtext

Figure 1: A caption of a `fullpage` object with `capPos=oddPage`, so it appears on an odd-numbered page. And some more text whch has no meaning because it merely fills the space for a long caption.

7



6 Heading on level 1 (section)

5.3.1 Example for list (4*description)

First itemtext
    First itemtext
        First itemtext
            First itemtext
            Second itemtext
        Last itemtext
    First itemtext

Second itemtext

```
"hvFloat[FULLPAGE,capPos=evenPage]%
  -figure"%
  -"includegraphics[FULLPAGE]-frose""%
  [A FULLPAGE float with the default caption setting]%
  -A default caption of a ``FULLPAGE'' object with the default
     setting, which
   is a ``left''  caption which means that it always appears before
      the object.
   This can be an even or odd page. And some more text whch has no
   real meaning because it fills only the space for a long caption."%
  -fig:FULLPAGE1"
```

Float default

6 Heading on level 1 (section)

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Figure 2: A default caption of a "FULLPAGE" object with the default setting, which is a "left" caption which means that it always appears before the object. This can be an even or odd page. And some more text whch has no real meaning because it fills only the space for a long caption.

8

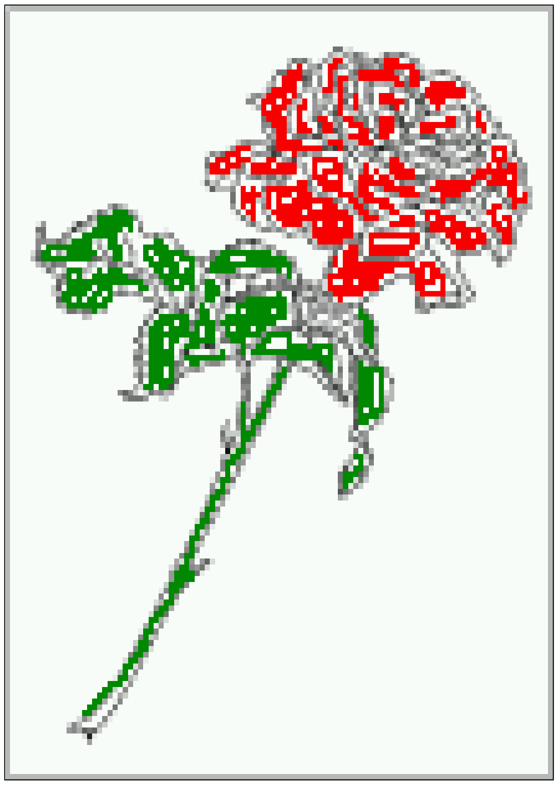


Figure 6: Twoside documents, onecolumn mode.
Top: a `fullpage` float and `capPos=oddPage` (example document `odd2s1c.tex`, pp. 6–7);
bottom: `capPos=evenPage` and a `FULLPAGE` float (example document `paper-even2s1c.tex`, pp. 8–9).

The caption is in the inner column, which is the second one for an even (left) page and the first for an odd (right) page. For a twoside document it also makes sense to have the caption on the even (left) page in the second margin and the object on the odd page (right) in the first margin. This can be achieved with the setting `capPos=inner`.

You can expect problems if you use the full column setting on a page which has full-width (double column) floats at the top. In such a case it is left to the user to modify the text structure to prevent such situations. You'll find many examples on CTAN (`https://mirror.ctan.org/macros/latex/contrib/hvfloat/doc/examples`) or in the documentation directory of your TEX distribution.

In twoside and twocolumn modes the setting `capPos=left` is different from `capPos=before`. For `capPos=before` it makes no difference on what page and column the caption appears, it will always be *before* the object. For `capPos=left` the caption will *always* be left of the object *and* on the same page! Figure 8 on the following page shows this behavior.

6 Doublepage objects

A doublepage object makes sense only for twoside documents. Then the doublepage object can be placed on facing left–right pages and the caption perhaps on the right page or, in a case where the complete paper width is used, below the right part of the image, or, if need be, on the bottom of the preceding or following page. For example: suppose a doublepage object uses the complete paper area (2`\paperwidth` × `\paperheight`) on the (left–right) pages 80–81; then the caption can be printed at the bottom of page 79 or page 82 (see Figure 12 on page 275). It is also possible to print the caption *over* the right part of the object (image) on the bottom or rotated at the right (see Figure 11 on page 275).

With the keyword `doublePage`, additional document text may appear below the doublepage object, that is, the object does not occupy the entire textheight. The other two possibilities `doublePAGE` (use the doublepage text area) and `doubleFULLPAGE` (use the doublepage paperwidth) have no additional document text on the two pages, but are still floating environments. We'll now describe these in detail.

Figure 7: A caption on the inner side of a twoside and twocolumn document (`capPos=inner`). This can be an even or odd page. And some more text with no real meaning because it merely fills the space for a long caption.

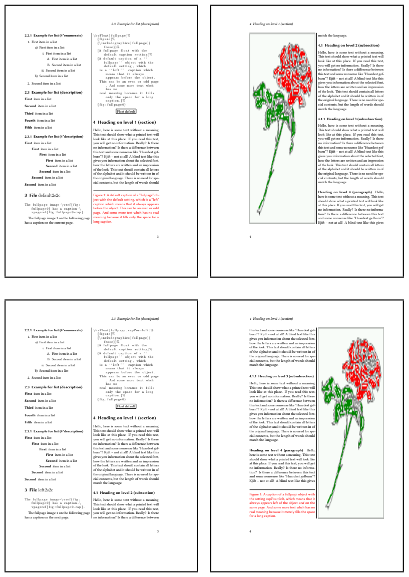


Figure 8: Twoside and twocolumn documents. Top: `capPos=before` (default); bottom: `capPos=left`. Pages 3–4 of example documents `default2s2c.tex` and `left2s2c.tex`, respectively.

6.1 Keyword `doublePage`

This is the same as putting two different floats, one each at the top of the left and right pages. The package **`hvfloat`** clips an image which would be wider than the paperwidth. Otherwise it makes no sense to use a doublepage float.

For `doublePage` the object starts at the left top of the text area and ends on the right page, depending on its width. The inner margins of the two-sided document are ignored, but a binding correction (`bindCorr`) can be set and will be taken into account. The caption will *always* be on the right page either beside, rotated or not, or below the object. For example, in Figure 13 on page 276 the caption is on the right (`capPos=right`) and rotated by 90° (`capAngle=90`). The left part of the image is on page 276, the right part on page 277 and the caption is on page 277. Incidentally, the internally-created labels described earlier were used to print this information. The label for the figure is `fig:dP`, and so the source for the previous sentence is:

```
The left part of the image
  is on page~\pageref{fig:dP},
the right part on page~\pageref{fig:dP-2} and
the caption is on page~\pageref{fig:dP-cap}.
```

A `doublePage` object allows for document text in addition to the two parts of the object. As for the caption, with `capWidth=n` and `capPos=right` the caption will be set to the right of the object with a natural width (from object to margin). This makes sense if the object is narrower than `\paperwidth` + `\textwidth`. Figure 13 on page 276 shows this, as well as (at a greatly reduced size) Figure 9. The source for Figure 13 is as follows.

```
\hvFloat[doublePage,capWidth=n,
  capPos=right,capVPos=bottom]{figure}
  {\includegraphics[width=2\textwidth]
                   {images/seiser}}
  [A short caption for the LoF]
  {A caption for a \texttt{doublePage} object,
   which will be placed on the right side of
   the right-hand part of the image. The image
   begins on the left edge of the paper [...]
   The photo was taken [...]}{fig:dP}
```

In some cases it makes sense to have some white-space, a binding correction, between the two split parts of the object. With the keyword `bindCorr` you can define a length value for the whitespace to be added both to the right of the left part and to the left of the right part (so the total whitespace added is $2 \times$ `bindCorr`).

The source for Figure 9 is the same as Figure 13, except for the addition of `bindCorr=1cm` (and the label name).

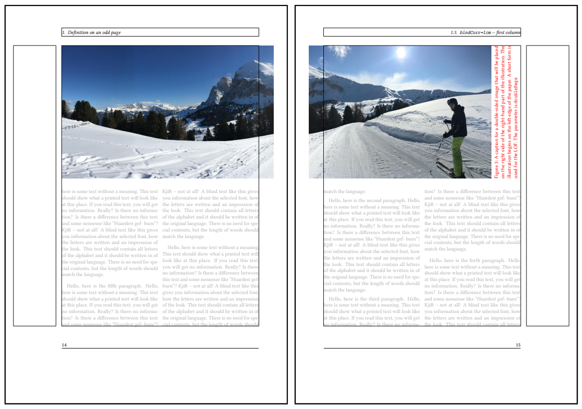


Figure 9: A `doublePage` object (the same image as Figure 13) with a binding correction of 1 cm. Pages 14–15 of example document `doublepage2s2c.tex`.

6.2 Keyword `doublePAGE`

A `doublePAGE` object appears alone on two facing pages, except for an optional caption. No additional document text will be printed on these two pages; this is the only difference between `doublePage` and `doublePAGE`. Figure 10 on the next page shows an example. The caption is below the object in the first column of the right (odd) page.

Figure 10 also shows an example of using the optional keyword `bindCorr` to specify whitespace

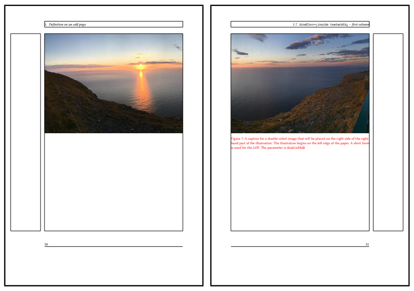


Figure 10: A `doublePAGE` image with `bindCorr` set to the inner margin. Pages 29–30 of example document `doublepage2s2c.tex`.

between the parts of the split object. In this case, we use the inner margin for the binding correction to get the two images exactly fitting the textwidth. The value for the inner margin is computed internally: `bindCorr=inner`

Here is the source for Figure 10:

```
\hvFloat[doublePAGE,capWidth=n,
         bindCorr=inner]
  {figure}
  {\includegraphics[width=2\textwidth]
                   {images/sonne-meer}}
  [A doublepage image with a caption ...]
  {A caption for a double-sided image ...
   The parameter is \texttt{doublePAGE}}
  {fig:doublePAGE3}
```

6.3 Keyword `doubleFULLPAGE`

A floating object specified with the `doubleFULLPAGE` option always starts in the upper left corner of the left (even) page. The defined text area has no meaning, it will be completely ignored for these two floating pages. The caption can be printed before, after, below, or superimposed on the object.

Table 3 on page 271 lists the corresponding two optional keywords for `\includegraphics`, namely `doubleFULLPAGE` and `doubleFULLPAGEbindCorr`, with a preset of `keepaspectratio` to false. These keywords may make code more readable but have otherwise no special meaning for any objects other than images, e.g. a tabular or something else.

The object can have any width and height but it should be at least as wide as the given `\paperwidth` and not less than 50% of the `\paperheight`. For smaller objects, use one of the other two possibilities, `doublePage` or `doublePAGE`.

The caption can be superimposed on the object or, as an alternative, printed on the bottom of the page preceding or following the doublepage (left–right) object. For a twocolumn document the keyword `twoColCaption` can be used to span both columns. This will only work for twocolumn documents which define the column mode using `\twocolumn`, such as the present *TUGboat* document class. The `multicol` package is *not* supported.

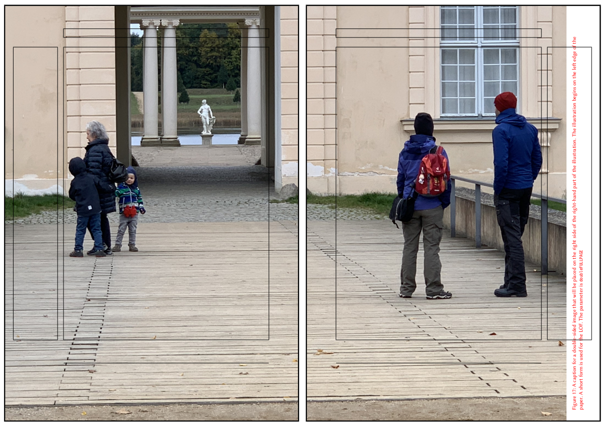


Figure 11: A `doubleFULLPAGE` object with `capPos=right`, so the caption appears on the right page. Pages 72–73 of example document `doubleFULLPAGE2s2c.tex`.

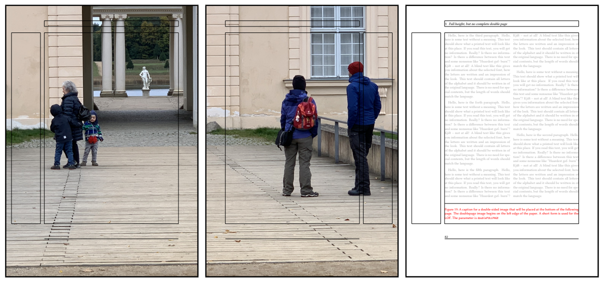


Figure 12: A `doubleFULLPAGE` object with `capPos=after`, so the caption is on the following page. Pages 80–82 of example document `doubleFULLPAGE2s2c.tex`.

Figure 11 shows two pages with an image spread across the double page which is small enough to get a rotated caption on the right of the page which, for our demonstration, is printed in red as usual. The page layout is also printed as frames, which makes it easier to understand and choose values for the full page mode. These frames are shown by loading the package `showframe`.

The code for Figure 11 is:

```
\hvFloat[doubleFULLPAGE,capPos=right]{figure}
  {\includegraphics[height=\paperheight]
                   {images/rheinsberg}}
  [A doublepage image ...]
  {A caption for a double-sided image ...
   The parameter is \texttt{doubleFULLPAGE}}
  {fig:doubleFULLPAGEOn}
```

If the image has nearly the same ratio as the current `\paperwidth` / `\paperheight`, then a caption can reasonably appear at the bottom of the following page. This is specified with `capPos=after`; Figure 12 on the previous page shows the result. Similarly, `capPos=before` would put the caption on the preceding page.

Here is the code for Figure 12, specifying the option `doubleFULLPAGE` option to both `\hvFloat` and `\includegraphics`:

```
\hvFloat[doubleFULLPAGE,capPos=after,
          twoColumnCaption]{figure}
  {\includegraphics[doubleFULLPAGE]
                    {rheinsberg}}
  {A caption for a double-sided image ...
   The parameter is \texttt{doubleFULLPAGE}}
  {fig:doubleFULLPAGE02ndnn}
```

7 Subfloats and multifloats

A floating environment can have any content except another floating environment. The only requirement for the content is that it must be smaller than one page spread. The content itself can be any combination of text, equations, tabulars, and/or images. We call it a *subfloat* if the content has *one* main caption and several subcaptions for any object. We call it a *multifloat* if the content has no main caption of its own, but the objects have their own captions.

Table 4 gives the two keywords, `subFloat` and `multiFloat`, which introduce such special content. They can be placed as a default floating environment, full column, full page, or full doublepage.

Table 4: Keywords `subFloat` and `multiFloat` for multiple objects in a float.

Name	*Description*
`subFloat`	For multiple objects with one main caption and several subcaptions.
`multiFloat`	For multiple objects, each with its own caption.

The syntax for the macro which defines such sub- or multifloats is somewhat complex. Only the keyword defines whether the float is a multifloat or subfloat; the syntax of the macro shows no difference. With the optional argument `vFill` the objects in a column (two column) or a page (one column) are stretched over the given height `\textheight`. The default is no stretching so that extra whitespace appears at the bottom of the column/page.

Figure 13: A caption for a doublePage object, which will be placed on the right side of the right-hand part of the image. The image begins on the left edge of the paper. A short form can be used for the LoF. The photo was taken in the Italian Alps at the Alpi di Siusi (Seiser Alm).

7.1 Subfloats

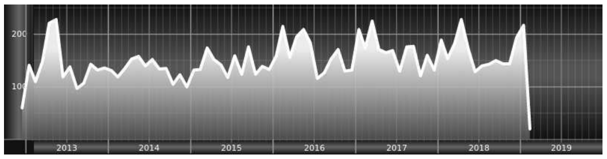
A subfloat page can have only one type of object which will have one main caption and individual subcaptions. (For completeness: If you define no subcaption then it does not matter what kind of object we have.) The syntax for subfloats and multifloats is similar, but some arguments are ignored for a subfloat, so can be left empty. The first line defines only the floating type and the main caption, the object entry is ignored! All additional lines will have the same float type; this is why the float type entry is ignored.

```
\hvFloat [subFloat,...]
  +{float type}{}% the main type
      [short caption] {long caption}{label}
  +{}{floating object}% a subobject
      [short caption] {long caption}{label}
  ⋮ ...
  +{}{floating object}% another subobject
      [short caption] {long caption}{label}
```

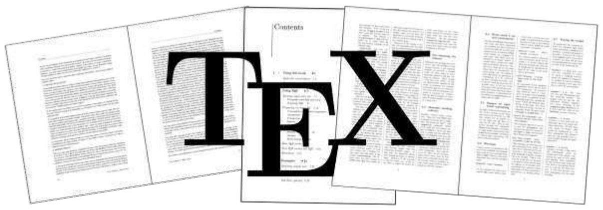
The + symbol defines an additional object which will be part of the same floating environment. It's up to the user to be sure that one page or one column can hold all defined objects.

The code for Figure 14 on the next page, which comprises the subfigures 14a to 14e, is as follows:

```
\hvFloat[subFloat,vFill,fullpage,capPos=after]
  +{figure}{}
   [Short caption of the subfloat]
   {The main caption of a fullpage subfloat,
    which appears in the left or right column.
    This can be an even or odd page.
    The \texttt{vFill} option is set,
    so vertical space is distributed between
    the subobjects.}
   {sub:demo}
  +{}{\includegraphics[columnWidth]{CTAN}}
   [Short caption A]
   {Subcaption A of a fullpage subobject.}
   {sub:demo0}
  +{}{\includegraphics[columnWidth]{CTAN1}}
   {Subcaption B of a fullpage subobject,
    a little longer for no particular reason.}
   {sub:demo1}
  +{}{\includegraphics[columnWidth]{CTAN2}}
   {Subcaption C of a fullpage subobject.}
   {sub:demo2}
  +{}{\includegraphics[columnWidth]{CTAN3}}
   {Subcaption D of a fullpage subobject.}
   {sub:demo3}
```

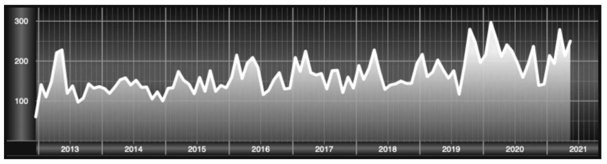


(a) Subcaption A of a fullpage subobject.



(b) Subcaption B of a fullpage subobject, a little longer for no particular reason.

(c) Subcaption C of a fullpage subobject.



(d) Subcaption D of a fullpage subobject.

(e) The last subcaption E of a fullpage subfloat object, which has subcaptions 14a–14e, and the main caption is beside (to the right of) this full column object.

```
  +{}{\includegraphics[trim=0 1.5cm 0 5mm,clip,
      columnWidth]{TUGboat}}
   {The last subcaption E of a fullpage
    subfloat object, which has subcaptions
    \ref{sub:demo0}--\ref{sub:demo5}, and the
    main caption is beside (to the right of)
    this full column object.}
   {sub:demo5}
```

The keyword `subFloat` defines the following images or tabulars as subfloats. The keyword `figure` in the second line of the code defines the main type of the floating environment; all subobjects must be of the same type. This is the reason why all following arguments are empty: `+{}{....`

The package `subcaption` is loaded by default and is usually activated with `\captionsetup[sub][singlelinecheck]`.

The main label of the subfloat is `sub:demo`, which points to the object column on page 278. In this case the internal label `sub:demo-cap` points to the same page 278, because object and caption are in different columns but on the same page. Both refer to the same object: `\ref{sub:demo}` → 14 and `\ref{sub:demo-cap}` → 14.

7.2 Multifloats

With a `multiFloat` object, no main caption is given. Every object gets its own caption, which is the reason that figures, tabulars, etc., can be mixed. All individual captions are listed before or after the full column/page, at the bottom of the column/page (see example on the facing page).

```
\hvFloat [multiFloat,...]
  +{float type}{floating object}
      [short caption] {long caption}{label}
  +{float type}{floating object}
      [short caption] {long caption}{label}
  ⋮ ...
  +{float type}{floating object}
      [short caption] {long caption}{label}
```

The + symbol defines an additional object which will be part of the same floating environment. For a multifloat object all parameters are valid. It's up to the user to be sure that one page or one column can hold all defined objects.

Figure 14: The main caption of a fullpage subfloat, which appears in the left or right column. This can be an even or odd page. The `vFill` option is set, so vertical space is distributed between the subobjects.

The captions of Figures 15–18 and of Tables 5 and 6 are on page 279, and all objects also appear on the same page. All of these figures and tables are part of the same multifloat. Here is the code of the multifloat example:

```
\captionsetup{singlelinecheck=false}
\hvFloat[multiFloat,vFill,
         fullpage,capPos=before]
  +{figure}
  {\includegraphics[columnWidth]{dove}}
   [Short caption A]
   {Caption A of a fullpage multifloat object,
    which follows in the left or right column.
    This can be an even or odd page. And some
    more text with no real meaning because it
    merely fills the space for a long caption.}
   {img:demo}
  +{table}{\begin{tabular}{lrcp{3cm}}\hline
    Left & Right & Centered & Parbox\\\hline
    L    & R     & C          & P\\
    left & right & center     & Text
                    with possible line breaks\\
    L    & R     & C          & P\\
    left & right & center     & Text
                    with possible line breaks\\
    \multicolumn{4}{c}{Centered multicolumn
                      over all columns}\\\hline
           \end{tabular}}
   [Short example caption B1]
   {Caption B of a fullpage object, a tabular
     in this case.}{tab:demo}
  +{figure}
   {\includegraphics[columnWidth]{CTAN1}}
   {Caption C of a fullpage object.}
   {img:demo1}
  +{figure}
   {\includegraphics[columnWidth]{CTAN2}}
   {Caption D of a fullpage object.}
   {img:demo2}
```

Figure 15: Caption A of a fullpage multifloat object, which follows in the left or right column. This can be an even or odd page. And some more text with no real meaning because it merely fills the space for a long caption.

Table 5: Caption B of a fullpage object, a tabular in this case.

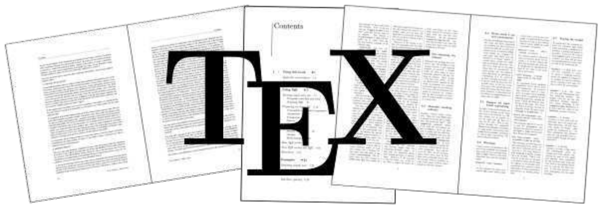
Figure 16: Caption C of a fullpage object.

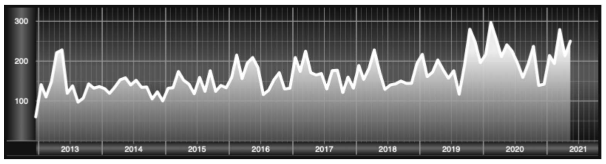
Figure 17: Caption D of a fullpage object.

Figure 18: Caption E of a fullpage object.

Table 6: Caption B2 of a fullpage object, another tabular repeating Table 5.

Left	Right	Centered	Parbox
L	R	C	P
left	right	center	Text with possible line breaks
L	R	C	P
left	right	center	Text with possible line breaks
Centered multicolumn over all columns			





Left	Right	Centered	Parbox
L	R	C	P
left	right	center	Text with possible line breaks
L	R	C	P
left	right	center	Text with possible line breaks
Centered multicolumn over all columns			

```
  +{figure}
   {\includegraphics[columnWidth]{CTAN3}}
   {Caption E of a fullpage object.}
   {img:demo3}
  +{table}{\begin{tabular}{lrcp{3cm}}\hline
    Left & Right & Centered & Parbox\\\hline
    L    & R     & C        & P\\
    left & right & center   & Text
                    with possible line breaks\\
    L    & R     & C        & P\\
    left & right & center   & Text
                    with possible line breaks\\
    \multicolumn{4}{c}{Centered multicolumn
                      over all columns}\\\hline
           \end{tabular}}
   [Short example caption B2]
   {Caption B2 of a fullpage object, another
    tabular repeating Table~\ref{tab:demo}.}
   {tab:demo2}
```

8 Splitting tables across two pages

By default a table can only be split in the vertical direction, as a so-called `longtable`. Large tables can be rotated on a page (see Table 2 on page 269), but splitting it automatically in the horizontal direction is not supported by core LaTeX.

However, saving a table without page breaks into a box is no problem and such a box can be handled like an image, which is also like a box. The only problem is that the table must be split horizontally between two columns, as a split column may likely be unreadable.

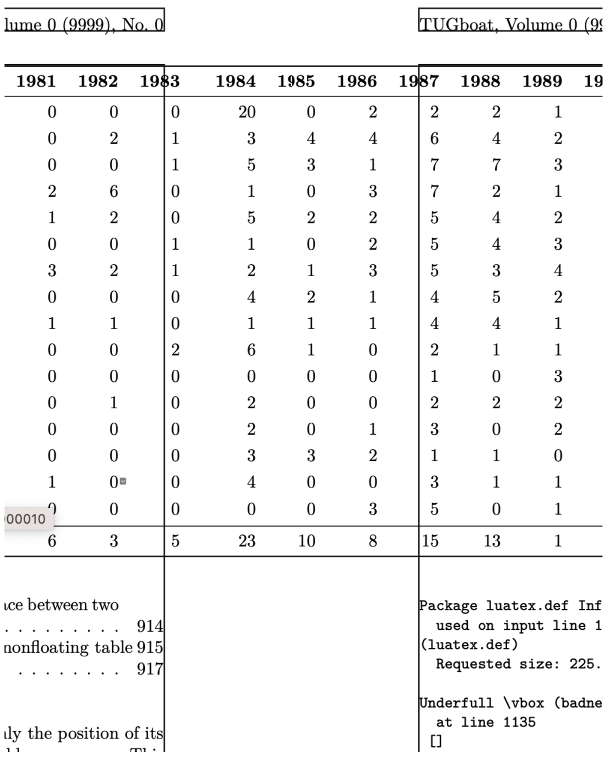
The package `hvfloat` provides the box `\hvOBox` for public use. We can save a table into this box:

```
\savebox\hvOBox{%
  \begin{tabular}{l @{} *{18}r}
    ... the table ...
  \end{tabular}}
```

and then use it in the same way as a doublepage image, with the table split in two pieces. If the split occurs at an unfavorable point in the table, e.g. in the middle of a column, then insert some horizontal space between the two columns with `@{\hspace{...}}`. For example (the output is shown in Table 7):

```
\begin{tabular}{lll@{\hspace{1cm}}ll}\hline
1 & 2 & 3 & 4 & 5 \\
1 & 2 & 3 & 4 & 5 \\
1 & 2 & 3 & 4 & 5 \\
1 & 2 & 3 & 4 & 5 \\\hline
\end{tabular}
```

Figure 19 shows how the table looks in the middle of the doublepage (the text shown at the bottom of the page is just filler). The column with *1985* will be cut and not readable. There are two solutions to split the table at a better position: insert some space



lume 0 (9999), No. 0

TUGboat, Volume 0 (9

1981	1982	1983	1984	1985	1986	1987	1988	1989	19
0	0	0	20	0	2	2	2	1	
0	2	1	3	4	4	6	4	2	
0	0	1	5	3	1	7	7	3	
2	6	0	1	0	3	7	2	1	
1	2	0	5	2	2	5	4	2	
0	0	1	1	0	2	5	4	3	
3	2	1	2	1	3	5	3	4	
0	0	0	4	2	1	4	5	2	
1	1	0	1	1	1	4	4	1	
0	0	2	6	1	0	2	1	1	
0	0	0	0	0	0	1	0	3	
0	1	0	2	0	0	2	2	2	
0	0	0	2	0	1	3	0	2	
0	0	0	3	3	2	1	1	0	
1	0	0	4	0	0	3	1	1	
0	0	0	0	0	3	5	0	1	
6	3	5	23	10	8	15	13	1	

ace between two
. . . . 914
nonfloating table 915
. . . . 917

ly the position of its

```
Package luatex.def Inf
  used on input line 1
(luatex.def)
  Requested size: 225.

Underfull \vbox (badne
  at line 1135
 []
```

Figure 19: The table column *1985* appears between the two pages and would not be readable.

Table 7: Adding space between two columns

1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5

before this column, or use the `bindCorr` keyword to insert a binding correction space. For Table 8 on page 282 both possibilities are used. Inserting more space:

```
\begin{tabular}{l @{} *{13}r @{\quad}*8r}
```

and using 8 mm for the binding correction (shown below) which was found by trial and error.

The code for the split table on a double page is:

```
\hvFloat[doublePage,capWidth=n,capPos=right,
   capVPos=bottom,useOBox,% use the defined box
   bindCorr=8mm]
  {table}
  {}% no need for an object
  [A doublepage tabular.]
  {A caption for a doublePage tabular that
   will be placed on the right side of the
   right-hand part of the tabular. The table
   begins on the left edge of the text area
   of the left page. The additional space
between the columns 1984 and 1985 is
\texttt{\textbackslash quad}, which is the
same as 1\,em. The binding correction is
set to 8\,mm, which gives additional
whitespace of 16\,mm.}{tab:dP}
```

and the output is Table 8 on pages 282 and 283. It depends on the way the document is printed whether more or less space between the two pages makes sense.

9 Todo list

The macro `\hvFloat` only checks the position of its definition if it is defined on an odd or even page. This is done with the help of the macro `\checkoddpage` from the package `ifoddpage`. Together with the internal LaTeX macro `\if@firstcolumn` it knows exactly the position of its definition in the source of the document: left or right page, first or second column. But it doesn't know if the current page is completely empty, which is the case if `\hvFloat` is the first command on a new page. If this is also an even page, then a doublepage object can be placed immediately. But the current code always uses the *next* even–odd page combination. In a future release there should be a test like `\if@newpage`.

More checks for the correct use of the parameters would be useful. For example: if one uses the keyword `doubleFULLPAGE` with an object which is narrower than `\textwidth`, then the output will be rubbish.

The optional argument `wide` as shown in Figure 21 on the next page works only in oneside mode if you also use twocolumn mode (see Figure 20). For twoside mode we have different margins for a possible wide float in the first or the second column; this is not recognized by `hvfloat`. However, if you need wide floats in a twoside and twocolumn mode you can move the macro `\hvFloat` to places in the source where the output is always in the outer column, which uses the marginpar width. Using the argument `nonFloat`, as shown in Figure 20, the float appears exactly at the place of the definition.

In some cases the option `useOBox` for a predefined savebox `\hvOBox` does not work. One can use instead `{\usebox\hvOBox}` as the argument for the object, which has the same effect. However, the box `\hvOBox` must have valid contents, and be set before it is used.

10 Conclusion

The package `hvfloat` should work with all kinds of documents, oneside in one- or twocolumn mode, twoside in one- or twocolumn mode. It is much easier to place doublepage objects in a onecolumn document than a twocolumn document. Internally, LaTeX puts two single pages together to one page with two columns. Only the optional header and footer are printed across these "two" pages.

The package `hvfloat` makes intensive use of the macro `\afterpage` [1]. If one defines a doublepage object in the first column of a left (even) page, `\hvFloat` needs *three* nested `\afterpage` commands, one for each column, to let an object or a caption start on the next left (even) page. Until LaTeX reaches this page for the object/caption, nearly two pages have to be filled with text or other objects which are defined after the macro `\hvFloat`. Especially in twocolumn mode you can expect problems, if you have too little text, images, tables or other simple objects to fill up these two pages until the doublepage object will be set. Such problems can only be solved by adding some text or moving the macro `\hvFloat` to another column of the document.

Just as with the standard floating environments `figure` and `table`, it is left to the user to ensure that the contents of the environment fit the page. If an object is wider than $2 \times$ `\paperwidth` or higher than `\paperheight` it cannot be placed on a doublepage and the output may be useless.

Before using a doublepage for an object, one should test if it might be sufficient to use the margin for additional space. `\hvFloat` knows the optional argument `wide` which allows using the space of `\marginparwidth`. The caption can be placed in the usual way, above/below or left/right relative to the object. The use of the inner/outer position for twoside documents is also possible.

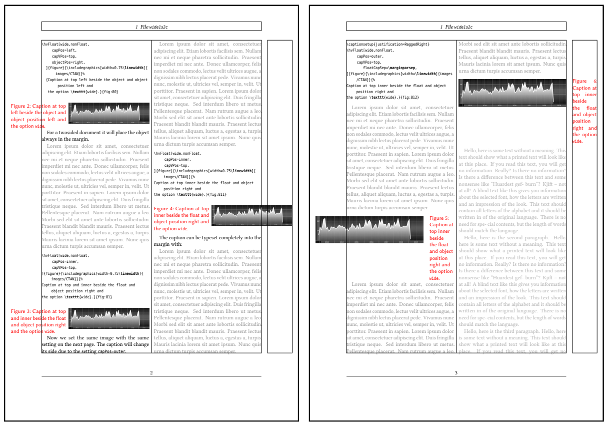
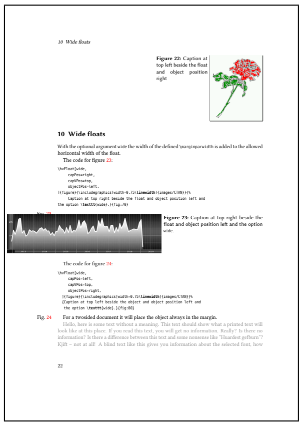


Figure 20: Pages 2–3 of example document `wide1s2c.tex`, oneside with twocolumn and the `wide` option.

	1972	**1973**	**1974**	**1975**	**1976**	**1977**	**1978**	**1979**	**1980**	**1981**	**1982**	**1983**	**1984**
Line No 1	1	3	1	1	1	0	1	1	0	0	0	0	20
Line No 2	1	1	3	1	0	0	0	0	0	0	2	1	3
Line No 3	2	1	2	1	0	0	0	0	0	0	0	1	5
Line No 4	1	0	5	1	2	0	0	0	0	2	6	0	1
Line No 6	2	1	1	0	0	0	0	0	0	1	2	0	5
Line No 5	0	0	4	2	1	2	2	1	0	0	0	1	1
Line No 8	0	1	1	0	0	0	1	1	0	3	2	1	2
Line No 9	0	0	0	0	0	1	2	1	0	0	0	0	4
Line No10	0	1	3	0	1	0	1	0	0	1	1	0	1
Line No11	0	2	2	1	1	0	1	0	0	0	0	2	6
Line No12	2	0	2	4	1	0	4	0	0	0	0	0	0
xyz	2	3	0	0	0	0	0	0	0	0	1	0	2
Line No13	0	1	0	0	1	0	3	0	0	0	0	0	2
Line No14	0	1	0	0	0	0	0	0	0	0	0	0	3
Line No15	0	0	0	0	0	0	0	0	0	1	0	0	4
Line No16	0	0	0	0	0	1	0	0	0	0	0	0	0
Some numbers	2	6	13	8	4	3	5	4	0	6	3	5	23



10 Wide floats

Figure 22: Caption at top left beside the float and object position right

10 Wide floats

With the optional argument `wide` the width of the defined `\marginparwidth` is added to the allowed horizontal width of the float.

The code for figure 23:

```
\hvFloat[wide,
     capPos=right,
     capVPos=top,
     objectPos=left,
]{figure}{\includegraphics[width=0.75\linewidth]{images/CTAN}}{%
     Caption at top right beside the float and object position left and
the option \texttt{wide}.}{fig:70}
```

Fig. 23

Figure 23: Caption at top right beside the float and object position left and the option `wide`.

The code for figure 24:

```
\hvFloat[wide,
     capPos=left,
     capVPos=top,
     objectPos=right,
  ]{figure}{\includegraphics[width=0.75\linewidth]{images/CTAN}}%
  {Caption at top left beside the object and object position left and
   the option \texttt{wide}.}{fig:80}
```

Fig. 24 For a twosided document it will place the object always in the margin.

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how

22

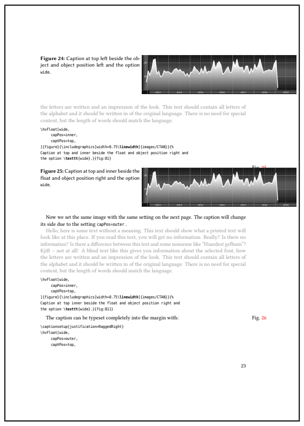


Figure 24: Caption at top left beside the object and object position left and the option `wide`.

the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

```
\hvFloat[wide,
     capPos=inner,
     capVPos=top,
]{figure}{\includegraphics[width=0.75\linewidth]{images/CTAN}}{%
Caption at top and inner beside the float and object position right and
the option \texttt{wide}.}{fig:81}
```

Fig. 25

Figure 25: Caption at top and inner beside the float and object position right and the option `wide`.

Now we set the same image with the same setting on the next page. The caption will change its side due to the setting `capPos=outer` .

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

```
\hvFloat[wide,
     capPos=inner,
     capVPos=top,
]{figure}{\includegraphics[width=0.75\linewidth]{images/CTAN}}{%
Caption at top inner beside the float and object position right and
the option \texttt{wide}.}{fig:811}
```

The caption can be typeset completely into the margin with: Fig. 26

```
\captionsetup{justification=RaggedRight}
\hvFloat[wide,
     capPos=outer,
     capVPos=top,
```

23

Figure 21: Pages 22–23 of the package documentation, showing examples using optional argument `wide` to use the margin space.

Figure 21 shows some examples of using the margin for a onecolumn document with the following use of `\hvFloat`.

```
\hvFloat[wide,capPos=inner,capVPos=top]{figure}
 {\includegraphics[width=0.75\linewidth]
   {images/CTAN}}
 {Caption at top inner beside the float ...
  and the option \texttt{wide}.}{fig:wide}
```

The list of figures and list of tables are not affected by package `hvfloat` and should work as usual. For example, here is the list of tables for this article:

List of Tables

1 A caption with no object 267
2 The optional keywords for the `\hvFloat` macro 269
3 Additional keywords for the `\includegraphics` macro 271
4 Keywords `subFloat` and `multiFloat` for multiple objects in a float 276
5 Short example caption B1 279
6 Short example caption B2 279
7 Adding space between two columns . 280
8 A doublepage tabular 283
9 A short nonfloating table 283

Another feature is that simple non-floating objects can be placed by the environment `hvFloatEnv`, which has only one optional argument, giving the horizontal width. For the caption one has to use the macro `\captionof{`*type*`}{...}` or the (usually internal) macro `\tabcaption{...}` mentioned on page 267:

```
\begin{hvFloatEnv}[0.5\columnwidth]
\centering\captionof{table}
  {A short nonfloating table.}
  \label{tab:nonfloat}
\begin{tabular}{@{} l c r @{}}\hline
left & center & right \\
L    & C      &  R    \\\hline
\end{tabular}
\end{hvFloatEnv}
```

1985	**1986**	**1987**	**1988**	**1989**	**1990**	**1991**	**1992**
0	2	2	2	1	2	1	0
4	4	6	4	2	2	1	0
3	1	7	7	3	2	1	0
0	3	7	2	1	2	1	0
2	2	5	4	2	2	1	0
0	2	5	4	3	2	1	0
1	3	5	3	4	2	1	0
2	1	4	5	2	2	1	0
1	1	4	4	1	2	1	0
1	0	2	1	1	2	6	0
0	0	1	0	3	2	6	0
0	0	2	2	2	2	6	0
0	1	3	0	2	2	6	0
3	2	1	1	0	2	6	0
0	0	3	1	1	2	6	0
0	3	5	0	1	2	6	1
10	8	15	13	1	32	51	1

Table 8: A caption for a doublePage tabular that will be placed on the right side of the right-hand part of the tabular. The table begins on the left edge of the text area of the left page. The additional space between the columns 1984 and 1985 is `\quad`, which is the same as 1 em. The binding correction is set to 8 mm, which gives additional whitespace of 16 mm.

Table 9: A short nonfloating table.

left	center	right
L	C	R

But pay attention to references if floating and non-floating environments are mixed on one page; they can point to wrong numbers. Moving the floating environment to another place in the document is one workaround for such a problem. Alternatively, using only floating environments is preferred, if your document is mainly text, with only some figures and/or tables.

References

[1] D. Carlisle, The LaTeX Team. The afterpage package, version 1.08, 2014-10-28. Execute command after the next page break. `https://ctan.org/pkg/afterpage`

[2] F. Mittelbach, M. Goossens, et al. *The LaTeX Companion*. Pearson Education, 2nd ed., 2004.

[3] M. Scharrer. The adjustbox package, version 1.3, 2020-08-19. Graphics package-alike macros for "general" boxes. `https://ctan.org/pkg/adjustbox`

[4] M. Scharrer. The ifoddpage package, version 1.1, 2016-04-23. Determine if the current page is odd or even. `https://ctan.org/pkg/ifoddpage`

[5] A. Sommerfeldt. The caption package, 2020-10-26. Customising captions in floating environments. `https://ctan.org/pkg/caption`

[6] S. Tolušis. The stfloats package, version 3.3, 2017-03-27. Commands to control the presentation of floats. `https://ctan.org/pkg/stfloats`

[7] H. Voß. The hvfloat package, version 2.34, 2021-09-23. Rotating and controlling float captions and objects. `https://ctan.org/pkg/hvfloat`

◇ Herbert Voß
Wasgenstraße 21
14129 Berlin, Germany
`herbert (at) dante dot de`
`https://hvoss.org/`

Preventing tofu with pdfTeX and Unicode engines

Frank Mittelbach

Abstract

Discussion of input encodings vs. font encodings, missing characters, Unicode, and TeX history.

1 With tofu through the years

Tofu is not just an essential ingredient for many Asian dishes, it is also the nickname for the little squares produced by many browsers when they are asked to render a character for which they do not have a glyph available.

Especially in the early days of the World Wide Web, websites in foreign languages (from the perspective of your computer) got often littered with such squares, making text comprehension quite difficult if not impossible in some cases. So instead of getting

> ¿But aren't Kafka's Schloß & Æsop's Œuvres often naïve vis-à-vis the dæmonic *phœnix's official rôle* in fluffy soufflés?

you might have seen something like

> □But aren't Kafka's Schlo□ □ □sop's □uvres often na□ve vis-□-vis the d□monic *ph□nix's official r□le* in fluffy souffl□s?

Over the years the situation with browsers improved (partly because using inferior fonts was deemed acceptable as long as they could render the needed glyphs), but even nowadays you may find tofu-littered sites, or perhaps worse, those where your browser thinks it can show you the glyphs but renders the wrong ones.

While with browsers you may accept a certain imperfection in the rendering, tofu in printed material is quite unacceptable. Typesetting systems should always use the correct glyphs or at least tell you very explicitly if they are unable to do so for some reason, to allow you to apply some corrective actions. In the remainder of this article we will discuss how TeX and in particular LaTeX is doing in this respect and what a user can or must do to avoid such a capital blunder.

Early vegetarian dishes with TeX

In the early days of TeX the use of fonts was easy because you could use any font you wanted as long as it was called Computer Modern.

In other words there was essentially only one set of fonts available for use with TeX and the glyphs it contained and how to address them was described in *The TeXbook* [3]. Furthermore, all fonts only contained 128 glyphs, i.e., essentially the base Latin characters, a few accents to construct diacritical characters using the `\accent` primitive and a few other symbols such as †, $ and so forth to be accessed through command names.

Thus, once you learned the construction methods and memorized the control sequences for accessing the existing symbols you could be sure that the characters you used would faithfully appear in the printed result. Of course, part of the reason for this was the limited glyph set; already any Latin-based language other than English posed serious issues, namely that necessary glyphs were missing entirely, or only available as constructed characters (whenever accents where involved) — which prevented TeX from applying hyphenation.

So as TeX got more popular outside the English-speaking world there was considerable pressure on Don Knuth (largely by European users, the author among them) to extend TeX so that it could better handle languages with larger character sets. At the 1989 TeX conference in Stanford we finally managed to convince Don to reopen (in a limited way) TeX development and produce TeX 3. This version of TeX was then able to deal with more than one language within a document (e.g., use multiple hyphenation patterns) and support 8-bit input and output (that is, 256 characters in a font).

While this enabled the use of different input code pages for different character sets, as was standard in those days, and also solved the problem of hyphenating words containing accented characters (by using fonts with precomposed glyphs), it also posed new challenges.

Depending on the active code page when writing a document, a given keyboard character might be associated with a different number (between 0 and 255) and that number had to be mapped to the right slot in a font to produce the glyph that was originally intended. So the days of input number equals font glyph position were definitely over, and the TeX world had to come up with a more elaborate scheme to translate one into the other to avoid missing or wrong characters in the output.

The LaTeX 2$_\varepsilon$ solution

For LaTeX the solution came in the form of the New Font Selection Scheme [4], and in particular with the packages `inputenc` (for managing input in different code pages and mapping it to a standard internal representation) and `fontenc` (for translating this internal representation to the correct glyph positions in different fonts).

Introducing font encodings

LaTeX classified the font encodings and gave them names such as `OT1`, `T1`, `TS1`, `T2A`, `T2B`, etc. Each such font encoding defined which glyphs are in a font using that encoding and to which character code (again, 0–255) each glyph was assigned in the font. Thus, if you had two different fonts with the same encoding you could exchange one for the other and still be one hundred percent sure[1] that your document typeset correctly, with no missing or incorrect glyphs in the output.

In practice only a small number of font encodings ever got used and new fonts usually were made available in these "popular" encodings by providing the necessary font re-encodings through the virtual font mechanism, or through re-encodings done by device drivers (such as `dvips`) or directly in the engine (in the case of `pdftex`).

As an overall result, life for LaTeX users was again fairly easy after 1994 and remained this way well into this century, because by simply specifying which font encoding to use, documents would normally be typeset without defects, regardless of the font family that got used. Further, due to the fact that for users writing in Latin-based languages essentially every interesting font available was provided in the `T1` encoding, it was also clear which glyphs were available and those were available almost universally.

Pitfalls with missing input encodings

There was still the need to specify the input encoding — at least if one wanted to input accented characters directly from the keyboard instead of using TeX constructs like `\"a`. One problem in this respect was that, depending on the language you were writing in, it sometimes worked even without specifying the input encoding. This was possible because the `T1` font encoding was nearly identical to the quite common `latin1` input encoding.[2] Years later, omitting the input encoding declaration even when it worked initially finally backfired: once LaTeX moved on to make UTF-8 the default encoding, documents stored in legacy encodings failed if they didn't contain an input declaration.[3]

Pitfalls with the `TS1` encoding

When 8-bit fonts became more common, the TeX community defined two font encodings during a conference at Cork in 1990. The first is `T1`, which holds common Latin text glyphs that play a role in hyphenation and therefore have to be present in the same font when seen by TeX. The second is `TS1`, which contains other symbols, such as oldstyle numerals or currency symbols; these can be fetched from a secondary font without harm to the hyphenation algorithm, because they do not appear as part of words to be hyphenated.

On the whole, the glyphs in the `T1` encoding were well-chosen and it is usually possible to arrange any commercial or free font to be presented in this encoding to TeX.[4] As a result, substituting `T1`-encoded fonts means that you can be fairly sure that there will be no tofu in your output afterwards.

Unfortunately, this is not at all true for the `TS1` encoding. Here the community made a big mistake by going overboard in adding several "supposedly" useful glyphs to the encoding that could be produced in theory (and for Computer Modern and similar TeX fonts were in fact produced), but that simply did not exist in any font that had its origin outside the TeX world.

As a result, to use such glyphs from the `TS1` encoding meant that you had to stay with a very limited number of font families. Alternatively, you had to be very careful not to use any of the problematic symbols to avoid tofu.

To ease this situation, the `TS1` encoding was subdivided into five sub-encodings and a LaTeX interface was established to identify that a font family with a certain NFSS name belonged to one of the sub-encodings. This way LaTeX was enabled to make "reasonable" adjustments when a requested symbol was not available in the current font, either by substituting it from a different font or by giving you an error message that the symbol is not there — not perfect but better than tofu in the end. This was implemented in the `textcomp` package which provided the LaTeX commands to access the symbols from `TS1`.

In one of the recent LaTeX releases the code from `textcomp` was moved to the LaTeX format, so that these extra symbols are now available out of the box without the need to load an additional package. At the same time, the classification of fonts into `TS1` sub-encodings was reworked. We now support nine sub-encodings and the LaTeX format contains close to 200 declarations that sort the commonly available font families into the right sub-encodings. Thus these days the situation is fairly well under control again — at least with pdfTeX.

2 Unicode

One of the goals of Unicode is to uniquely identify each and every character used in different languages and scripts around the world, thereby avoiding some of the possible translation problems that occurred because a text was written under the assumption of one (8-bit) encoding, but interpreted later under a different encoding.

While this was a huge step forward for correctly interpreting any source document (because it eliminated all of the the different input encodings — all is now

[1] Well, more like 99% since sometimes fonts claimed to be in one encoding but didn't faithfully follow its specification, e.g., didn't provide all glyphs or sometimes even placed wrong glyphs at some slots.

[2] For example, with French texts it worked throughout. However, with German only the "umlauts" worked, but the sharp s "ß" generated a different character.

[3] The remedy for such old documents is to either add the missing declaration or re-encode the old source and store it in UTF-8.

[4] There are a few exceptions where some seldom used glyphs are missing, e.g., `\textpertenthousand` or `\textcompwordmark`.

Unicode), it unfortunately reintroduced a new helping of tofu through the back door.

The reason is simple: with Unicode as the means to reliably address a glyph to be typeset in a font, such a font has to contain glyphs for *all* characters available in Unicode, because TeX just takes the Unicode number and tells the current font "typeset this glyph". While this is in theory possible in the TrueType or OpenType font formats (using font collections), there is no single font (or collection) that offers anything close to this.[5] LaTeX has no way to identify if glyphs are missing, because the typesetting of paragraph text is a very low-level process in TeX and in contrast to the pdfTeX engine where LaTeX can reliably assume that a font in T1 encoding implements the whole encoding, in Unicode engines all fonts are in the TU encoding (the whole of Unicode), which no font provides.

In theory it would have been possible to devise sub-encodings of TU and assign each and every font to the appropriate sub-encoding, as was done with TS1, but in practice this would be a hopeless undertaking, because each and every font implements its own set of glyphs, so no useful classification is possible.

Thus when you typeset in XeTeX or LuaTeX and you request using a certain font family with something like

```
\setmainfont{Alegreya}
```

you just have to hope your chosen family contains glyphs for all characters that you intend to use in your document; if not, you will end up with tofu.

To give you some figures: Latin Modern Roman (the default font in LaTeX on Unicode engines) implements 794 characters, the ParaType font used for this article 717, the Optima font on the Mac just 264, the free Alegreya font 1251 and Noto Serif 2840. Regardless, there can be no guarantee that the characters contained in your document are covered.

Letting TeX tell you about your tofu

The TeX program offers one tracing parameter, called `\tracinglostchars`, that, if set to a positive value, reports missing glyphs (a.k.a. tofu) in the log file, e.g.,

```
Missing character: There is no
                È (U+00C8) in font cmr10!
```

[5] The font I know that comes closest is Code2000 [2], which provides around 90K characters in its latest incarnation — but even that is only a fraction of the Unicode universe (over 140K characters). Google's Noto project [1], which stands for "**no to**fu", was established to develop fonts for typesetting text in any of the world's languages and scripts. It currently has almost 64K characters, which are split across nearly two hundred font families, e.g., if you want to typeset in Latin you can use Noto Sans, but for Japanese you need Noto Sans Japanese and so forth.

Interestingly enough, this information is not even given by default, but only when you explicitly ask for it — obviously, Don Knuth did not foresee that TeX is used with fonts other than those carefully crafted for TeX and containing all the characters you may want.

Recently all TeX engines were enhanced to make tofu reporting a little better: you can now set this parameter to 2, after which it reports its finding also on the terminal (the new default value in LaTeX), or you can set it to 3, after which it will throw an error rather than the easy-to-miss warning. With Unicode engines we strongly recommend to always set

```
\tracinglostchars=3
```

in the preamble of your document — it is much better to get errors when writing your documents instead of getting reports by others about tofu in your published work. As explained before, when typesetting with pdfTeX there is little danger of ending up with tofu, so there it is less important to change the parameter, though it obviously doesn't hurt.

3 Typesetting Unicode font tables

When I worked on the font chapter for the new edition of *The LaTeX Companion*, third edition [5], I wanted to produce glyph tables for various fonts to examine which characters they encode and how they looked. To my surprise I could not find any TeX tool to do this for me. There is, of course, the old `nfssfont` which I had adapted from work by Don Knuth, but that is of no help with Unicode fonts as it can only display tables of the first 256 characters, i.e., 8-bit fonts. So during my last stay at Bachotek (before the pandemic) I sketched out some code, the result of which is now available as the `unicodefonttable` package (see companion article).

References

[1] Google Fonts. Noto: A typeface for the world. `fonts.google.com/noto`

[2] J. Kass. Code2000. Font resource implementing much of Unicode. `en.wikipedia.org/wiki/Code2000`

[3] D.E. Knuth. *The TeXbook*, vol. A of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986.

[4] F. Mittelbach, R. Schöpf. The new font family selection — User interface to standard LaTeX. *TUGboat* 11(1):91–97, Apr. 1990. `tug.org/TUGboat/tb11-1/tb27mitt.pdf`

[5] F. Mittelbach with U. Fischer. *The LaTeX Companion*. Pearson Education, Boston, MA, USA, third ed., to appear in 2022.

◇ Frank Mittelbach
Mainz, Germany
`https://www.latex-project.org`
`https://ctan.org/pkg/unicodefonttable`

The `unicodefonttable` package[*]

Frank Mittelbach

Abstract

A package for typesetting font tables for larger fonts, e.g., TrueType or OpenType Unicode fonts. To produce a one-off table, a standalone version is available as well.

Contents

1 Introduction 287
2 The user interface 288
2.1 Keys and their values 289
2.2 A standalone interactive version 291
3 Notes on the table data 291
A Index 292
B Examples 292
B.1 Computer Modern Sans — 7-bit font 292
B.2 TEX Gyre Heros — 8-bit font 293
B.3 Latin Modern Math — 8-bit fonts 293
B.4 Latin Modern Math compared to New Computer Modern Math . . 295
B.5 Garamond Libre's Byzantine Musical Symbols 303

1 Introduction

When I started to write a new chapter for the third edition of *The LATEX Companion* on modern fonts available for different LATEX engines, I was a bit surprised that I couldn't find a way to easily typeset tables showing the glyphs available in TrueType or OpenType fonts. The `nfssfont` package available with LATEX only supports fonts from the 8-bit world, but modern fonts that can be used with XƎTEX or LuaTEX can contain thousands of glyphs and having a method to display what is available in them was important for me.

I therefore set out to write my own little package and what started as an afternoon exercise ended up being this package, offering plenty of bells and whistles for typesetting such font tables.

As there can be many glyphs in such fonts a tabular representation of them might run for several pages, so the package internally uses the `longtable` package to handle that.

In most cases the glyphs inside the fonts are indexed by their Unicode numbers so it is natural to display them sorted by their position in the Unicode character set.

Unicode is organized in named blocks such as "Basic Latin", "Latin-1 Supplement", etc., typically consisting of 265 characters each.[1] It is therefore helpful to use these block names as subtitles within the table, to more easily find the information one is looking for.

A common way to represent the number of a single Unicode character is `U+` followed by four (or more) hexadecimal digits. For example, `U+0041` represents the letter "A" and `U+20AC` the Euro currency symbol "€". We use this convention by showing a Unicode range of sixteen characters at the left of each table row, e.g., `U+0040 - 004F`, followed by the sixteen glyphs in the range. Thus that particular table row from the "Basic

[*] This is version v1.0e of the package, dated 2021/10/29; the license is LPPL.
[1] Some blocks are smaller, while those containing the Asian ideographs are much larger.

doi.org/10.47397/tb/42-3/tb132mitt-unicodefonttable

Latin" block would show something like

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+0040 - 004F	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O

If a Unicode character has no glyph representation in a given font then this is indicated by a special symbol (by default a colored hyphen). By default some color is used, but we've grayscaled the output for *TUGboat*.

In order to easily locate any Unicode character the table shows by default sixteen hex digits as a column heading. For example, to find Euro currency symbol (`U+20AC`) one first finds the right row, which is the range `U+20A0 - 20AF`, and then the `C` column in that row, and the glyph is there (or an indication that the font is missing that glyph; the line shows that for some of the other slots).

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+20A0 - 20AF	-	₡	-	-	₤	-	₦	-	-	₩	-	₫	€	-	-	-

It can be useful to compare two fonts with each other by filling the table with glyphs from a secondary font if the primary font is missing them. For example, the next display shows two rows of Latin Modern Math (black glyphs) and instead of showing a missing glyph symbol in most slots, we use the glyphs from New Computer Modern Math, which has a much larger glyph set (normally red glyphs with gray background but again, grayscaled for *TUGboat*).

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+2A00 - 2A0F	⨀	⨁	⨂	⨃	⨄	⨅	⨆	⨇	⨈	⨉	⨊	⨋	⨌	⨍	⨎	⨏
U+2A10 - 2A1F	⨐	⨑	⨒	⨓	⨔	⨕	⨖	⨗	⨘	⨙	⨚	⨛	⨜	⨝	⨞	⨟

2 The user interface

The package offers one command to typeset a font table. The appearance of the table can be customized by specifying key/value pairs.

`\displayfonttable`

`\displayfonttable * [`⟨*key/value-list*⟩`] {`⟨*font-name*⟩`} [`⟨*font-features*⟩`]`

The ⟨*font-name*⟩ is the font to be displayed. This and the ⟨*font-features*⟩ argument are passed to fontspec, thus they should follow the conventions of that package for specifying a font. The ⟨*key/value-list*⟩ offers customization possibilities discussed below.

The `\displayfonttable*` is a variant of the command, intended for use with 8-bit legacy fonts. It presets some keys, but otherwise behaves identically. The preset values are:

```
nostatistics, display-block=none, hex-digits=head, range-end=FF
```

For details see the next section.

`\fonttablesetup`

`\fonttablesetup {`⟨*key/value-list*⟩`}`

Instead of or in addition to specifying key/values to `\displayfonttable` it is possible to set them up as defaults. Inside `\displayfonttable` the defaults are applied first, so one can still overwrite their values for an individual table.

`\fonttableglyphcount`

`\fonttableglyphcount`

While typesetting a font table the package keeps track of the number of glyphs it finds in the font. After the table has finished, this value is available in `\fonttableglyphcount` and it is, for example, used when statistics are produced. At the start of the next table it is reset to zero.

2.1 Keys and their values

Several of the available keys are booleans accepting `true` or `false`. They usually exist in pairs so that one can specify the desired behavior without needing to provide a value, e.g., specifying `header` is equivalent to specifying `header=true` or `noheader=false`, etc. In the lists below the default settings are indicated by an underline.

`header`
`noheader`
`title-format`
`title-format-cont`

The first set of keys is concerned with the overall look and feel of the generated table.

<u>`header`</u>, `noheader` These keys determine whether a header to the table is produced.

`title-format`, `title-format-cont` These keys define what is provided as a header title or continuation title if the table consists of several pages. They expect code as their value. This code can contain `#1` and `#2` to denote the ⟨*font-name*⟩ and ⟨*font-features*⟩ arguments, respectively.

By default a title using the `\caption` command is produced; on continuation titles, the ⟨*font-features*⟩ are not shown. This is typeset as a `longtable` header row, so you either need to use `\multicolumn` or a `\caption` command — otherwise everything ends up in the first column.

`display-block`
`hex-digits`
`color`

These keys handle the inner parts of the table.

`display-block` The Unicode dataset is organized in named blocks that are typically 128 or 256 characters, though some are noticeably larger and a few are smaller. With the `display-block` key it is possible to specify if and how such blocks should be made visible. The following values are supported:

- **<u>`titles`</u>** Above each display block that contains glyphs the Unicode title of the block is displayed.
- **`rules`** Display blocks are indicated only by a `\midrule`.
- **`none`** Display blocks are not indicated at all.

`hex-digits` To ease reading the table, rows of hex digits are added to it. Where or if this happens is controlled by this key. Allowed values for it are the following:

- **`block`** A row of hex digits is placed at the beginning of each Unicode block containing glyphs in the displayed font.
- **`foot`** A row is added to the foot of each table page.
- **<u>`head`</u>** A row is added to the top of each table page.
- **`head+foot`** A row is added to the top and the foot of each table page.
- **`none`** All hex digit rows are suppressed.

`hex-digits-font` The font to use for the hex digits, by default <u>`\ttfamily\scriptsize`</u>.

`color` This key determines the color for parts of the table (hex digits and Unicode ranges). It can be either `none` or a color specification as understood by the `\color` command. The default is <u>`blue`</u>.

`statistics`
`nostatistics`
`statistics-font`
`statistics-format`

The next set of keys allows altering the statistics that are produced.

<u>`statistics`</u>, `nostatistics` These keys determine whether some statistics are listed at the end of the table.

`statistics-font` The font used to typeset the statistics; the default is <u>`\normalfont\small`</u>.

`statistics-format` Code (text) to specify what should be typeset in the statistics. One can use `#1` for the ⟨*font-name*⟩ and `#2` for the glyph count. The material is typeset on a single line at the end of the table. If several lines are needed you need to use `\parbox` or a similar construct.

`glyph-width`
`missing-glyph`
`missing-glyph-font`
`missing-glyph-color`

Another set of keys deals with customization on the glyph level.

`glyph-width` All glyphs are typeset in a box with the same width, the default value is `6pt` which is suitable for most 10pt fonts and make the table fit comfortably into the text width of a typical document.

`missing-glyph` If a slot in a row doesn't have a glyph in the font you may still want display something to indicate this state. By giving the key a value any arbitrary glyph or material can be typeset. The default is to typeset a `-` (hyphen) in a special color.

Rows that contain no glyph whatsoever are not displayed at all. Instead a small vertical space is added to indicate the one or more rows are omitted.

`missing-glyph-font` The font used for the missing glyphs (the default value is `\ttfamily\scriptsize`).

`missing-glyph-color` If not specified it uses the value specified with the `color` key. If you want a different color, e.g., `red`, you can use a color value or you can specify `none` to use no coloring.

`compare-with`
`compare-color`
`compare-bgcolor`
`statistics-compare-format`

You can make comparisons between two fonts, which is useful, for example when dealing with incomplete math fonts and you need to see how well the symbols from one font blend with the supplementary symbols from another font.

`compare-with` If given, the value is a ⟨*comparison-font-name*⟩ that is used to supply missing glyphs. This means that if the ⟨*font-name*⟩ to be displayed is missing a glyph in a slot, then the ⟨*comparison-font-name*⟩ is checked, and if that font has the glyph in question, it will be displayed instead of showing a missing glyph indicator.

`compare-color`, `compare-bgcolor` To distinguish real glyphs from missing but substituted glyphs, they can be colored specially (default `red`) and/or you can have their background colored (default is `black!10`, i.e., a light gray).

`statistics-compare-format` Code (text) to specify what should be typeset in the statistics when comparing two fonts. One can use `#1` for the ⟨*font-name*⟩ and `#2` for its glyph count, `#3` is the name of the comparison font, `#4` its glyph count, `#5` for the number of glyphs missing in this font and `#6` the number of extra glyphs in it. This code is used instead of `statistics-format` when comparisons are made.

The material is typeset on a single line at the end of the table. If several lines are needed you need to use `\parbox` or a similar construct.

`range-start`
`range-end`

Finally there are two keys for restricting the display range.

`range-start`, `range-end` The full Unicode set of characters is huge and checking every slot to see if the current font contains a glyph in the slot takes a long time. If you know that font contains only a certain subset then you can speed up the table generation considerably by limiting the search (and consequently the output generation). The `range-start` specifies where to start with the search (default `0000`) and `range-end` gives the last slot that is tested (default `FFFF`).

Thus, by default we restrict the display to slots below `10000`, because text fonts seldom contain glyphs in the higher planes. But if you want to see everything of the font (as far as supported by this package) and are prepared to wait for the higher planes to be scanned, you can go up to a value of `FFFFF`.

These keys are also quite useful in combination with the previous `compare-with` key, to display only, for example, the Greek letters and see how glyphs from two fonts blend with each other.

Frank Mittelbach

2.2 A standalone interactive version

If you want to quickly display a single font, you can run `unicodefont.tex` through LuaTEX (or XeTEX). Similar to `nfssfont.tex` (which is for 8-bit fonts with pdfTEX) it asks you a few questions and then generates the font table for you. There are fewer configuration options available, but this workflow saves you writing a document to get a one-off table.

Most font tables need several runs due to the use of `longtable`, which has to find the right width for the columns across several pages. The `unicodefont` file therefore remembers your selection from the previous run and asks you if you want to reapply it to speed up the process.

3 Notes on the table data

If you look at some parts of a Unicode font table you see a number of slots that do not show a "missing glyph" sign, but nonetheless appear to be empty. For example:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+0020 - 002F		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
U+0030 - 003F	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
U+0040 - 004F	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
U+0050 - 005F	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
U+0060 - 006F	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
U+0070 - 007F	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	-
U+00A0 - 00AF		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬		®	¯
U+00B0 - 00BF	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿

The reason is that Unicode contains a lot of special spaces or otherwise invisible characters, e.g., `U+0020` is the normal space, `U+00A0` is a non-breaking space, `U+00AD` is a soft-hyphen (what LaTeX users would indicate with `\-`), and so forth. Especially the row `U+2000-200F` in Table 6 looks strange as it appears to be totally empty, but in fact most of its slots contain spaces of different width.

General Punctuation

U+2000 - 200F															-	-
U+2010 - 201F	‐	‑	‒	–	—	―	‖	‗	‘	’	‚	-	“	”	„	-
U+2020 - 202F	†	‡	•	-	-	-	…	-	-	-	-	-	-	-	-	
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Another somewhat surprising area is the "Mathematical Alphanumeric Symbols" block in math fonts, starting at `U+1D400`. There you see a number of missing characters, the first two being `U+1D455` (math italic small h) and `U+1D49D` (math script B).

Mathematical Alphanumeric Symbols

U+1D400 - 1D40F	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**
U+1D410 - 1D41F	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**	**a**	**b**	**c**	**d**	**e**	**f**
U+1D420 - 1D42F	**g**	**h**	**i**	**j**	**k**	**l**	**m**	**n**	**o**	**p**	**q**	**r**	**s**	**t**	**u**	**v**
U+1D430 - 1D43F	**w**	**x**	**y**	**z**	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*
U+1D440 - 1D44F	*M*	*N*	*O*	*P*	*Q*	*R*	*S*	*T*	*U*	*V*	*W*	*X*	*Y*	*Z*	*a*	*b*
U+1D450 - 1D45F	*c*	*d*	*e*	*f*	*g*	-	*i*	*j*	*k*	*l*	*m*	*n*	*o*	*p*	*q*	*r*
U+1D460 - 1D46F	*s*	*t*	*u*	*v*	*w*	*x*	*y*	*z*	***A***	***B***	***C***	***D***	***E***	***F***	***G***	***H***
U+1D470 - 1D47F	***I***	***J***	***K***	***L***	***M***	***N***	***O***	***P***	***Q***	***R***	***S***	***T***	***U***	***V***	***W***	***X***
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

U+1D480 - 1D48F	𝒀	𝒁	𝒂	𝒃	𝒄	𝒅	𝒆	𝒇	𝒈	𝒉	𝒊	𝒋	𝒌	𝒍	𝒎	𝒏
U+1D490 - 1D49F	𝒐	𝒑	𝒒	𝒓	𝒔	𝒕	𝒖	𝒗	𝒘	𝒙	𝒚	𝒛	𝒜	-	𝒞	𝒟
U+1D4A0 - 1D4AF	-	-	𝒢	-	-	𝒥	𝒦	-	-	𝒩	𝒪	𝒫	𝒬	-	𝒮	𝒯
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

In this case the reason is *not* that the font fails to implement the characters, but that these characters have already been defined in earlier revisions of the Unicode standard in the lower Unicode plane. For example, the “h” is the Planck constant `U+210E` and `U+212C` is the script capital B, etc. The Unicode Consortium decided not to encode the *same* character twice, hence the apparent holes.

A Index

Numbers written in italic refer to the page where the corresponding entry is described or mentioned.

`\-` *291*

`\caption` *289*
`\color` *289*
`color` *289*
`compare-bgcolor` *290*
`compare-color` *290*
`compare-with` *290*

`display-block` *289*
`\displayfonttable` *288*, *288*, *292*
`\displayfonttable*` *288*

`\fonttableglyphcount` *288*
`\fonttablesetup` *288*

`glyph-width` *290*

`header` *289*
`hex-digits` *289*

`\midrule` *289*
`missing-glyph` *290*
`missing-glyph-color` *290*
`missing-glyph-font` *290*
`\multicolumn` *289*

`noheader` *289*
`\normalfont` *289*
`nostatistics` *289*

`\parbox` *289*, *290*

`range-end` *290*
`range-start` *290*

`\scriptsize` *289*, *290*
`\small` *289*
`statistics` *289*
`statistics-compare-format` *290*
`statistics-font` *289*
`statistics-format` *289*

`title-format` *289*
`title-format-cont` *289*
`\ttfamily` *289*, *290*

B Examples

In this section we show the results of a few calls to `\displayfonttable`. The tables are a bit easier to navigate if they use color in some places, but for *TUGboat* this is not practical, so we use black and gray.

B.1 Computer Modern Sans — 7-bit font

Our first example is the original Computer Modern Sans, with character codes ≤ 127. Command used:

```
\displayfonttable*[color=none, range-end=7F]{cmss10}
```

Table 1: cmss10

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+0000 - 000F	Γ	Δ	Θ	Λ	Ξ	Π	Σ	Υ	Φ	Ψ	Ω	ff	fi	fl	ffi	ffl

Table 1: cmss10 *cont.*

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+0010 - 001F	ı	ȷ	`	´	ˇ	˘	¯	˚	¸	ß	æ	œ	ø	Æ	Œ	Ø
U+0020 - 002F	˗	!	”	#	$	%	&	’	(	)	*	+	,	-	.	/
U+0030 - 003F	0	1	2	3	4	5	6	7	8	9	:	;	¡	=	¿	?
U+0040 - 004F	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
U+0050 - 005F	P	Q	R	S	T	U	V	W	X	Y	Z	[	“	]	ˆ	˙
U+0060 - 006F	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
U+0070 - 007F	p	q	r	s	t	u	v	w	x	y	z	–	—	˝	˜	¨

B.2 TEX Gyre Heros — 8-bit font

This example shows the TEX Gyre Heros 8-bit font, in the T1 encoding, with character codes ≤ 255. Command used:

```
\displayfonttable*[color=none]{ec-qhvr}
```

Table 2: ec-qhvr

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+0000 - 000F	`	´	ˆ	˜	¨	˝	˚	ˇ	˘	¯	˙	¸	˛	‚	‹	›
U+0010 - 001F	“	”	„	«	»	–	—		∘	ı	ȷ	ff	fi	fl	ffi	ffl
U+0020 - 002F	␣	!	"	#	$	%	&	’	(	)	*	+	,	-	.	/
U+0030 - 003F	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
U+0040 - 004F	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
U+0050 - 005F	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
U+0060 - 006F	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
U+0070 - 007F	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	‐
U+0080 - 008F	Ă	Ą	Ć	Č	Ď	Ě	Ę	Ğ	Ĺ	Ľ	Ł	Ń	Ň	Ŋ	Ő	Ŕ
U+0090 - 009F	Ř	Ś	Š	Ş	Ť	Ţ	Ű	Ů	Ÿ	Ź	Ž	Ż	Ĳ	İ	đ	§
U+00A0 - 00AF	ă	ą	ć	č	ď	ě	ę	ğ	ĺ	ľ	ł	ń	ň	ŋ	ő	ŕ
U+00B0 - 00BF	ř	ś	š	ş	ť	ţ	ű	ů	ÿ	ź	ž	ż	ĳ	¡	¿	£
U+00C0 - 00CF	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
U+00D0 - 00DF	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	Œ	Ø	Ù	Ú	Û	Ü	Ý	Þ	SS
U+00E0 - 00EF	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
U+00F0 - 00FF	ð	ñ	ò	ó	ô	õ	ö	œ	ø	ù	ú	û	ü	ý	þ	ß

B.3 Latin Modern Math — 8-bit fonts

The traditional Latin Modern Math Italic, Symbol and Extension fonts. The symbol font (`lmsy10`) has two characters added to the Computer Modern symbol repertoire, seen in the last row of the table. Commands used:

```
\displayfonttable*[color=none]{lmmi10}
\displayfonttable*[color=none]{lmsy10}
\displayfonttable*[color=none]{lmex10}
```

Table 3: lmmi10

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+0000 - 000F	$\mathit{\Gamma}$	$\mathit{\Delta}$	$\mathit{\Theta}$	$\mathit{\Lambda}$	$\mathit{\Xi}$	$\mathit{\Pi}$	$\mathit{\Sigma}$	$\mathit{\Upsilon}$	$\mathit{\Phi}$	$\mathit{\Psi}$	$\mathit{\Omega}$	α	β	γ	δ	ϵ
U+0010 - 001F	ζ	η	θ	ι	κ	λ	μ	ν	ξ	π	ρ	σ	τ	υ	ϕ	χ
U+0020 - 002F	ψ	ω	ε	ϑ	ϖ	ϱ	ς	φ	$\leftharpoonup$	$\leftharpoondown$	$\rightharpoonup$	$\rightharpoondown$	˓	˒	$\triangleright$	$\triangleleft$
U+0030 - 003F	0	1	2	3	4	5	6	7	8	9	.	,	$<$	$/$	$>$	$\star$
U+0040 - 004F	∂	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O

Table 3: lmmi10 *cont.*

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+0050 - 005F	P	Q	R	S	T	U	V	W	X	Y	Z	$\flat$	$\natural$	$\sharp$	$\smile$	$\frown$
U+0060 - 006F	ℓ	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
U+0070 - 007F	p	q	r	s	t	u	v	w	x	y	z	$\imath$	$\jmath$	$\wp$	$\vec{}$	⁀

Table 4: lmsy10

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+0000 - 000F	−	·	×	∗	÷	⋄	±	∓	⊕	⊖	⊗	⊘	⊙	◯	∘	∙
U+0010 - 001F	≍	≡	⊆	⊇	≤	≥	⪯	⪰	∼	≈	⊂	⊃	≪	≫	≺	≻
U+0020 - 002F	←	→	↑	↓	↔	↗	↘	≃	⇐	⇒	⇑	⇓	⇔	↖	↙	∝
U+0030 - 003F	′	∞	∈	∋	△	▽	/	ꜗ	∀	∃	¬	∅	ℜ	ℑ	⊤	⊥
U+0040 - 004F	ℵ	$\mathcal{A}$	$\mathcal{B}$	$\mathcal{C}$	$\mathcal{D}$	$\mathcal{E}$	$\mathcal{F}$	$\mathcal{G}$	$\mathcal{H}$	$\mathcal{I}$	$\mathcal{J}$	$\mathcal{K}$	$\mathcal{L}$	$\mathcal{M}$	$\mathcal{N}$	$\mathcal{O}$
U+0050 - 005F	$\mathcal{P}$	$\mathcal{Q}$	$\mathcal{R}$	$\mathcal{S}$	$\mathcal{T}$	$\mathcal{U}$	$\mathcal{V}$	$\mathcal{W}$	$\mathcal{X}$	$\mathcal{Y}$	$\mathcal{Z}$	∪	∩	⊎	∧	∨
U+0060 - 006F	⊢	⊣	⌊	⌋	⌈	⌉	{	}	⟨	⟩	|	‖	↕	⇕	∖	≀
U+0070 - 007F	√	⨿	∇	∫	⊔	⊓	⊑	⊒	§	†	‡	¶	♣	♢	♡	♠
U+00A0 - 00AF	-	-	-	-	-	-	-	-	-	-	-	-	⩽	⩾	-	-

Table 5: lmex10

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+0000 - 000F	(	)	[	]	⌊	⌋	⌈	⌉	{	}	⟨	⟩	|	‖	/	\
U+0010 - 001F	(	)	(	)	[	]	⌊	⌋	⌈	⌉	{	}	⟨	⟩	/	\
U+0020 - 002F	(	)	[	]	⌊	⌋	⌈	⌉	{	}	⟨	⟩	/	\	/	\
U+0030 - 003F	⎛	⎞	⎡	⎤	⎣	⎦	⎢	⎥	⎧	⎫	⎩	⎭	⎨	⎬	⎪	⏐
U+0040 - 004F	⎝	⎠	⎜	⎟	⟨	⟩	⊔	⨆	∮	∮	⊙	⨀	⊕	⨁	⊗	⨂
U+0050 - 005F	∑	∏	∫	⋃	⋂	⊎	⋀	⋁	∑	∏	∫	⋃	⋂	⨄	⋀	⋁
U+0060 - 006F	∐	∐	$\widehat{}$	$\widehat{\ \ }$	$\widehat{\ \ \ }$	$\widetilde{}$	$\widetilde{\ \ }$	$\widetilde{\ \ \ }$	[	]	⌊	⌋	⌈	⌉	{	}
U+0070 - 007F	√	√	√	√	⎷	⏐	⌜	‖	↑	↓	◜	◝	◟	◞	⇑	⇓

B.4 Latin Modern Math compared to New Computer Modern Math

This example shows the extra symbols available in New Computer Modern Math in comparison to Latin Modern Math as the base font. We use the following setup (including settings for the grayscaled *TUGboat* output, as an example of color overrides):

```
\displayfonttable[hex-digits=head+foot, range-end=1FFFF,
                  compare-with=New Computer Modern Math,
                  title-format=\caption{Latin Modern Math compared to
                                        New Computer Modern Math},
                  title-format-cont=\caption{LM Math vs.\ NewCM Math,
                                        \emph{cont.}},
                  compare-color=black,  compare-bgcolor=black!5,
                  missing-glyph-color=black!50, color=black!75]
                 {Latin Modern Math}
```

That is, glyphs only in `NewCM` are shown with a light gray background.

We also extended the range to cover `U+10000` to `U+1FFFF` in order to include the Unicode Math alphabets.

Table 6: Latin Modern Math compared to New Computer Modern Math

Basic Latin

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+0020 - 002F		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
U+0030 - 003F	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
U+0040 - 004F	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
U+0050 - 005F	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
U+0060 - 006F	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
U+0070 - 007F	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Latin-1 Supplement

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+00A0 - 00AF		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬		®	¯
U+00B0 - 00BF	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
U+00C0 - 00CF	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
U+00D0 - 00DF	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
U+00E0 - 00EF	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
U+00F0 - 00FF	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Latin Extended-A

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+0100 - 010F	Ā	ā	Ă	ă	Ą	ą	Ć	ć	Ĉ	ĉ	Ċ	ċ	Č	č	Ď	ď
U+0110 - 011F	Đ	đ	Ē	ē	Ĕ	ĕ	Ė	ė	Ę	ę	Ě	ě	Ĝ	ĝ	Ğ	ğ
U+0120 - 012F	Ġ	ġ	Ģ	ģ	Ĥ	ĥ	Ħ	ħ	Ĩ	ĩ	Ī	ī	Ĭ	ĭ	Į	į
U+0130 - 013F	İ	ı	Ĳ	ĳ	Ĵ	ĵ	Ķ	ķ	ĸ	Ĺ	ĺ	Ļ	ļ	Ľ	ľ	Ŀ
U+0140 - 014F	ŀ	Ł	ł	Ń	ń	Ņ	ņ	Ň	ň	ŉ	Ŋ	ŋ	Ō	ō	Ŏ	ŏ
U+0150 - 015F	Ő	ő	Œ	œ	Ŕ	ŕ	Ŗ	ŗ	Ř	ř	Ś	ś	Ŝ	ŝ	Ş	ş
U+0160 - 016F	Š	š	Ţ	ţ	Ť	ť	Ŧ	ŧ	Ũ	ũ	Ū	ū	Ŭ	ŭ	Ů	ů
U+0170 - 017F	Ű	ű	Ų	ų	Ŵ	ŵ	Ŷ	ŷ	Ÿ	Ź	ź	Ż	ż	Ž	ž	ſ

Latin Extended-B

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+0180 - 018F	ƀ	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Table 6: LM Math vs. NewCM Math, *cont.*

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+01A0 – 01AF	Ơ	ơ	–	–	–	–	–	–	–	–	–	–	–	–	–	Ư
U+01B0 – 01BF	ư	–	–	–	–	Ƶ	–	–	–	–	–	–	–	–	–	–
U+0210 – 021F	–	–	–	–	–	–	–	–	Ș	ș	Ț	ț	–	–	–	–
U+0230 – 023F	–	–	–	–	–	–	–	ȷ	–	–	–	–	–	–	–	–

Spacing Modifier Letters

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+02C0 – 02CF	–	–	–	–	–	–	ˆ	ˇ	–	–	–	–	–	–	–	–
U+02D0 – 02DF	–	–	–	–	–	–	–	–	˘	˙	˚	˛	˜	˝	–	–

Combining Diacritical Marks

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+0300 – 030F	̀	́	̂	̃	̄	̅	̆	̇	̈	̉	̊	̋	̌	–	–	̏
U+0310 – 031F	̐	̑	̒	–	–	̕	–	–	–	–	̚	–	–	–	–	–
U+0320 – 032F	–	–	–	̣	–	–	̦	–	–	–	–	–	̬	̭	̮	̯
U+0330 – 033F	̰	̱	̲	̳	–	–	–	–	̸	–	–	–	–	–	–	̿
U+0340 – 034F	–	–	–	–	–	–	–	–	–	–	–	–	–	͍	–	–

Greek and Coptic

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+0390 – 039F	–	Α	Β	Γ	Δ	Ε	Ζ	Η	Θ	Ι	Κ	Λ	Μ	Ν	Ξ	Ο
U+03A0 – 03AF	Π	Ρ	–	Σ	Τ	Υ	Φ	Χ	Ψ	Ω	–	–	–	–	–	–
U+03B0 – 03BF	–	α	β	γ	δ	ε	ζ	η	θ	ι	κ	λ	μ	ν	ξ	ο
U+03C0 – 03CF	π	ρ	ς	σ	τ	υ	φ	χ	ψ	ω	–	–	–	–	–	–
U+03D0 – 03DF	–	ϑ	–	–	–	ϕ	ϖ	–	–	–	–	–	Ϝ	ϝ	–	–
U+03F0 – 03FF	ϰ	ϱ	–	–	ϴ	ϵ	϶	–	–	–	–	–	–	–	–	–

Latin Extended Additional

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+1EA0 – 1EAF	Ạ	ạ	Ả	ả	Ấ	ấ	Ầ	ầ	Ẩ	ẩ	Ẫ	ẫ	Ậ	ậ	Ắ	ắ
U+1EB0 – 1EBF	Ằ	ằ	Ẳ	ẳ	Ẵ	ẵ	Ặ	ặ	Ẹ	ẹ	Ẻ	ẻ	Ẽ	ẽ	Ế	ế
U+1EC0 – 1ECF	Ề	ề	Ể	ể	Ễ	ễ	Ệ	ệ	Ỉ	ỉ	Ị	ị	Ọ	ọ	Ỏ	ỏ
U+1ED0 – 1EDF	Ố	ố	Ồ	ồ	Ổ	ổ	Ỗ	ỗ	Ộ	ộ	Ớ	ớ	Ờ	ờ	Ở	ở
U+1EE0 – 1EEF	Ỡ	ỡ	Ợ	ợ	Ụ	ụ	Ủ	ủ	Ứ	ứ	Ừ	ừ	Ử	ử	Ữ	ữ
U+1EF0 – 1EFF	Ự	ự	Ỳ	ỳ	Ỵ	ỵ	Ỷ	ỷ	Ỹ	ỹ	–	–	–	–	–	–

General Punctuation

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+2000 – 200F															–	–
U+2010 – 201F	‐	‑	‒	–	—	―	‖	‗	‘	’	‚	‛	“	”	„	‟
U+2020 – 202F	†	‡	•	‣	․	‥	…	‧	–	–	–	–	–	–	–	
U+2030 – 203F	‰	‱	′	″	‴	‵	‶	‷	‸	‹	›	※	‼	‽	‾	‿
U+2040 – 204F	⁀	⁁	⁂	⁃	⁄	⁅	⁆	⁇	⁈	⁉	⁊	⁋	⁌	⁍	⁎	⁏
U+2050 – 205F	⁐	⁑	⁒	⁓	⁔	⁕	⁖	⁗	⁘	⁙	⁚	⁛	⁜	⁝	⁞	
U+2060 – 206F						–	–	–	–	–	–	–	–	–	–	–

Currency Symbols

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+20A0 – 20AF	–	₡	–	–	–	–	–	–	–	–	–	–	€	–	–	–

Combining Diacritical Marks for Symbols

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+20D0 – 20DF	⃐	⃑	⃒	⃓	⃔	⃕	⃖	⃗	⃘	–	–	⃛	⃜	⃝	⃞	⃟
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Table 6: LM Math vs. NewCM Math, *cont.*

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+20E0 - 20EF	-	⃡	-	-	⃤	⃥	⃦	⃧	⃨	⃩	⃪	⃫	⃬	⃭	⃮	⃯
U+20F0 - 20FF	⃰	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Letterlike Symbols

U+2100 - 210F	℀	℁	ℂ	℃	℄	℅	℆	ℇ	℈	℉	ℊ	ℋ	ℌ	ℍ	ℎ	ℏ
U+2110 - 211F	ℐ	ℑ	ℒ	ℓ	℔	ℕ	№	℗	℘	ℙ	ℚ	ℛ	ℜ	ℝ	℞	℟
U+2120 - 212F	℠	℡	™	℣	ℤ	℥	Ω	℧	ℨ	℩	K	Å	ℬ	ℭ	℮	ℯ
U+2130 - 213F	ℰ	ℱ	Ⅎ	ℳ	ℴ	ℵ	ℶ	ℷ	ℸ	ℹ	℺	℻	ℼ	ℽ	ℾ	ℿ
U+2140 - 214F	⅀	⅁	⅂	⅃	⅄	ⅅ	ⅆ	ⅇ	ⅈ	ⅉ	⅊	⅋	⅌	⅍	ⅎ	⅏

Arrows

U+2190 - 219F	←	↑	→	↓	↔	↕	↖	↗	↘	↙	↚	↛	↜	↝	↞	↟
U+21A0 - 21AF	↠	↡	↢	↣	↤	↥	↦	↧	↨	↩	↪	↫	↬	↭	↮	↯
U+21B0 - 21BF	↰	↱	↲	↳	↴	↵	↶	↷	↸	↹	↺	↻	↼	↽	↾	↿
U+21C0 - 21CF	⇀	⇁	⇂	⇃	⇄	⇅	⇆	⇇	⇈	⇉	⇊	⇋	⇌	⇍	⇎	⇏
U+21D0 - 21DF	⇐	⇑	⇒	⇓	⇔	⇕	⇖	⇗	⇘	⇙	⇚	⇛	⇜	⇝	⇞	⇟
U+21E0 - 21EF	⇠	⇡	⇢	⇣	⇤	⇥	⇦	⇧	⇨	⇩	⇪	⇫	⇬	⇭	⇮	⇯
U+21F0 - 21FF	⇰	⇱	⇲	⇳	⇴	⇵	⇶	⇷	⇸	⇹	⇺	⇻	⇼	⇽	⇾	⇿

Mathematical Operators

U+2200 - 220F	∀	∁	∂	∃	∄	∅	∆	∇	∈	∉	∊	∋	∌	∍	∎	∏
U+2210 - 221F	∐	∑	−	∓	∔	∕	∖	∗	∘	∙	√	∛	∜	∝	∞	∟
U+2220 - 222F	∠	∡	∢	∣	∤	∥	∦	∧	∨	∩	∪	∫	∬	∭	∮	∯
U+2230 - 223F	∰	∱	∲	∳	∴	∵	∶	∷	∸	∹	∺	∻	∼	∽	∾	∿
U+2240 - 224F	≀	≁	≂	≃	≄	≅	≆	≇	≈	≉	≊	≋	≌	≍	≎	≏
U+2250 - 225F	≐	≑	≒	≓	≔	≕	≖	≗	≘	≙	≚	≛	≜	≝	≞	≟
U+2260 - 226F	≠	≡	≢	≣	≤	≥	≦	≧	≨	≩	≪	≫	≬	≭	≮	≯
U+2270 - 227F	≰	≱	≲	≳	≴	≵	≶	≷	≸	≹	≺	≻	≼	≽	≾	≿
U+2280 - 228F	⊀	⊁	⊂	⊃	⊄	⊅	⊆	⊇	⊈	⊉	⊊	⊋	⊌	⊍	⊎	⊏
U+2290 - 229F	⊐	⊑	⊒	⊓	⊔	⊕	⊖	⊗	⊘	⊙	⊚	⊛	⊜	⊝	⊞	⊟
U+22A0 - 22AF	⊠	⊡	⊢	⊣	⊤	⊥	⊦	⊧	⊨	⊩	⊪	⊫	⊬	⊭	⊮	⊯
U+22B0 - 22BF	⊰	⊱	⊲	⊳	⊴	⊵	⊶	⊷	⊸	⊹	⊺	⊻	⊼	⊽	⊾	⊿
U+22C0 - 22CF	⋀	⋁	⋂	⋃	⋄	⋅	⋆	⋇	⋈	⋉	⋊	⋋	⋌	⋍	⋎	⋏
U+22D0 - 22DF	⋐	⋑	⋒	⋓	⋔	⋕	⋖	⋗	⋘	⋙	⋚	⋛	⋜	⋝	⋞	⋟
U+22E0 - 22EF	⋠	⋡	⋢	⋣	⋤	⋥	⋦	⋧	⋨	⋩	⋪	⋫	⋬	⋭	⋮	⋯
U+22F0 - 22FF	⋰	⋱	⋲	⋳	⋴	⋵	⋶	⋷	⋸	⋹	⋺	⋻	⋼	⋽	⋾	⋿

Miscellaneous Technical

U+2300 - 230F	⌀	⌁	⌂	⌃	⌄	⌅	⌆	⌇	⌈	⌉	⌊	⌋	⌌	⌍	⌎	⌏
U+2310 - 231F	⌐	⌑	⌒	⌓	⌔	⌕	⌖	⌗	⌘	⌙	⌚	⌛	⌜	⌝	⌞	⌟
U+2320 - 232F	⌠	⌡	⌢	⌣	⌤	⌥	⌦	⌧	⌨	〈	〉	⌫	⌬	⌭	⌮	⌯
U+2330 - 233F	⌰	⌱	⌲	⌳	⌴	⌵	⌶	⌷	⌸	⌹	⌺	⌻	⌼	⌽	⌾	⌿
U+2340 - 234F	⍀	⍁	⍂	⍃	⍄	⍅	⍆	⍇	⍈	⍉	⍊	⍋	⍌	⍍	⍎	⍏
U+2350 - 235F	⍐	⍑	⍒	⍓	⍔	⍕	⍖	⍗	⍘	⍙	⍚	⍛	⍜	⍝	⍞	⍟
U+2360 - 236F	⍠	⍡	⍢	⍣	⍤	⍥	⍦	⍧	⍨	⍩	⍪	⍫	⍬	⍭	⍮	⍯
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Table 6: LM Math vs. NewCM Math, *cont.*

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+2370 – 237F	⍰	⍱	⍲	⍳	⍴	⍵	⍶	⍷	⍸	⍹	⍺	⍻	⍼	⍽	⍾	⍿
U+2380 – 238F	⎀	⎁	⎂	⎃	⎄	⎅	⎆	⎇	⎈	⎉	⎊	⎋	⎌	⎍	⎎	⎏
U+2390 – 239F	⎐	⎑	⎒	⎓	⎔	⎕	⎖	⎗	⎘	⎙	⎚	⎛	⎜	⎝	⎞	⎟
U+23A0 – 23AF	⎠	⎡	⎢	⎣	⎤	⎥	⎦	⎧	⎨	⎩	⎪	⎫	⎬	⎭	⎮	⎯
U+23B0 – 23BF	⎰	⎱	⎲	⎳	⎴	⎵	⎶	⎷	⎸	⎹	⎺	⎻	⎼	⎽	⎾	⎿
U+23C0 – 23CF	⏀	⏁	⏂	⏃	⏄	⏅	⏆	⏇	⏈	⏉	⏊	⏋	⏌	⏍	⏎	⏏
U+23D0 – 23DF	⏐	⏑	⏒	⏓	⏔	⏕	⏖	⏗	⏘	⏙	⏚	⏛	⏜	⏝	⏞	⏟
U+23E0 – 23EF	⏠	⏡	⏢	⏣	⏤	⏥	⏦	⏧	⏨	⏩	⏪	⏫	⏬	⏭	⏮	⏯
U+23F0 – 23FF	⏰	⏱	⏲	⏳	⏴	⏵	⏶	⏷	⏸	⏹	⏺	⏻	⏼	⏽	⏾	⏿

Control Pictures

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+2420 – 242F	–	–	␢	␣	–	–	–	–	–	–	–	–	–	–	–	–

Box Drawing

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+2500 – 250F	─	━	│	┃	┄	┅	┆	┇	┈	┉	┊	┋	┌	┍	┎	┏
U+2510 – 251F	┐	┑	┒	┓	└	┕	┖	┗	┘	┙	┚	┛	├	┝	┞	┟
U+2520 – 252F	┠	┡	┢	┣	┤	┥	┦	┧	┨	┩	┪	┫	┬	┭	┮	┯
U+2530 – 253F	┰	┱	┲	┳	┴	┵	┶	┷	┸	┹	┺	┻	┼	┽	┾	┿
U+2540 – 254F	╀	╁	╂	╃	╄	╅	╆	╇	╈	╉	╊	╋	╌	╍	╎	╏
U+2550 – 255F	═	║	╒	╓	╔	╕	╖	╗	╘	╙	╚	╛	╜	╝	╞	╟
U+2560 – 256F	╠	╡	╢	╣	╤	╥	╦	╧	╨	╩	╪	╫	╬	╭	╮	╯
U+2570 – 257F	╰	╱	╲	╳	╴	╵	╶	╷	╸	╹	╺	╻	╼	╽	╾	╿

Block Elements

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+2580 – 258F	▀	▁	▂	▃	▄	▅	▆	▇	█	▉	▊	▋	▌	▍	▎	▏
U+2590 – 259F	▐	░	▒	▓	▔	▕	▖	▗	▘	▙	▚	▛	▜	▝	▞	▟

Geometric Shapes

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+25A0 – 25AF	■	□	▢	▣	▤	▥	▦	▧	▨	▩	▪	▫	▬	▭	▮	▯
U+25B0 – 25BF	▰	▱	▲	△	▴	▵	▶	▷	▸	▹	►	▻	▼	▽	▾	▿
U+25C0 – 25CF	◀	◁	◂	◃	◄	◅	◆	◇	◈	◉	◊	○	◌	◍	◎	●
U+25D0 – 25DF	◐	◑	◒	◓	◔	◕	◖	◗	◘	◙	◚	◛	◜	◝	◞	◟
U+25E0 – 25EF	◠	◡	◢	◣	◤	◥	◦	◧	◨	◩	◪	◫	◬	◭	◮	◯
U+25F0 – 25FF	◰	◱	◲	◳	◴	◵	◶	◷	◸	◹	◺	◻	◼	◽	◾	◿

Miscellaneous Shapes

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+2600 – 260F	☀	–	–	–	–	★	☆	–	–	☉	–	–	–	–	–	–
U+2620 – 262F	–	☡	–	–	–	–	–	–	–	–	–	–	–	–	–	–
U+2630 – 263F	–	–	–	–	–	–	–	–	–	☹	☺	☻	☼	☽	☾	–
U+2640 – 264F	♀	–	♂	–	–	–	–	–	–	–	–	–	–	–	–	–
U+2660 – 266F	♠	♡	♢	♣	♤	♥	♦	♧	–	♩	♪	♫	–	♭	♮	♯
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Table 6: LM Math vs. NewCM Math, *cont.*

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+2670 – 267F	–	–	–	–	–	–	–	–	–	–	–	–	–	–	♾	–
U+2680 – 268F	⚀	⚁	⚂	⚃	⚄	⚅	⚆	⚇	⚈	⚉	–	–	–	–	–	–
U+26A0 – 26AF	–	–	–	–	–	⚥	–	–	–	–	⚪	⚫	⚬	⚭	⚮	–
U+26B0 – 26BF	–	–	⚲	–	–	–	–	–	–	–	–	–	–	–	–	–

Dingbats

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+2710 – 271F	–	–	–	✓	–	–	–	–	–	–	–	–	–	–	–	–
U+2720 – 272F	✠	–	–	–	–	–	–	–	–	–	✪	–	–	–	–	–
U+2730 – 273F	–	–	–	–	–	–	✶	–	–	–	–	–	–	✽	–	–
U+2750 – 275F	–	–	–	–	–	–	–	–	–	–	❚	–	–	–	–	–
U+2770 – 277F	–	–	❲	❳	–	–	–	–	–	–	–	–	–	–	–	–
U+2790 – 279F	–	–	–	–	–	–	–	–	–	–	–	➛	–	–	–	–
U+27A0 – 27AF	–	➡	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Miscellaneous Mathematical Symbols-A

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+27C0 – 27CF	⟀	⟁	⟂	⟃	⟄	⟅	⟆	⟇	⟈	⟉	⟊	⟋	⟌	⟍	⟎	⟏
U+27D0 – 27DF	⟐	⟑	⟒	⟓	⟔	⟕	⟖	⟗	⟘	⟙	⟚	⟛	⟜	⟝	⟞	⟟
U+27E0 – 27EF	⟠	⟡	⟢	⟣	⟤	⟥	⟦	⟧	⟨	⟩	⟪	⟫	⟬	⟭	⟮	⟯

Supplemental Arrows-A

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+27F0 – 27FF	⟰	⟱	⟲	⟳	⟴	⟵	⟶	⟷	⟸	⟹	⟺	⟻	⟼	⟽	⟾	⟿

Supplemental Arrows-B

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+2900 – 290F	⤀	⤁	⤂	⤃	⤄	⤅	⤆	⤇	⤈	⤉	⤊	⤋	⤌	⤍	⤎	⤏
U+2910 – 291F	⤐	⤑	⤒	⤓	⤔	⤕	⤖	⤗	⤘	⤙	⤚	⤛	⤜	⤝	⤞	⤟
U+2920 – 292F	⤠	⤡	⤢	⤣	⤤	⤥	⤦	⤧	⤨	⤩	⤪	⤫	⤬	⤭	⤮	⤯
U+2930 – 293F	⤰	⤱	⤲	⤳	⤴	⤵	⤶	⤷	⤸	⤹	⤺	⤻	⤼	⤽	⤾	⤿
U+2940 – 294F	⥀	⥁	⥂	⥃	⥄	⥅	⥆	⥇	⥈	⥉	⥊	⥋	⥌	⥍	⥎	⥏
U+2950 – 295F	⥐	⥑	⥒	⥓	⥔	⥕	⥖	⥗	⥘	⥙	⥚	⥛	⥜	⥝	⥞	⥟
U+2960 – 296F	⥠	⥡	⥢	⥣	⥤	⥥	⥦	⥧	⥨	⥩	⥪	⥫	⥬	⥭	⥮	⥯
U+2970 – 297F	⥰	⥱	⥲	⥳	⥴	⥵	⥶	⥷	⥸	⥹	⥺	⥻	⥼	⥽	⥾	⥿

Miscellaneous Mathematical Symbols-B

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+2980 – 298F	⦀	⦁	⦂	⦃	⦄	⦅	⦆	⦇	⦈	⦉	⦊	⦋	⦌	⦍	⦎	⦏
U+2990 – 299F	⦐	⦑	⦒	⦓	⦔	⦕	⦖	⦗	⦘	⦙	⦚	⦛	⦜	⦝	⦞	⦟
U+29A0 – 29AF	⦠	⦡	⦢	⦣	⦤	⦥	⦦	⦧	⦨	⦩	⦪	⦫	⦬	⦭	⦮	⦯
U+29B0 – 29BF	⦰	⦱	⦲	⦳	⦴	⦵	⦶	⦷	⦸	⦹	⦺	⦻	⦼	⦽	⦾	⦿
U+29C0 – 29CF	⧀	⧁	⧂	⧃	⧄	⧅	⧆	⧇	⧈	⧉	⧊	⧋	⧌	⧍	⧎	⧏
U+29D0 – 29DF	⧐	⧑	⧒	⧓	⧔	⧕	⧖	⧗	⧘	⧙	⧚	⧛	⧜	⧝	⧞	⧟
U+29E0 – 29EF	⧠	⧡	⧢	⧣	⧤	⧥	⧦	⧧	⧨	⧩	⧪	⧫	⧬	⧭	⧮	⧯
U+29F0 – 29FF	⧰	⧱	⧲	⧳	⧴	⧵	⧶	⧷	⧸	⧹	⧺	⧻	⧼	⧽	⧾	⧿
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Table 6: LM Math vs. NewCM Math, *cont.*

Supplemental Mathematical Operators

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+2A00 – 2A0F	⨀	⨁	⨂	⨃	⨄	⨅	⨆	⨇	⨈	⨉	⨊	⨋	⨌	⨍	⨎	⨏
U+2A10 – 2A1F	⨐	⨑	⨒	⨓	⨔	⨕	⨖	⨗	⨘	⨙	⨚	⨛	⨜	⨝	⨞	⨟
U+2A20 – 2A2F	⨠	⨡	⨢	⨣	⨤	⨥	⨦	⨧	⨨	⨩	⨪	⨫	⨬	⨭	⨮	⨯
U+2A30 – 2A3F	⨰	⨱	⨲	⨳	⨴	⨵	⨶	⨷	⨸	⨹	⨺	⨻	⨼	⨽	⨾	⨿
U+2A40 – 2A4F	⩀	⩁	⩂	⩃	⩄	⩅	⩆	⩇	⩈	⩉	⩊	⩋	⩌	⩍	⩎	⩏
U+2A50 – 2A5F	⩐	⩑	⩒	⩓	⩔	⩕	⩖	⩗	⩘	⩙	⩚	⩛	⩜	⩝	⩞	⩟
U+2A60 – 2A6F	⩠	⩡	⩢	⩣	⩤	⩥	⩦	⩧	⩨	⩩	⩪	⩫	⩬	⩭	⩮	⩯
U+2A70 – 2A7F	⩰	⩱	⩲	⩳	⩴	⩵	⩶	⩷	⩸	⩹	⩺	⩻	⩼	⩽	⩾	⩿
U+2A80 – 2A8F	⪀	⪁	⪂	⪃	⪄	⪅	⪆	⪇	⪈	⪉	⪊	⪋	⪌	⪍	⪎	⪏
U+2A90 – 2A9F	⪐	⪑	⪒	⪓	⪔	⪕	⪖	⪗	⪘	⪙	⪚	⪛	⪜	⪝	⪞	⪟
U+2AA0 – 2AAF	⪠	⪡	⪢	⪣	⪤	⪥	⪦	⪧	⪨	⪩	⪪	⪫	⪬	⪭	⪮	⪯
U+2AB0 – 2ABF	⪰	⪱	⪲	⪳	⪴	⪵	⪶	⪷	⪸	⪹	⪺	⪻	⪼	⪽	⪾	⪿
U+2AC0 – 2ACF	⫀	⫁	⫂	⫃	⫄	⫅	⫆	⫇	⫈	⫉	⫊	⫋	⫌	⫍	⫎	⫏
U+2AD0 – 2ADF	⫐	⫑	⫒	⫓	⫔	⫕	⫖	⫗	⫘	⫙	⫚	⫛	⫝̸	⫝	⫞	⫟
U+2AE0 – 2AEF	⫠	⫡	⫢	⫣	⫤	⫥	⫦	⫧	⫨	⫩	⫪	⫫	⫬	⫭	⫮	⫯
U+2AF0 – 2AFF	⫰	⫱	⫲	⫳	⫴	⫵	⫶	⫷	⫸	⫹	⫺	⫻	⫼	⫽	⫾	⫿

Miscellaneous Symbols and Arrows

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+2B00 – 2B0F	⬀	⬁	⬂	⬃	⬄	⬅	⬆	⬇	⬈	⬉	⬊	⬋	⬌	⬍	⬎	⬏
U+2B10 – 2B1F	⬐	⬑	⬒	⬓	⬔	⬕	⬖	⬗	⬘	⬙	⬚	⬛	⬜	⬝	⬞	⬟
U+2B20 – 2B2F	⬠	⬡	⬢	⬣	⬤	⬥	⬦	⬧	⬨	⬩	⬪	⬫	⬬	⬭	⬮	⬯
U+2B30 – 2B3F	⬰	⬱	⬲	⬳	⬴	⬵	⬶	⬷	⬸	⬹	⬺	⬻	⬼	⬽	⬾	⬿
U+2B40 – 2B4F	⭀	⭁	⭂	⭃	⭄	⭅	⭆	⭇	⭈	⭉	⭊	⭋	⭌	⭍	⭎	⭏
U+2B50 – 2B5F	⭐	⭑	⭒	⭓	⭔	⭕	⭖	⭗	⭘	⭙	⭚	⭛	⭜	⭝	⭞	⭟
U+2B60 – 2B6F	⭠	⭡	⭢	⭣	⭤	⭥	⭦	⭧	⭨	⭩	⭪	⭫	⭬	⭭	⭮	⭯
U+2B70 – 2B7F	⭰	⭱	⭲	⭳	–	–	⭶	⭷	⭸	⭹	⭺	⭻	⭼	⭽	⭾	⭿
U+2B80 – 2B8F	⮀	⮁	⮂	⮃	⮄	⮅	⮆	⮇	⮈	⮉	⮊	⮋	⮌	⮍	⮎	⮏
U+2B90 – 2B9F	⮐	⮑	⮒	⮓	⮔	⮕	–	⮗	⮘	⮙	⮚	⮛	⮜	⮝	⮞	⮟
U+2BA0 – 2BAF	⮠	⮡	⮢	⮣	⮤	⮥	⮦	⮧	⮨	⮩	⮪	⮫	⮬	⮭	⮮	⮯
U+2BB0 – 2BBF	⮰	⮱	⮲	⮳	⮴	⮵	⮶	⮷	⮸	⮹	⮺	⮻	⮼	⮽	⮾	⮿
U+2BC0 – 2BCF	⯀	⯁	⯂	⯃	⯄	⯅	⯆	⯇	⯈	⯉	⯊	⯋	⯌	⯍	⯎	⯏
U+2BD0 – 2BDF	⯐	⯑	⯒	⯓	⯔	⯕	⯖	⯗	⯘	⯙	⯚	⯛	⯜	⯝	⯞	⯟
U+2BE0 – 2BEF	⯠	⯡	⯢	⯣	⯤	⯥	⯦	⯧	⯨	⯩	⯪	⯫	⯬	⯭	⯮	⯯
U+2BF0 – 2BFF	⯰	⯱	⯲	⯳	⯴	⯵	⯶	⯷	⯸	⯹	⯺	⯻	⯼	⯽	⯾	⯿

Supplemental Punctuation

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+2E10 – 2E1F	–	–	–	–	–	–	–	–	⸘	–	–	–	–	–	–	–

Table 6: LM Math vs. NewCM Math, *cont.*

CJK Symbols and Punctuation

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+3010 - 301F	–	–	〒	–	–	–	〖	〗	–	–	–	–	–	–	–	–
U+3030 - 303F	〰	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Private Use Area

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+E000 - E00F	Α	Β	Γ	Δ	Ε	Ζ	Η	Θ	Ι	Κ	Λ	Μ	Ν	Ξ	Ο	Π
U+E010 - E01F	Ρ	Σ	Τ	ϒ	Φ	Χ	Ψ	Ω	α	β	γ	δ	ε	ζ	η	θ
U+E020 - E02F	ι	κ	λ	μ	ν	ξ	ο	π	ρ	ς	σ	τ	υ	φ	χ	ψ
U+E030 - E03F	ω	ϵ	⧣	⧥	–	–	–	–	–	–	–	–	–	–	–	–
U+E040 - E04F	–	*Α*	*Β*	*Γ*	*Δ*	*Ε*	*Ζ*	*Η*	*Θ*	*Ι*	*Κ*	*Λ*	*Μ*	*Ν*	*Ξ*	*Ο*
U+E050 - E05F	*Π*	*Ρ*	*Σ*	*Τ*	*ϒ*	*Φ*	*Χ*	*Ψ*	*Ω*	*α*	*β*	*γ*	*δ*	*ε*	*ζ*	*η*
U+E060 - E06F	*θ*	*ι*	*κ*	*λ*	*μ*	*ν*	*ξ*	*ο*	*π*	*ρ*	*ς*	*σ*	*τ*	*υ*	*φ*	*χ*
U+E070 - E07F	*ψ*	*ω*	*ϵ*	–	–	–	–	–	–	–	–	–	–	–	–	–
U+E370 - E37F	–	–	–	–	–	–	⨕	⨒	–	–	–	–	–	–	–	–
U+E390 - E39F	–	–	–	–	–	⨐	–	⨒	⨓	⨕	⨗	⨑	–	–	–	–
U+E3D0 - E3DF	–	–	–	∫	–	–	–	–	–	–	–	–	–	–	–	–
U+EA50 - EA5F	–	–	–	–	–	–	–	⨋	–	–	–	–	–	–	–	–

Alphabetic Presentation Forms

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+FB00 - FB0F	ﬀ	ﬁ	ﬂ	ﬃ	ﬄ	–	–	–	–	–	–	–	–	–	–	–

Arabic Presentation Forms-B

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+FEF0 - FEFF	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	

Mathematical Alphanumeric Symbols

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+1D400 - 1D40F	𝐀	𝐁	𝐂	𝐃	𝐄	𝐅	𝐆	𝐇	𝐈	𝐉	𝐊	𝐋	𝐌	𝐍	𝐎	𝐏
U+1D410 - 1D41F	𝐐	𝐑	𝐒	𝐓	𝐔	𝐕	𝐖	𝐗	𝐘	𝐙	𝐚	𝐛	𝐜	𝐝	𝐞	𝐟
U+1D420 - 1D42F	𝐠	𝐡	𝐢	𝐣	𝐤	𝐥	𝐦	𝐧	𝐨	𝐩	𝐪	𝐫	𝐬	𝐭	𝐮	𝐯
U+1D430 - 1D43F	𝐰	𝐱	𝐲	𝐳	𝐴	𝐵	𝐶	𝐷	𝐸	𝐹	𝐺	𝐻	𝐼	𝐽	𝐾	𝐿
U+1D440 - 1D44F	𝑀	𝑁	𝑂	𝑃	𝑄	𝑅	𝑆	𝑇	𝑈	𝑉	𝑊	𝑋	𝑌	𝑍	𝑎	𝑏
U+1D450 - 1D45F	𝑐	𝑑	𝑒	𝑓	𝑔	–	𝑖	𝑗	𝑘	𝑙	𝑚	𝑛	𝑜	𝑝	𝑞	𝑟
U+1D460 - 1D46F	𝑠	𝑡	𝑢	𝑣	𝑤	𝑥	𝑦	𝑧	𝑨	𝑩	𝑪	𝑫	𝑬	𝑭	𝑮	𝑯
U+1D470 - 1D47F	𝑰	𝑱	𝑲	𝑳	𝑴	𝑵	𝑶	𝑷	𝑸	𝑹	𝑺	𝑻	𝑼	𝑽	𝑾	𝑿
U+1D480 - 1D48F	𝒀	𝒁	𝒂	𝒃	𝒄	𝒅	𝒆	𝒇	𝒈	𝒉	𝒊	𝒋	𝒌	𝒍	𝒎	𝒏
U+1D490 - 1D49F	𝒐	𝒑	𝒒	𝒓	𝒔	𝒕	𝒖	𝒗	𝒘	𝒙	𝒚	𝒛	𝒜	–	𝒞	𝒟
U+1D4A0 - 1D4AF	–	–	𝒢	–	–	𝒥	𝒦	–	–	𝒩	𝒪	𝒫	𝒬	–	𝒮	𝒯
U+1D4B0 - 1D4BF	𝒰	𝒱	𝒲	𝒳	𝒴	𝒵	𝒶	𝒷	𝒸	𝒹	–	𝒻	–	𝒽	𝒾	𝒿
U+1D4C0 - 1D4CF	𝓀	𝓁	𝓂	𝓃	–	𝓅	𝓆	𝓇	𝓈	𝓉	𝓊	𝓋	𝓌	𝓍	𝓎	𝓏
U+1D4D0 - 1D4DF	𝓐	𝓑	𝓒	𝓓	𝓔	𝓕	𝓖	𝓗	𝓘	𝓙	𝓚	𝓛	𝓜	𝓝	𝓞	𝓟
U+1D4E0 - 1D4EF	𝓠	𝓡	𝓢	𝓣	𝓤	𝓥	𝓦	𝓧	𝓨	𝓩	𝓪	𝓫	𝓬	𝓭	𝓮	𝓯
U+1D4F0 - 1D4FF	𝓰	𝓱	𝓲	𝓳	𝓴	𝓵	𝓶	𝓷	𝓸	𝓹	𝓺	𝓻	𝓼	𝓽	𝓾	𝓿
U+1D500 - 1D50F	𝔀	𝔁	𝔂	𝔃	𝔄	𝔅	–	𝔇	𝔈	𝔉	𝔊	–	–	𝔍	𝔎	𝔏
U+1D510 - 1D51F	𝔐	𝔑	𝔒	𝔓	𝔔	–	𝔖	𝔗	𝔘	𝔙	𝔚	𝔛	𝔜	–	𝔞	𝔟
U+1D520 - 1D52F	𝔠	𝔡	𝔢	𝔣	𝔤	𝔥	𝔦	𝔧	𝔨	𝔩	𝔪	𝔫	𝔬	𝔭	𝔮	𝔯
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Table 6: LM Math vs. NewCM Math, *cont.*

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+1D530 – 1D53F	𝔰	𝔱	𝔲	𝔳	𝔴	𝔵	𝔶	𝔷	𝔸	𝔹	–	𝔻	𝔼	𝔽	𝔾	–
U+1D540 – 1D54F	𝕀	𝕁	𝕂	𝕃	𝕄	–	𝕆	–	–	–	𝕊	𝕋	𝕌	𝕍	𝕎	𝕏
U+1D550 – 1D55F	𝕐	–	𝕒	𝕓	𝕔	𝕕	𝕖	𝕗	𝕘	𝕙	𝕚	𝕛	𝕜	𝕝	𝕞	𝕟
U+1D560 – 1D56F	𝕠	𝕡	𝕢	𝕣	𝕤	𝕥	𝕦	𝕧	𝕨	𝕩	𝕪	𝕫	𝕬	𝕭	𝕮	𝕯
U+1D570 – 1D57F	𝕰	𝕱	𝕲	𝕳	𝕴	𝕵	𝕶	𝕷	𝕸	𝕹	𝕺	𝕻	𝕼	𝕽	𝕾	𝕿
U+1D580 – 1D58F	𝖀	𝖁	𝖂	𝖃	𝖄	𝖅	𝖆	𝖇	𝖈	𝖉	𝖊	𝖋	𝖌	𝖍	𝖎	𝖏
U+1D590 – 1D59F	𝖐	𝖑	𝖒	𝖓	𝖔	𝖕	𝖖	𝖗	𝖘	𝖙	𝖚	𝖛	𝖜	𝖝	𝖞	𝖟
U+1D5A0 – 1D5AF	𝖠	𝖡	𝖢	𝖣	𝖤	𝖥	𝖦	𝖧	𝖨	𝖩	𝖪	𝖫	𝖬	𝖭	𝖮	𝖯
U+1D5B0 – 1D5BF	𝖰	𝖱	𝖲	𝖳	𝖴	𝖵	𝖶	𝖷	𝖸	𝖹	𝖺	𝖻	𝖼	𝖽	𝖾	𝖿
U+1D5C0 – 1D5CF	𝗀	𝗁	𝗂	𝗃	𝗄	𝗅	𝗆	𝗇	𝗈	𝗉	𝗊	𝗋	𝗌	𝗍	𝗎	𝗏
U+1D5D0 – 1D5DF	𝗐	𝗑	𝗒	𝗓	𝗔	𝗕	𝗖	𝗗	𝗘	𝗙	𝗚	𝗛	𝗜	𝗝	𝗞	𝗟
U+1D5E0 – 1D5EF	𝗠	𝗡	𝗢	𝗣	𝗤	𝗥	𝗦	𝗧	𝗨	𝗩	𝗪	𝗫	𝗬	𝗭	𝗮	𝗯
U+1D5F0 – 1D5FF	𝗰	𝗱	𝗲	𝗳	𝗴	𝗵	𝗶	𝗷	𝗸	𝗹	𝗺	𝗻	𝗼	𝗽	𝗾	𝗿
U+1D600 – 1D60F	𝘀	𝘁	𝘂	𝘃	𝘄	𝘅	𝘆	𝘇	𝘈	𝘉	𝘊	𝘋	𝘌	𝘍	𝘎	𝘏
U+1D610 – 1D61F	𝘐	𝘑	𝘒	𝘓	𝘔	𝘕	𝘖	𝘗	𝘘	𝘙	𝘚	𝘛	𝘜	𝘝	𝘞	𝘟
U+1D620 – 1D62F	𝘠	𝘡	𝘢	𝘣	𝘤	𝘥	𝘦	𝘧	𝘨	𝘩	𝘪	𝘫	𝘬	𝘭	𝘮	𝘯
U+1D630 – 1D63F	𝘰	𝘱	𝘲	𝘳	𝘴	𝘵	𝘶	𝘷	𝘸	𝘹	𝘺	𝘻	𝘼	𝘽	𝘾	𝘿
U+1D640 – 1D64F	𝙀	𝙁	𝙂	𝙃	𝙄	𝙅	𝙆	𝙇	𝙈	𝙉	𝙊	𝙋	𝙌	𝙍	𝙎	𝙏
U+1D650 – 1D65F	𝙐	𝙑	𝙒	𝙓	𝙔	𝙕	𝙖	𝙗	𝙘	𝙙	𝙚	𝙛	𝙜	𝙝	𝙞	𝙟
U+1D660 – 1D66F	𝙠	𝙡	𝙢	𝙣	𝙤	𝙥	𝙦	𝙧	𝙨	𝙩	𝙪	𝙫	𝙬	𝙭	𝙮	𝙯
U+1D670 – 1D67F	𝙰	𝙱	𝙲	𝙳	𝙴	𝙵	𝙶	𝙷	𝙸	𝙹	𝙺	𝙻	𝙼	𝙽	𝙾	𝙿
U+1D680 – 1D68F	𝚀	𝚁	𝚂	𝚃	𝚄	𝚅	𝚆	𝚇	𝚈	𝚉	𝚊	𝚋	𝚌	𝚍	𝚎	𝚏
U+1D690 – 1D69F	𝚐	𝚑	𝚒	𝚓	𝚔	𝚕	𝚖	𝚗	𝚘	𝚙	𝚚	𝚛	𝚜	𝚝	𝚞	𝚟
U+1D6A0 – 1D6AF	𝚠	𝚡	𝚢	𝚣	𝚤	𝚥	–	–	𝚨	𝚩	𝚪	𝚫	𝚬	𝚭	𝚮	𝚯
U+1D6B0 – 1D6BF	𝚰	𝚱	𝚲	𝚳	𝚴	𝚵	𝚶	𝚷	𝚸	𝚹	𝚺	𝚻	𝚼	𝚽	𝚾	𝚿
U+1D6C0 – 1D6CF	𝛀	𝛁	𝛂	𝛃	𝛄	𝛅	𝛆	𝛇	𝛈	𝛉	𝛊	𝛋	𝛌	𝛍	𝛎	𝛏
U+1D6D0 – 1D6DF	𝛐	𝛑	𝛒	𝛓	𝛔	𝛕	𝛖	𝛗	𝛘	𝛙	𝛚	𝛛	𝛜	𝛝	𝛞	𝛟
U+1D6E0 – 1D6EF	𝛠	𝛡	𝛢	𝛣	𝛤	𝛥	𝛦	𝛧	𝛨	𝛩	𝛪	𝛫	𝛬	𝛭	𝛮	𝛯
U+1D6F0 – 1D6FF	𝛰	𝛱	𝛲	𝛳	𝛴	𝛵	𝛶	𝛷	𝛸	𝛹	𝛺	𝛻	𝛼	𝛽	𝛾	𝛿
U+1D700 – 1D70F	𝜀	𝜁	𝜂	𝜃	𝜄	𝜅	𝜆	𝜇	𝜈	𝜉	𝜊	𝜋	𝜌	𝜍	𝜎	𝜏
U+1D710 – 1D71F	𝜐	𝜑	𝜒	𝜓	𝜔	𝜕	𝜖	𝜗	𝜘	𝜙	𝜚	𝜛	𝜜	𝜝	𝜞	𝜟
U+1D720 – 1D72F	𝜠	𝜡	𝜢	𝜣	𝜤	𝜥	𝜦	𝜧	𝜨	𝜩	𝜪	𝜫	𝜬	𝜭	𝜮	𝜯
U+1D730 – 1D73F	𝜰	𝜱	𝜲	𝜳	𝜴	𝜵	𝜶	𝜷	𝜸	𝜹	𝜺	𝜻	𝜼	𝜽	𝜾	𝜿
U+1D740 – 1D74F	𝝀	𝝁	𝝂	𝝃	𝝄	𝝅	𝝆	𝝇	𝝈	𝝉	𝝊	𝝋	𝝌	𝝍	𝝎	𝝏
U+1D750 – 1D75F	𝝐	𝝑	𝝒	𝝓	𝝔	𝝕	𝝖	𝝗	𝝘	𝝙	𝝚	𝝛	𝝜	𝝝	𝝞	𝝟
U+1D760 – 1D76F	𝝠	𝝡	𝝢	𝝣	𝝤	𝝥	𝝦	𝝧	𝝨	𝝩	𝝪	𝝫	𝝬	𝝭	𝝮	𝝯
U+1D770 – 1D77F	𝝰	𝝱	𝝲	𝝳	𝝴	𝝵	𝝶	𝝷	𝝸	𝝹	𝝺	𝝻	𝝼	𝝽	𝝾	𝝿
U+1D780 – 1D78F	𝞀	𝞁	𝞂	𝞃	𝞄	𝞅	𝞆	𝞇	𝞈	𝞉	𝞊	𝞋	𝞌	𝞍	𝞎	𝞏
U+1D790 – 1D79F	𝞐	𝞑	𝞒	𝞓	𝞔	𝞕	𝞖	𝞗	𝞘	𝞙	𝞚	𝞛	𝞜	𝞝	𝞞	𝞟
U+1D7A0 – 1D7AF	𝞠	𝞡	𝞢	𝞣	𝞤	𝞥	𝞦	𝞧	𝞨	𝞩	𝞪	𝞫	𝞬	𝞭	𝞮	𝞯
U+1D7B0 – 1D7BF	𝞰	𝞱	𝞲	𝞳	𝞴	𝞵	𝞶	𝞷	𝞸	𝞹	𝞺	𝞻	𝞼	𝞽	𝞾	𝞿
U+1D7C0 – 1D7CF	𝟀	𝟁	𝟂	𝟃	𝟄	𝟅	𝟆	𝟇	𝟈	𝟉	𝟊	𝟋	–	–	𝟎	𝟏
U+1D7D0 – 1D7DF	𝟐	𝟑	𝟒	𝟓	𝟔	𝟕	𝟖	𝟗	𝟘	𝟙	𝟚	𝟛	𝟜	𝟝	𝟞	𝟟
U+1D7E0 – 1D7EF	𝟠	𝟡	𝟢	𝟣	𝟤	𝟥	𝟦	𝟧	𝟨	𝟩	𝟪	𝟫	𝟬	𝟭	𝟮	𝟯
U+1D7F0 – 1D7FF	𝟰	𝟱	𝟲	𝟳	𝟴	𝟵	𝟶	𝟷	𝟸	𝟹	𝟺	𝟻	𝟼	𝟽	𝟾	𝟿

Arabic Mathematical Alphabetic Symbols

U+1EE00 – 1EE0F	ا	ب	ج	د	د	و	ز	ح	ط	ي	ك	ل	م	ن	س	ع
U+1EE10 – 1EE1F	ف	ص	ق	ر	ش	ت	ث	خ	ذ	ض	ظ	غ	ٮ	ں	ڡ	ٯ
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Table 6: LM Math vs. NewCM Math, *cont.*

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+1EE20 – 1EE2F	𞸠	𞸡	𞸢	–	𞸤	–	–	𞸧	–	𞸩	𞸪	𞸫	𞸬	𞸭	𞸮	𞸯
U+1EE30 – 1EE3F	𞸰	𞸱	𞸲	–	𞸴	𞸵	𞸶	𞸷	–	𞸹	–	𞸻	–	–	–	–
U+1EE40 – 1EE4F	–	–	𞹂	–	–	–	–	𞹇	–	𞹉	–	𞹋	–	𞹍	𞹎	𞹏
U+1EE50 – 1EE5F	–	𞹑	𞹒	–	𞹔	–	–	𞹗	–	𞹙	–	𞹛	–	𞹝	–	𞹟
U+1EE60 – 1EE6F	–	𞹡	𞹢	–	𞹤	–	–	𞹧	𞹨	𞹩	𞹪	–	𞹬	𞹭	𞹮	𞹯
U+1EE70 – 1EE7F	𞹰	𞹱	𞹲	–	𞹴	𞹵	𞹶	𞹷	–	𞹹	𞹺	𞹻	𞹼	–	𞹾	𞹿
U+1EE80 – 1EE8F	𞺀	𞺁	𞺂	𞺃	𞺄	𞺅	𞺆	𞺇	𞺈	𞺉	–	𞺋	𞺌	𞺍	𞺎	𞺏
U+1EE90 – 1EE9F	𞺐	𞺑	𞺒	𞺓	𞺔	𞺕	𞺖	𞺗	𞺘	𞺙	𞺚	𞺛	–	–	–	–
U+1EEA0 – 1EEAF	–	𞺡	𞺢	𞺣	–	𞺥	𞺦	𞺧	𞺨	𞺩	–	𞺫	𞺬	𞺭	𞺮	𞺯
U+1EEB0 – 1EEBF	𞺰	𞺱	𞺲	𞺳	𞺴	𞺵	𞺶	𞺷	𞺸	𞺹	𞺺	𞺻	–	–	–	–
U+1EEF0 – 1EEFF	𞻰	𞻱	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Geometric Shapes Extended

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+1F780 – 1F78F	🞀	🞁	🞂	🞃	🞄	🞅	🞆	🞇	🞈	🞉	🞊	🞋	🞌	🞍	🞎	🞏
U+1F790 – 1F79F	🞐	🞑	🞒	🞓	🞔	🞕	🞖	🞗	🞘	🞙	🞚	🞛	🞜	🞝	🞞	🞟
U+1F7A0 – 1F7AF	🞠	🞡	🞢	🞣	🞤	🞥	🞦	🞧	🞨	🞩	🞪	🞫	🞬	🞭	🞮	🞯
U+1F7B0 – 1F7BF	🞰	🞱	🞲	🞳	🞴	🞵	🞶	🞷	🞸	🞹	🞺	🞻	🞼	🞽	🞾	🞿
U+1F7C0 – 1F7CF	🟀	🟁	🟂	🟃	🟄	🟅	🟆	🟇	🟈	🟉	🟊	🟋	🟌	🟍	🟎	🟏
U+1F7D0 – 1F7DF	🟐	🟑	🟒	🟓	🟔	🟕	🟖	🟗	🟘	–	–	–	–	–	–	–
U+1F7E0 – 1F7EF	🟠	🟡	🟢	🟣	🟤	🟥	🟦	🟧	🟨	🟩	🟪	🟫	–	–	–	–

Supplemental Arrows-C

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+1F800 – 1F80F	🠀	🠁	🠂	🠃	🠄	🠅	🠆	🠇	🠈	🠉	🠊	🠋	–	–	–	–
U+1F810 – 1F81F	🠐	🠑	🠒	🠓	🠔	🠕	🠖	🠗	🠘	🠙	🠚	🠛	🠜	🠝	🠞	🠟
U+1F820 – 1F82F	🠠	🠡	🠢	🠣	🠤	🠥	🠦	🠧	🠨	🠩	🠪	🠫	🠬	🠭	🠮	🠯
U+1F830 – 1F83F	🠰	🠱	🠲	🠳	🠴	🠵	🠶	🠷	🠸	🠹	🠺	🠻	🠼	🠽	🠾	🠿
U+1F840 – 1F84F	🡀	🡁	🡂	🡃	🡄	🡅	🡆	🡇	–	–	–	–	–	–	–	–
U+1F850 – 1F85F	🡐	🡑	🡒	🡓	🡔	🡕	🡖	🡗	🡘	🡙	–	–	–	–	–	–
U+1F860 – 1F86F	🡠	🡡	🡢	🡣	🡤	🡥	🡦	🡧	🡨	🡩	🡪	🡫	🡬	🡭	🡮	🡯
U+1F870 – 1F87F	🡰	🡱	🡲	🡳	🡴	🡵	🡶	🡷	🡸	🡹	🡺	🡻	🡼	🡽	🡾	🡿
U+1F880 – 1F88F	🢀	🢁	🢂	🢃	🢄	🢅	🢆	🢇	–	–	–	–	–	–	–	–
U+1F890 – 1F89F	🢐	🢑	🢒	🢓	🢔	🢕	🢖	🢗	🢘	🢙	🢚	🢛	🢜	🢝	🢞	🢟
U+1F8A0 – 1F8AF	🢠	🢡	🢢	🢣	🢤	🢥	🢦	🢧	🢨	🢩	🢪	🢫	🢬	🢭	–	–
U+1F8B0 – 1F8BF	🢰	🢱	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Total number of glyphs in `Latin Modern Math`: 2046
Comparison font `New Computer Modern Math` has 0 missing and 1958 extra glyphs

B.5 Garamond Libre's Byzantine Musical Symbols

As a final example we exhibit the Byzantine Musical Symbols as provided by Garamond Libre. Command used:

```
\displayfonttable[range-start=1D000, range-end=1D0FF,
                  hex-digits=block,
                  missing-glyph-color=black!50, color=black!75,
                  statistics-format=Total number of glyphs in
                    this block of #1 is #2]
                 {Garamond Libre}
```

Note that we have altered the text produced by the statistics, because the default is somewhat misleading if only a portion of the font is displayed. This produces the following table:

Table 7: Garamond Libre

Byzantine Musical Symbols

U+1D000 - 1D00F	𝀀	𝀁	𝀂	𝀃	𝀄	𝀅	𝀆	𝀇	𝀈	𝀉	𝀊	𝀋	𝀌	𝀍	𝀎	𝀏
U+1D010 - 1D01F	𝀐	𝀑	𝀒	𝀓	𝀔	𝀕	𝀖	𝀗	𝀘	𝀙	𝀚	𝀛	𝀜	𝀝	𝀞	𝀟
U+1D020 - 1D02F	𝀠	𝀡	𝀢	𝀣	𝀤	𝀥	𝀦	𝀧	𝀨	𝀩	𝀪	𝀫	𝀬	𝀭	𝀮	𝀯
U+1D030 - 1D03F	𝀰	𝀱	𝀲	𝀳	𝀴	𝀵	𝀶	𝀷	𝀸	𝀹	𝀺	𝀻	𝀼	𝀽	𝀾	𝀿
U+1D040 - 1D04F	𝁀	𝁁	𝁂	𝁃	𝁄	𝁅	𝁆	𝁇	𝁈	𝁉	𝁊	𝁋	𝁌	𝁍	𝁎	𝁏
U+1D050 - 1D05F	𝁐	𝁑	𝁒	𝁓	𝁔	𝁕	𝁖	𝁗	𝁘	𝁙	𝁚	𝁛	𝁜	𝁝	𝁞	𝁟
U+1D060 - 1D06F	𝁠	𝁡	𝁢	𝁣	𝁤	𝁥	𝁦	𝁧	𝁨	𝁩	𝁪	𝁫	𝁬	𝁭	𝁮	𝁯
U+1D070 - 1D07F	𝁰	𝁱	𝁲	𝁳	𝁴	𝁵	𝁶	𝁷	𝁸	𝁹	𝁺	𝁻	𝁼	𝁽	𝁾	𝁿
U+1D080 - 1D08F	𝂀	𝂁	𝂂	𝂃	𝂄	𝂅	𝂆	𝂇	𝂈	𝂉	𝂊	𝂋	𝂌	𝂍	𝂎	𝂏
U+1D090 - 1D09F	𝂐	𝂑	𝂒	𝂓	𝂔	𝂕	𝂖	𝂗	𝂘	𝂙	𝂚	𝂛	𝂜	𝂝	𝂞	𝂟
U+1D0A0 - 1D0AF	𝂠	𝂡	𝂢	𝂣	𝂤	𝂥	𝂦	𝂧	𝂨	𝂩	𝂪	𝂫	𝂬	𝂭	𝂮	𝂯
U+1D0B0 - 1D0BF	𝂰	𝂱	𝂲	𝂳	𝂴	𝂵	𝂶	𝂷	𝂸	𝂹	𝂺	𝂻	𝂼	𝂽	𝂾	𝂿
U+1D0C0 - 1D0CF	𝃀	𝃁	𝃂	𝃃	𝃄	𝃅	𝃆	𝃇	𝃈	𝃉	𝃊	𝃋	𝃌	𝃍	𝃎	𝃏
U+1D0D0 - 1D0DF	𝃐	𝃑	𝃒	𝃓	𝃔	𝃕	𝃖	𝃗	𝃘	𝃙	𝃚	𝃛	𝃜	𝃝	𝃞	𝃟
U+1D0E0 - 1D0EF	𝃠	𝃡	𝃢	𝃣	𝃤	𝃥	𝃦	𝃧	𝃨	𝃩	𝃪	𝃫	𝃬	𝃭	𝃮	𝃯
U+1D0F0 - 1D0FF	𝃰	𝃱	𝃲	𝃳	𝃴	𝃵	–	–	–	–	–	–	–	–	–	–
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Total number of glyphs in this block of Garamond Libre is 246

◇ Frank Mittelbach
Mainz, Germany
https://www.latex-project.org
https://ctan.org/pkg/unicodefonttable

LaTeX News

Issue 34, November 2021

Contents

Introduction 1

Hook business 1
- Provide `\ActivateGenericHook` 1
- Standardized names for the generic hooks . . . 2
- Some file hooks made one-time 2
- Clearing extra hook code for the next invocation 2
- Cleaning up after `\UseOneTimeHook` 2
- `\RemoveFromHook` with a missing code label . . 2
- Patching commands with parameter tokens . . 2

New or improved commands 3
- `\NewCommandCopy` and `\ShowCommand` extended 3
- Undo math alphabet allocations if necessary . . 3
- New default value for `\tracinglostchars` . . . 3
- `\PackageNote` and `\ClassNote` added 3
- New `\ShowFloat` command 3
- New argument for `\counterwithin/without` . 3
- Tests for package and class loading 4
- Better handling for a misuse of `\include` . . . 4

Code improvements 4
- Use OpenType version of Latin Modern Upright Italic font 4
- Additional Extended Latin characters predefined 4
- Check `\endfoo` in `\NewDocumentEnvironment` . 4
- Improve the error message `\begin ended by ...` 4
- Pick up all arguments to `\contentsline` . . . 4
- Allow dropping a math list in LuaTeX callback 4
- Extended label handling in package code 4
- Better message if text accent used in math mode 4

Bug fixes 5
- Replicate argument processors for all embellishments in command declarations . 5
- Correct case changing of `\ij` and `\IJ` 5
- Legacy font series default changes 5
- Use of `#` in `\textbf` and similar commands . . 5

Changes to packages in the amsmath category 5
- Improved compatibility with hyperref 5

Changes to packages in the graphics category 5
- graphicx: New key, for alt text 5

Changes to packages in the tools category 5
- array: No `\mathsurround` around a `tabular` . . 5
- longtable: Improvements after a section heading 5
- multicol: Better column break control 5
- varioref: Improved handling of missing labels . 6

Introduction

This release of LaTeX does not contain any major new modules, but is focused around consolidation and improvements of the functionality introduced in previous releases. In addition, various smaller enhancements and bug fixes have been added to the kernel and the core packages.

Hook business

Since the introduction of the hook management system in the 2020 release of LaTeX [4] package developers have started to make more and more use of this new functionality. One result of this increased activity has been a number of queries which show that some of the documentation was not precise enough and that some clarifications were needed; these deficiencies have now been addressed in the documentation. The increased usage has also revealed a small number of errors that we thought should be corrected now, while the adoption rate is still relatively small; the following problems have therefore been addressed in this release.

Provide `\ActivateGenericHook`

The hook management system offers a number of generic hooks, i.e., hooks whose names contain a variable component such as the name of an environment. Predeclaring such hooks is not feasible, so these hooks use a different mechanism: they are implicitly available, springing into life the moment a package, or the document preamble, adds any code to one by using `\AddToHook`. The kernel offers such hooks for environments (`env/...`) and commands (`cmd/...`), and also for files, packages and classes (`file/...`, `include/...`, `package/...`, `class/...`).

It is also possible to offer such generic hooks in packages if, for example, hooks are needed that depend on the current language and therefore need the language name as part of the hook name (but you probably don't know beforehand all the necessary names).

If you want to offer such generic hooks, you can now do this by using `\UseHook` or `\UseOneTimeHook` in your (package) code, but *without declaring the hook* with `\NewHook`. However, without further work, a call to `\UseHook` with an undeclared hook name will do nothing; so, as an additional setup step, it is necessary to explicitly activate the generic hook by using `\ActivateGenericHook`.[1]

[1] Note that in the previous release we offered `\ProvideHook` as a means to achieve this effect, but the name was badly chosen so we decided to deprecate it and now offer `\ActivateGenericHook` instead.

doi.org/10.47397/tb/42-3/tb132ltnews34

Assuming that you don't know all the different hook names up front, it will remain the task of the users of your package to activate the hook themselves before adding code to it. For example, Babel offers hooks such as `babel/`⟨*language*⟩`/afterextras` that enable a user to add language specific declarations to these "extras". One can then write

```
\ActivateGenericHook
       {babel/ngerman/afterextras}
\AddToHook{babel/ngerman/afterextras}
       {\color{blue}}
```

after which all German words would be colored blue in the text.

Note that a generic hook produced in this way is always a normal hook.

Standardized names for the generic hooks

The initial set of generic hooks provided by the kernel had two patterns of names: ones like `env/`⟨*name*⟩`/after`, with the variable, ⟨*name*⟩, part in the middle position; and ones like `file/after/`⟨*name*⟩, with the variable part in the third position. The coexistence of these two types caused confusion because the user had to remember in which position the variable part was supposed to go; and it also made the code more complicated and slower.

The file-related hooks have therefore been renamed so that the variable part of the name is in the middle, as with all other hooks. The changes are listed here:

Old name	New name
`file/before/`⟨*name*⟩	→ `file/`⟨*name*⟩`/before`
`file/after/`⟨*name*⟩	→ `file/`⟨*name*⟩`/after`
`package/before/`⟨*name*⟩	→ `package/`⟨*name*⟩`/before`
`package/after/`⟨*name*⟩	→ `package/`⟨*name*⟩`/after`
`class/before/`⟨*name*⟩	→ `class/`⟨*name*⟩`/before`
`class/after/`⟨*name*⟩	→ `class/`⟨*name*⟩`/after`
`include/before/`⟨*name*⟩	→ `include/`⟨*name*⟩`/before`
`include/end/`⟨*name*⟩	→ `include/`⟨*name*⟩`/end`
`include/after/`⟨*name*⟩	→ `include/`⟨*name*⟩`/after`

Since this is a breaking change, the old names will still work for a while so that users and package authors have enough time to adjust; but a warning will be issued when the old names are used. Eventually the deprecated names will be turned into errors and then removed completely. *(github issue 648)*

Some file hooks made one-time

Classes, packages and included files can only be loaded once in a LaTeX document. For this reason, the hooks that are specific to loading such files have been made one-time hooks. Beside being more efficient, this supports the following important use case

```
\AddToHook{package/varioref/after}
  {... apply when the package gets loaded,
   or apply now (if it is already loaded) ...}
```

without the need to first test whether the package is already loaded. *(github issue 623)*

Clearing extra hook code for the next invocation

There are a few use cases where it is helpful if one can cancel an earlier use of `\AddToHookNext`: for example, when a page is discarded with `\DiscardShipoutBox` because only some pages of the document are printed. For such situations the new command `\ClearHookNext` is now provided. *(github issue 565)*

Cleaning up after `\UseOneTimeHook`

Some hooks are meant to be used only once in a document, and any further attempt to add code to one of these will cause the code to be executed immediately instead of being added to the hook. The initial implementation of this concept was very simple and didn't anticipate that packages may try to execute a one-time hook several times, resulting in the hook code being executed repeatedly. Thus the implementation was fine for simple cases (such as the `begindocument` hook) but it causes trouble if the one-time hook was intended, for example, as an initialization hook that is used just once (when a command is first called) but is then ignored in further calls.

This deficiency has been addressed, and now a one-time hook will only be executed once, with its code being removed after use to free up some memory. *(github issue 565)*

`\RemoveFromHook` with a missing code label

In the first version of `\RemoveFromHook`, when the code label to be removed didn't exist in the hook a "removal order" would be queued; and then, the next time something tried to add that label to the hook, this `\AddToHook` action would be cancelled by the removal order, so that no code would be added that one time. This was so that, in principle, package loading order wouldn't matter. However, this implementation didn't work as intended because, while two `\AddToHook` actions with a given label would be removed by a single `\RemoveFromHook`, one `\RemoveFromHook` could not cancel two `\AddToHook` actions for that label; this caused confusion and also led to further problems.

The implementation has now been changed, so that `\RemoveFromHook` removes only code labels that already exist in a hook: it will display a warning if there is no such code label.

Note that, whereas when working with a single package you should use `\RemoveFromHook` to remove a code label, when working with more than one package, the `voids` relation should preferably be used. This is best because this relation is non-destructive (meaning that it can be reverted later by using another relation), and it is also truly independent of package loading order. *(github issue 625)*

Patching commands with parameter tokens

In the last release, LaTeX's hook mechanism was extended to add support for hooking into commands using generic `cmd` hooks (see [5]). That version of the extension had a bug: the patching of some commands that contained a parameter token (normally `#`) in their definition would fail with a low-level TeX error. This has now been fixed so that patching now works for those commands as well. *(github issue 697)*

New or improved commands

`\NewCommandCopy` and `\ShowCommand` extended

Since the 2020-10-01 release (see [4]), LaTeX has provided `\NewCommandCopy` to copy robust commands, and `\ShowCommand` to show their definitions on the terminal. In that same release, the xparse package was integrated into the kernel (as ltcmd) to offer `\NewDocumentCommand`, etc. However, the extended support for `\NewCommandCopy` and `\ShowCommand` was not implemented in ltcmd. The present LaTeX release implements this support, so now commands defined with `\NewDocumentCommand` and friends can also be copied, and their definitions can be easily shown on the terminal without the need for "`\csname` gymnastics". *(github issue 569)*

Undo math alphabet allocations if necessary

TeX, or more exactly the 8-bit versions of TeX, such as pdfTeX, have a hard limit of 16 on the number of different math font groups (`\fam` or `\mathgroup`) that can be used in a single formula. For each symbol font declared (by a package or in the preamble) an extra math group is allocated, and the same happens for each math alphabet, (such as `\mathbf`) once it gets used anywhere in the document. Up to now, these math alphabet allocations were permanent, even if they were used only once; the result was that in complex documents you could easily run out of available math font groups. The only remedy for this was to define your own math version, which is a complicated and cumbersome process.

This situation has now been improved by the introduction of a new counter `localmathalphabets`: this counter governs how many of the math group slots are assigned locally when a new math alphabet (and a new math group) is needed. Once the current formula is finished, every such further (local) allocation is undone, giving you a fighting chance of being able to use different new math alphabets in the next formula.

The default value of `localmathalphabets` is 2, but if you need more local alphabets because of the complexity of your document, you can set this to a higher value such as 4 or 5. Setting it even higher is possible, but this would seldom be useful because many group slots will be taken up by symbol fonts and such slots are always permanently allocated, whether used or not. *(github issue 676)*

New default value for `\tracinglostchars`

In 2021 all TeX engines were enhanced so that `\tracinglostchars` supported the value `3` to turn missing characters into errors and not just warnings. This engine change made us realize that LaTeX should set a better default value for this parameter (previously, the warning was written only to the transcript file). Using the now available value of `3` as the default would be ideal, but for compatibility reasons we have only increased it to `2` in the kernel. However, we recommend setting `\tracinglostchars=3`, in either a package or the preamble of your documents: this is because having missing glyphs in the output is definitely an error and should therefore be flagged as such (to ensure that it gets proper attention). Further reasons, related especially to Unicode engines, for making this recommended change are explained later in this newsletter (in connection with the misuse of text accents in math mode).

`\PackageNote` and `\ClassNote` added

LaTeX offers these three commands: `\PackageError` to signal errors that stop the processing; `\PackageWarning` to generate a warning message on the terminal but continue with the processing; and `\PackageInfo` to provide some information that is only written to the `.log` file but not sent to the terminal. What has not existed up to now is a way to provide information on the terminal that identifies itself as coming from a specific package but which does not claim to be a warning. (Packages that wanted to write to the terminal used `\PackageWarning` even though the information was not in fact a warning.)

We have therefore now added `\PackageNote` (and the closely related `\PackageNoteNoLine`); these identify themselves as "informational", but they still go to the terminal and not only to the `.log` file. Similar commands exist for classes and so there too we have new commands: `\ClassNote` and `\ClassNoteNoLine`. *(github issue 613)*

New `\ShowFloat` command

The package fltrace offers a (fairly low-level but very detailed) way to trace LaTeX's float mechanism. This can help in understanding why a certain float is placed into a certain region, or why it shows up unexpectedly on a later page. LaTeX stores floats in registers named `\bx@A`, `\bx@B`, etc., and these names show up in the tracing information.

To display the contents of a float register, you can now say `\ShowFloat{`*identifier*`}` where *identifier* is the uppercase letter (or letters) after `bx@` in the register name shown in the tracing. If additional registers have been allocated (with `\extrafloats`), the *identifier* can also be a number. The command is generally available, whether or not you have loaded fltrace, because it is also useful when interpreting the tracing output of the fewerfloatpages package.

New argument for `\counterwithin/without`

The commands `\counterwithout` and `\counterwithin` each now has an additional optional argument, similar to that of the command `\numberwithin` from amsmath, for which these are now the preferred replacements. This optional argument specifies the format of the counter, such as `\roman`; the default value is `\arabic`.

Tests for package and class loading

To test whether a package has been loaded you can now use `\IfPackageLoadedTF` {⟨*package*⟩} {⟨*true*⟩} {⟨*false*⟩} and, based on the result, execute different code. It is also possible to check whether the package was loaded with certain options. This is done with `\IfPackageLoadedWithOptionsTF`. It takes four arguments: {⟨*package*⟩}{⟨*option-list*⟩}{⟨*true*⟩}{⟨*false*⟩}. It uses the ⟨*false*⟩ code if one or more options in the ⟨*option-list*⟩ were not specified when loading the package, or if the package has never been loaded. Both commands can be used anywhere in the document, i.e., they are not restricted to the preamble.[2]

For classes, similar commands, with `Package` replaced by `Class` in the name, are provided. *(github issue 621)*

Better handling for a misuse of `\include`

The command `\include` has by now been used quite often, but erroneously, to input a variety of files in the preamble of the document (before `\begin{document}`). Therefore LaTeX now warns about such bad use of `\include`. As a recovery action it will nevertheless input the specified file if it exists (this is as before). Note, however, that this is now done without any adjustments to the `.aux` file settings and without running the `\include` file hooks (only the generic file hooks from `\InputIfFileExists` are run). *(github issue 645)*

Code improvements

Use OpenType version of Latin Modern Upright Italic font

When a Latin Modern font is used with the TU encoding under XeTeX or LuaTeX and fontshape `ui` is requested, LaTeX now uses the OpenType version of the font instead of substituting the (T1-encoded) Type 1 version.

Additional Extended Latin characters predefined

More characters, such as ḱ (U+1E131), are now predefined and do not need a `\DeclareUnicodeCharacter` declaration. *(github issue 593)*

Check `\endfoo` *in* `\NewDocumentEnvironment`

The `\newenvironment` command has always checked that neither `\foo` nor `\endfoo` exists before creating a `foo` environment. In contrast (for historical reasons) the more recently introduced command `\NewDocumentEnvironment` checked only for `\foo`. The behavior of `\NewDocumentEnvironment` now aligns with that of `\newenvironment`, except that it gives distinct errors concerning the existence of `\foo` and `\endfoo`.

Improve the error message `\begin ended by ...`

In the past it was possible to get an error message along the lines of "`\begin{foo} ended by \end{foo}`". This could happen when the environment name was partly hidden inside a macro. It happened because the test was comparing the literal strings, whereas in the error message these got fully expanded. This has now been changed to show a more sensible error message. *(github issue 587)*

Pick up all arguments to `\contentsline`

A `\contentsline` command in the `.toc` file is always followed by four arguments, the last one being empty except when using the hyperref package. The `\contentsline` command itself only used the first three arguments and it relied on the fourth being empty (and thus doing no harm). But this assumption is not always correct: e.g., if you at first decide to load hyperref but then later you remove this loading from the preamble. So now all four arguments are picked up, with the fourth being saved away so that it can be used by hyperref. *(github issue 633)*

Allow dropping a math list in LuaTeX callback

The LuaTeX callbacks `pre_mlist_to_hlist_filter` and `post_mlist_to_hlist_filter` no longer create an error when the callback handler indicates removal of the entire math list. *(github issue 644)*

Extended label handling in package code

Since 2020, as noted in LaTeX News 32 [4], LaTeX has recorded the name of the counter associated with the current label in the internal command `\@currentcounter`. This facility (originally from the zref package of Heiko Oberdiek) can be used to generate prefixes such as "Figure" before the reference text, as long as the counter is not counting different objects in a single sequence (e.g., lemmas and theorems). In the most common cases the current label is set by `\refstepcounter`, which automatically stores the counter name; but some constructs (alignments and footnotes) may need to store the current label directly and so for these it is useful to update additionally `\@currentcounter` so as to store this counter name.

In this release both the footnote command in the kernel and also some of the environments in the amsmath package have been updated in this way. We encourage the maintainers of any class or package files that define `\@currentlabel` to also set `\@currentcounter` at the same point. *(github issue 300, 687)*

Better message if text accent used in math mode

Using text accents like `\^` in math does not work (and TeX explicitly provides math accents such as `\hat` for accessing such symbols in math mode). Therefore LaTeX issued a warning when such a wrongly placed accent was encountered and this was often followed by a strange, and apparently unrelated, low-level error. This has now been changed so that the message from this error is at least about accents, which we hope is less puzzling.

Discussion of such warnings or errors reminds us to reinforce here a recommendation from earlier in this newsletter, as part of the item on the value of `\tracinglostchars`. Using TeX implementations from 2020 onwards, any warning that concerns missing characters can be converted to an error by setting

[2]This is now also true for the corresponding internal commands, e.g., `\@ifpackageloaded`, that had this restriction in the past.

\tracinglostchars to 3; we therefore now recommend changing this setting to 3, especially for Unicode engines where such missing characters are common (because no font supports the full Unicode range). *(github issue 643)*

Bug fixes

Replicate argument processors for all embellishments in command declarations

There was a bug in ltcmd (formerly xparse) that caused commands to misbehave if they were defined with embellishments and argument processors. In that case, only one (possibly void) argument processor would be added to the full set of embellishment arguments, resulting in too few processors in some cases and thus leading to unpredictable behavior. This bug has been fixed by applying the same argument processors to all the embellishments in a set, so that a declaration like:

```
\NewDocumentCommand\foo{>{\TrimSpaces}e{_^}}
                       {(#1)[#2]}
\foo^{ a }_{ b }
```

will now correctly apply \TrimSpaces to both arguments. *(github issue 639)*

Correct case changing of \ij and \IJ

The ligatures “ij” and “IJ”, as used in Dutch, are available (for most TEX fonts) only when the commands \ij or \IJ are used, or when you enter them as the Unicode characters U+0133 or U+0132. However, when using OT1 or T1 encoded fonts in pdfTEX, the upper or lower casing with \MakeUppercase and \MakeLowercase would always fail regardless of the input method. This has now been corrected. At the same time we improved the hyphenation results for words containing this ligature (when using the OT1 encoding). *(github issue 658)*

Legacy font series default changes

In the past, changes to the font series defaults were made by directly altering \bfdefault or \mddefault. Since 2020 there is now \DeclareFontSeriesDefault that allows more granular control: with this declaration you can alter the default for individual meta font families by, for example, changing the bold setting only for the sans serif family, without changing it for \rmfamily or \ttfamily. See [3] for more details.

For backwards compatibility, changing \bfdefault with \renewcommand remained possible; if used, this alters the setting for all meta families in one go. This alteration cannot be done when the \renewcommand happens and it was therefore delayed until the next time \bfseries or \mdseries was executed. However, the problem with that approach was that any call to \DeclareFontSeriesDefault in the meantime was overwritten; thus, these two approaches didn’t work well in combination. There was a problem because older font packages use the legacy method while newer ones use \DeclareFontSeriesDefault.

This has now been resolved by changing \DeclareFontSeriesDefault to do any necessary resetting prior to setting the new defaults. *(github issue 663)*

Use of # in \textbf and similar commands

Previously you could not use the macro parameter character # in inline functions within the argument of \texbf or similar text font commands. An internal definition is now guarded with \unexpanded so that the use of # here no longer generates an error. *(github issue 665)*

Changes to packages in the amsmath category

Improved compatibility with hyperref

This change in amsmath fixes a spacing problem caused by the method used in hyperref to change the equation environment. For simplicity, an explicit, low-level (hence possibly temporary) patch has been added to amsmath: this consists of an extra, empty (hence invisible) \mathopen atom (with no mathematical meaning) at the start of the environment’s mathematical content. *(github issue 652)*

Changes to packages in the graphics category

graphicx: New key, for alt text

A new key, alt, has been added to \includegraphics to support the addition of descriptive text that is important for accessibility. This key is unused by default; it can be deployed by extension packages and it will provide useful support for other future possibilities. *(github issue 651)*

Changes to packages in the tools category

array: No \mathsurround around a tabular

A tabular environment is typeset (internally) as an array environment with special settings, and it therefore uses (hidden) math mode. Since it is not in fact a math formula, no extra space from \mathsuround should be added (the spacing around the tabular should not get changed). Note that this bug has been present “forever”, which shows that \mathsurround is never used, or at least its use is never noticed. At any rate, this bug has now finally been fixed. *(github issue 614)*

longtable: Improvements after a section heading

The longtable environment now sets the \@nobreakfalse flag to correct the typesetting when a table immediately follows a heading. Previously the spacing and indentation changes that are required immediately after a section heading were incorrectly triggered within the next paragraph (if any) following the table. A similar test for \if@noskipsec has been added, so that a table is correctly placed after a run-in heading rather than appearing before that heading. *(github issues 131 and 173)*

multicol: Better column break control

From version 1.9 onwards \columnbreak accepts an optional argument (like \pagebreak) in which you can specify the desirability of breaking the column after the current line: supported values are 0 to 4, with higher numbers indicating increased desirability. This version also adds \newcolumn, which forces a break but runs the column short (comparable to `\newpage` for pages). *(github issue 682)*

varioref: Improved handling of missing labels

If an undefined label is referenced, varioref makes a default definition so that later processing finds the right structure (two brace groups inside `\r@`⟨*label*⟩) However, if nameref or hyperref is loaded, this data structure changes to having five arguments; this could cause low-level errors in some cases. The code has therefore now been changed to avoid these errors. *(https://tex.stackexchange.com/603948)*

References

[1] Frank Mittelbach and Chris Rowley: *LaTeX Tagged PDF—A blueprint for a large project.* https://latex-project.org/publications/indexbyyear/2020/

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

[3] LaTeX Project Team: *LaTeX 2ε news 31.* https://latex-project.org/news/latex2e-news/ltnews31.pdf

[4] LaTeX Project Team: *LaTeX 2ε news 32.* https://latex-project.org/news/latex2e-news/ltnews32.pdf

[5] LaTeX Project Team: *LaTeX 2ε news 33.* https://latex-project.org/news/latex2e-news/ltnews33.pdf

LaTeX News #34

Production notes

Karl Berry

As you may have noticed, this issue mainly consists of a few longer-than-usual articles. We didn't plan it that way, but we were glad to have them, since the regular variety of shorter submissions did not happen for this issue. So we'd like to greatly encourage articles for the next issue, which will be in the spring of next year (deadline March 31, 2022)! Please, if you have any TeX-related projects or experiences you've been thinking about writing up, it's a good time. Early submissions are especially welcome.

A few TeXnical words about some of the articles in this issue. First, for Charles Bigelow's "Form, pattern & texture ...", the biggest complication was using the (unreleased) Lucida Book variant for the main font. I did this by the simple expedient of loading Lucida OpenType as usual, with `\setmainfont{LucidaBrightOT}`, but copying the Lucida Book `.otf` file into the current directory as `LucidaBrightOT.otf`.

For Amine Anane's article on Arabic typesetting, it was necessary to use Amine's modified (and renamed) `luahbtex` engine and `luaotfload` package. Fortunately, Amine had carefully documented his development process and I was able to make a binary without too much trouble. I built the `.fmt` file by hand, in the current directory, and then set `TEXMFDOTDIR=.//`, and unset `TEXMFSYSVAR`, to use that `.fmt` file and Amine's modified packages. So I'd like to thank Linas Stonys for suggesting the `TEXMFDOTDIR` feature for TeX Live, not so long ago.

For Herbert Voß's article on his remarkable `hvfloat` package, the main complication was ensuring that it started on an odd page; otherwise, the floats that spread across two pages would not have worked. A judicious choice for article ordering, plus a test `\ifodd\count0\else \errmessage{...}\fi` for explicit notification of the problem, did the trick. Of course figure placements here, as in all the articles (as in all articles), was a challenge, but Herbert had carefully designed his material to work for the *TUGboat* layout, for which I was extremely grateful.

If anyone is extra-curious, a few macro files and scripts used for *TUGboat* production are available from TUG's GitHub page, linked below.

I would like to thank all the authors, not just those mentioned above, for their invaluable assistance in finalizing their articles, even beyond writing them in the first place.

◇ Karl Berry
github.com/TeXUsersGroup

doi.org/10.47397/tb/42-3/tb132prod

A new unit for LMTX: The `dk`

Hans Hagen

At the ConTEXt 2021 meeting I mixed my TEX talks with showing some of the (upcoming) LuaMetaTEX source code. One evening we had a extension party where a new unit was implemented, the `dk`. This event was triggered by a remark Hraban [Ramm] made on the participants list in advance of the meeting, where he pointed to a Wikipedia article from which we quote:

> In issue 33, Mad published a partial table of the "Potrzebie System of Weights and Measures", developed by 19-year-old Donald E. Knuth, later a famed computer scientist. According to Knuth, the basis of this new revolutionary system is the potrzebie, which equals the thickness of Mad issue 26, or 2.2633484517438173216473 mm [. . .].

So, as the result of that session, the source code now has this comment:

> We support the Knuthian Potrzebie, cf. `en.wikipedia.org/wiki/Potrzebie`, as the `dk` unit. It was added on 2021-09-22 exactly when we crossed the season during an evening session at the 15th ConTEXt meeting in Bassenge (Boirs) Belgium.
> It took a few iterations to find the best numerator and denominator, but Taco Hoekwater, Harald Koenig and Mikael Sundqvist figured it out in this interactive session. The error messages have been adapted accordingly and the scanner in the Lua `tex` library also handles it. One `dk` is 6.43985pt. There is no need to make MetaPost aware of this unit because there it is just a numeric multiplier in a macro package.

When compared to the already present units the `dk` nicely fills a gap:

unit	points	scaled	visual
`sp`	0.00002	1	
`pt`	1.0	65536	
`bp`	1.00374	65781	
`dd`	1.07	70124	
`mm`	2.84526	186467	
`dk`	6.43985	422042	
`pc`	12.0	786432	
`cc`	12.8401	841489	
`cm`	28.45274	1864679	
`in`	72.26999	4736286	

Deep down, the unit scanner uses a numerator and denominator in order to map the given value onto the internally used scaled points, so the relevant snippet of C code is:

```
*num   = 49838; // 152940;
*denom =  7739; //  23749;
return normal_unit_scanned;
```

The impact on performance of scanning an additional unit can be neglected because the scanning code is a bit different from the code in LuaTEX and (probably the) other engines anyway.

Under consideration are a few extra units in the `relative_unit_scanned` category that we see in CSS: `vw` (relative to the `\hsize`), `vh` (relative to the `\vsize`), maybe a percentage (but of what) and `ch` (width of the current zero digit character). As usual with TEXies, once it's there it will be (ab)used.

◇ Hans Hagen
`http://pragma-ade.com`

doi.org/10.47397/tb/42-3/tb132hagen-dk

The Treasure Chest

These are the new packages posted to CTAN (`ctan.org`) from September–October 2021. 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 especially notable are starred (*); of course, this is not intended to slight the other contributions.

We hope this column helps people access the vast amount of material available through CTAN and the distributions. See also `ctan.org/topic`. Comments are welcome, as always.

◇ Karl Berry
`https://tug.org/TUGboat/Chest`

fonts

`bboldx` in `fonts`
More weights for blackboard board.

`nahuatl` in `fonts`
Render Nahuatl (native Mexican writing system) glyphs.

* `notocondensed` in `fonts`
Support for the condensed variants of the Noto fonts. This package is available through `https://contrib.texlive.info`.

graphics

`luapstricks` in `graphics/pstricks/contrib`
Use PSTricks in LuaLaTeX with no need for special environments or external commands.

`tikz-bagua` in `graphics/pgf/contrib`
Draw Yijing (I Ching) or Zhouyi symbols in Ti*k*Z.

`zx-calculus` in `graphics/pgf/contrib`
Ti*k*Z library to typeset ZX Calculus diagrams.

info

`tex-vpat` in `info`
TeX accessibility conformance report.

macros/latex/contrib

`cdcmd` in `macros/latex/contrib`
Expandable conditional commands for LaTeX.

`clicks` in `macros/latex/contrib`
Simulate animation in slide deck.

`debate` in `macros/latex/contrib`
Insert notes in the form of dialogues.

`linenoamsmath` in `macros/latex/contrib`
Use `lineno` package together with `amsmath`.

`phfcc` in `macros/latex/contrib`
Inline commenting for collaborative documents.

`phfextendedabstract` in `macros/latex/contrib`
Typeset extended abstracts, e.g., for conferences.

* `unicodefonttable` in `macros/latex/contrib`
Font tables for Unicode fonts. See articles in this issue.

`uwa-colours` in `macros/latex/contrib`
Colour palette of the Univ. of Western Australia.

macros/luatex/generic

`lua-widow-control` in `macros/luatex/generic`
Automatically remove widows and orphans from any document.

`lutabulartools` in `macros/luatex/generic`
Support commands for `tabular` material.

`penlight` in `macros/luatex/generic`
The Penlight pure-Lua library for LuaLaTeX.

macros/luatex/latex

`truthtable` in `macros/luatex/latex`
Automatically generate truth tables for given variables and statements.

`yamlvars` in `macros/luatex/latex`
YAML parser (Lua package `tinyyaml`) and support functions to make LaTeX definitions using YAML.

macros/unicodetex/latex

`njuthesis` in `macros/unicodetex/latex`
LaTeX thesis template for Nanjing University.

`uwa-letterhead` in `macros/unicodetex/latex`
Letterhead of the Univ. of Western Australia.

`uwa-pcf` in `macros/unicodetex/latex`
Participant Consent Form for human research protocols at the Univ. of Western Australia.

`uwa-pif` in `macros/unicodetex/latex`
Participant Information Form for same.

macros/xetex/latex

`hanzibox` in `macros/xetex/latex`
Simplify input of Chinese characters.

`zitie` in `macros/xetex/latex`
CJK character calligraphy practice sheets.

`macros/latex/contrib/cdcmd`

doi.org/10.47397/tb/42-3/tb132chest

Abstracts

La Lettre GUTenberg 41–44, 2020–2021

La Lettre GUTenberg is a publication of GUTenberg, the French-language TEX user group (`www.gutenberg.eu.org`).

La Lettre GUTenberg #41

Published December 18, 2020.

Patrick Bideault & Céline Chevalier, Éditorial [Editorial]; pp. 1–2

Céline Chevalier, Journée GUTenberg 2020 [GUTenberg 2020 meeting]; p. 2

A summary of the GUTenberg 2020 meeting, the first held since June 2013. A record of the discussions is published in the same issue. The proceedings of the conference have not yet been published.

Arthur Reutenauer, Compte rendu de l'assemblée générale du 14 novembre 2020 [GUTenberg 2020 annual meeting]; pp. 2–4

The report of the discussions and elections, by the meeting's chairman.

Céline Chevalier, Compte rendu du conseil d'administration du 21 novembre 2020 [Report of the board's meeting]; pp. 5–6

The report of the first meeting of the newly-elected board.

Maxime Chupin, Décomposition en série de Fourier d'un caract ère avec LuaTEX et MPLIB [Animating Fourier series decomposition of a character with LuaTEX and MPLIB]; pp. 6–16

The English translation of this article was published in TUGboat 42:1, 2021, pages 67–71.

Jérémy Just, Campus du libre, édition 2020 [Free software campus 2020 meeting]; p. 16

A brief report on GUTenberg's presence at this annual French free software community gathering.

Maxime Chupin, Notes de lecture [Book review]; p. 17

A review about Christian Tellechea's book *Apprendre à programmer en TEX* (*How to program with TEX*). This book is now a CTAN package.

La Lettre GUTenberg #42

Published March 13, 2021.

Patrick Bideault & Céline Chevalier, Éditorial [Editorial]; pp. 1–2

Jérémy Just, En souvenir d'Hervé Choplin [R.I.P. Hervé Choplin]; p. 2

Céline Chevalier, Moment d'échange avec les adhérents [Meeting with GUTenberg's members]; pp. 3–12

A report about an online meeting held at the end of January. The talks were about the projects developed by GUTenberg.

Maxime Chupin, Compte rendu du conseil d'administration du 8 février 2021 [The report of the board's meeting on February 8]; pp. 12–15

Maxime Chupin, Une très courte histoire de notre logo [A brief history of our logo]; pp. 15–16

Denis Bitouzé, Et maintenant, une bonne *vieille* veille technologique ! [Technology watch]; pp. 17–21

New CTAN packages, November 2020–March 2021.

Patrick Bideault, La fonte de ce numéro : `kpfonts-otf` [This issue's font: `kpfonts-otf`]; p. 22

A brief presentation about the OTF conversion of the KP-fonts, used for this issue.

Christian Hinque, TEX et moi : mon système de production, structure et nextcloud [TEX and I: my personal production système, file structure and nextcloud]; pp. 23–28

The author reports about the way he uses nextcloud to access his `.tex` files on several devices.

Patrick Bideault, Dernières nouvelles : un compilateur en ligne sur `texnique.fr` ! [Breaking news: online compilation available on `texnique.fr`!]; pp. 28–29

A report about the online compilation tool `texlive.net`, now usable from the French Q&A website `texnique.fr`.

Patrick Bideault, De l'importance des cotisations [About the significance of financial contributions.]; p. 30

La Lettre GUTenberg #43

Published April 10, 2021.

Patrick Bideault, Éditorial [Editorial]; pp. 1–2

GUTenberg, Journée GUTenberg 2021 [GUTenberg 2021 conference program]; pp. 2–4

Announcement of the upcoming GUTenberg conference with presentation of its program.

Maxime Chupin, Bilan moral : novembre 2020 — avril 2021 [Activity report, November 2020–April 2021]; pp. 5–7

Flora Vern, Rapport financier pour l'année 2020 [Financial report 2020]; pp. 8–9

Flora Vern, Proposition de statuts pour GUTenberg [Proposal for GUTenberg's new bylaws]; pp. 9–16

Flora Vern, the treasurer of GUTenberg, is a lawyer. She publishes a draft of new bylaws, hopefully more suitable, to be discussed by the members.

Jean-Michel Hufflen, Du côté d'autres groupes d'utilisateurs de TEX [Other TEX user groups' activities]; pp. 17–18

Reports about the GuIT and DANTE annual meetings.

Denis Bitouzé, Et maintenant, une bonne *vieille* veille technologique ! [Technology watch]; pp. 18–21

New CTAN packages, March–April 2021.

Donald E. Knuth, La mise au point de TEX de 2021 [The TEX tuneup of 2021]; pp. 22–26

The French translation, by Maxime Chupin, of the article published in *TUGboat* 42:1, 2021, pages 7–10.

Patrick Bideault, La fonte de ce numéro : Schola [This issue's font: Schola]; pp. 28–31

About Schola and the other TEX Gyre fonts provided by the GUST e-foundry.

Antoine Leblanc, TEX et moi : à propos d'un tableau... [TEX and I: About a table]; pp. 32–33

About TEX, chemistry, beautiful documents and the relationship between a father and his daughter.

La Lettre GUTenberg #44

Published August 12, 2021. Without modifying its design, the document class was completely renewed by Denis Bitouzé.

Patrick Bideault, Éditorial [Editorial]; pp. 1–2

Denis Bitouzé, Compte rendu des conférences de la journée GUTenberg [GUTenberg 2021 proceedings]; pp. 2–16

Report on the three talks at the 2021 meeting:

- *Literate programming with org-mode*, by Fabrice Niessen
- *New PDF support for LATEX*, by Ulrike Fischer
- *Introduction to the TikZducks and TikZlings packages*, by samcarter

Céline Chevalier, Compte rendu de l'assemblée générale de l'association GUTenberg [GUTenberg 2021 annual meeting notes]; pp. 16–25

Patrick Bideault, Les différents travaux de l'association [The various works of the group]; pp. 26–27

Patrick Bideault, Quelques nouvelles de Pologne [Some news about Poland]; p. 28

After the article in the previous issue, a follow-up report about the GUST e-foundry.

François Pantigny, Aperçu du package `nicematrix` [Overview of the `nicematrix` package]; pp. 29–36

GUTenberg, Publicité [Advertising]; p. 37

Patrick Bideault, Denis Bitouzé, Maxime Chupin, Et maintenant, une bonne *vieille* veille technologique ! [Technology watch]; pp. 38–44

New CTAN packages, April–August 2021.

Harald Lichtenstein, Utiliser des boucles avec TikZ [How to use loops with TikZ]; pp. 45–49

The French translation, by Patrick Bideault, of the article *Schleifen in TikZ* published in *Die TEXnische Komödie* 2021/2, pages 48–52.

Patrick Bideault, Brèves typographiques [Typographic news]; pp. 50–51

About the international graphic design biennial exhibition in Chaumont, France; R.I.P. Wolfgang Weingart.

Maxime Chupin & Patrick Bideault, La fonte de ce numéro : Libertinus [This issue's font: Libertinus]; pp. 52–55

Small review of the Libertinus font, its history and presentation of some of its properties.

Jérémy Just, Compte rendu de lecture [Book review]; p. 56

About Christophe Aubry's refcard *LATEX – Conception de documents élaborés et structurés*, ENI, 2021.

Patrick Bideault, Compte rendu de lecture [Book review]; pp. 56–57

About Fred Smeijers' book *Les Contrepoinçons* (*Counterpunch*, Hyphen Press, 1996), published in French in 2014 by B42.

[Received from Patrick Bideault.]

Les Cahiers GUTenberg 58 (2021)

Les Cahiers GUTenberg is the journal of GUTenberg, the French-language TeX user group (`gutenberg.eu.org`). After a break of some years, it has resumed publication with a new editor and a new committee.

Jean-Michel Hufflen, Éditorial [Editorial]; pp. 3–4

Sandrine Chenevez, Myriam Hamla, Stéphanie Louison, Donald Knuth : des mathématiques à la typographie [Donald Knuth: From mathematics to typography]; pp. 5–33

This article traces first the main stages of Donald Ervin Knuth's life, then his contributions to computer science and typography, mainly through his book series *The Art of Computer Programming* and the TeX/METAFONT pair. Reading it does not require special technical knowledge.

Peter Flynn, Imprimer ou ne pas imprimer [To print or not to print]; pp. 35–40

Translation by Patrick Bideault of this item published in *TUGboat* 41:3 (2021).

Christophe Poulain, LaTeX au collège ? Une avancée [LaTeX at middle school? Some advances]; pp. 41–54

Commands that ease the writing of theorems, as part of teaching mathematics in middle school, are shown. All these commands are grouped into the `ProfCollege` package, available on CTAN.

Maxime Chupin, LuaLaTeX et METAPOST avec `luamplib`, une introduction [An introduction to LuaLaTeX and METAPOST with `luamplib`]; pp. 55–80

This article is an introduction to the use of the `mplib` library of LuaTeX, which is an embedded version of the METAPOST program. This library can be used with the LaTeX format thanks to the `luamplib` package and we present its main features. This package has become mature and suitable for intensive use, and we illustrate this with some examples.

Jean-Michel Hufflen, Histoire des altérations musicales [History of accidentals in music]; pp. 81–105

Revised and extended translation by the author of the article published in *TUGboat* 38:2 (2017), pp. 147–156.

[Received from Jean-Michel Hufflen.]

ConTeXt Group Journal 2020

The ConTeXt Group publishes proceedings of the ConTeXt meetings: `articles.contextgarden.net`.

Henning Hraban Ramm, Editorial Note; p. 4

Day Plan; pp. 5–6

Hans Hagen, MkII MkIV LMTX: where does it end?; pp. 8–9

This meeting was special because of the COVID situation. It forced us to adapt and think about how to deal with this. But, we had a very nice meeting as usual. The first talk was a summary of where we started and where we are now. The other talks were more specialized.

Hans Hagen, LuaMetaTeX: where do we stand?; pp. 10–13

History and overview of the LMTX engine.

Hans Hagen, Extensions related to programming macros; pp. 14–28

Published in *TUGboat* 42:1.

Hans Hagen, Low-level: tokens; pp. 29–40

Detailed description of token parsing in LMTX and original TeX.

Hans Hagen, ECMAScript: just because it can be done; pp. 41–44

Using the standalone ECMAScript interpreter from `mupdf` in LMTX. The document interface is somewhat similar to handling Lua.

Hans Hagen, Lost in fonts; pp. 45–47

A case study of restoring ligature support in LMTX, with no TFM loaded.

Hans Hagen, Simple fonts with MetaFun; pp. 48–52

Defining shapes using MetaPost and MetaFun and adding them as new glyphs in fonts.

Taco Hoekwater, MetaPost definitions; pp. 53–72

Writing definitions in MetaPost: macros, operators, variables, grouping, and more.

Mikael P. Sundqvist, Scrutinized paths: A new path transformation in MetaPost; pp. 73–81

In this article, I discuss a problem that occurred while trying to find `directionpoint`s of joined paths in MetaPost. We found that when joining two paths where the final point of the first path is the same as, or at least very close to, the first point of the second path, numerical problems might appear. Finally, we present different solutions that appeared on the ConTeXt mailing list on how to avoid the issue; the most generic solutions work by sanitizing the joined graph. In fact, this resulted in a new path transformation called `scrutinized`.

Hans Hagen, UTF-8 in MetaPost; pp. 82–86
Using UTF-8 in MetaPost source code.

Hans Hagen, SVG graphics: some demos and discussion; pp. 87–88
Overview and examples of scalable vector graphics (SVG) images.

Michal Vlásak, Multimedia, PDF and ConTEXt; pp. 89–96
The possibility of inserting multimedia (audio, video, 3D) into PDF files has been here for a long time, albeit in different forms. Traditionally ConTEXt has had support for it, but the support in PDF viewers was dubious. What is the situation today, and is it worth all the hassle?

Willi Egger, Book production; pp. 97–106
Producing a book is comparable to making a wooden window. Yes, in both cases you need a plan/strategy, a list of requirements, and information in order to be able to get the desired result. After presenting the window project there are a couple of thoughts gathered which concern the work of the typographer, the printer and the bookbinder. With regard to the typographer using ConTEXt, three examples of special cases are described and illustrated with pictures and sample code.

Ton Otten, Proofing and production with ConTEXt; pp. 107–117
ConTEXt has several built-in tools that can help you during the proofing stage of your project. Furthermore there are a few ConTEXt features you can invoke in your style that gives you visual feedback of inconsistencies or errors. And then there are modes. Below I will describe a number of the techniques I use in the proofing stage of the Math4All project.

Harald König, 14th ConTEXt meeting 2020: 6.–12. September in Sibřina near Prague; pp. 118–124
The year is 1 after COVID-19. The TEX world is entirely occupied by corona viruses and thwarted. Well, not entirely… One small group of indomitable TEXies still met completely offline, on-site, face to face, but conforming to Corona regulations nonetheless. And since it was apparently the only TEX conference that went ahead as planned, it was, with only twelve participants, the biggest TEX conference worldwide! The meeting took place, as in 2018, at the former nicely renovated farm of the Škoda family, that also grows many herbs, keeps honey bees and offers all kinds of tasty products from honey, nuts and herbs. A wonderful quiet place that we had all to ourselves and where we could engross ourselves undisturbed in TEXnical and other thoughts.

Abstracts without papers; pp. 125–126

CG Secretary, 2020 Annual General Meeting Minutes; pp. 127–130

Participant list of the 14th ConTEXt meeting; p. 131

[Received from Taco Hoekwater.]

Fontzie
AAAA
AAAAA
AAAA
© John Atkinson, Wrong Hands
© John Atkinson, Wrong Hands • gocomics.com/wrong-hands • wronghands1.com

Comic by John Atkinson (`https://wronghands1.com`).

TEX Consultants

The information here comes from the consultants themselves. We do not include information we know to be false, but we cannot check out any of the information; we are transmitting it to you as it was given to us and do not promise it is correct. Also, this is not an official endorsement of the people listed here. We provide this list to enable you to contact service providers and decide for yourself whether to hire one.

TUG also provides an online list of consultants at `tug.org/consultants.html`. If you'd like to be listed, please see there.

Dangerous Curve

Email: `typesetting (at) dangerouscurve.org`
Typesetting for over 40 years, we have experience in production typography, graphic design, font design, and computer science, to name a few things. One of us co-authored, designed, and illustrated a TEX book (*TEX for the Impatient*).

We can: convert your documents to LATEX from just about anything, type your documents from handwritten pages, proofread, copyedit, and structure documents in English; apply publishers' specs; write custom packages and documentation; resize and edit your images for a better aesthetic effect; make your mathematics beautiful, produce commercial-quality tables with optimal column widths for headers and wrapped paragraphs; modify bibliography styles, make images using TEX-related graphic programs; design programmable fonts using METAFONT; and more! (Just ask.)

Our clients include high-end branding and advertising agencies, academics at top universities, leading publishers. A member of TUG, we also have supported the GNU Project for decades (and even have worked for them).

All quote work is complimentary.

Hendrickson, Amy

57 Longwood Avenue Apt. 8
Brookline, MA 02446
+1 617-738-8029
Email: `amyh (at) texnology.com`
Web: `http://www.texnology.com`
Full time LATEX consultant for more than 30 years — Our macro packages are used by thousands of authors. See our site for many samples: `texnology.com`.

Macro packages for books, journals, slides, posters, e-publishing and more.

Design as well as LATEX implementation for e-publishing or print books and journals, or specialized projects.

Data visualization, database publishing.

LATEX training via Zoom: Many years experience in on-site training, now offering scheduled Zoom classes! See `www.texnology.com/train.htm`.

I'll be glad to hear from you!

Dominici, Massimiliano

Email: `info (at) typotexnica.it`
Web: `http://www.typotexnica.it`
Our skills: layout of books, journals, articles; creation of LATEX classes and packages; graphic design; conversion between different formats of documents.

We offer our services (related to publishing in Mathematics, Physics and Humanities) for documents in Italian, English, or French. Let us know the work plan and details; we will find a customized solution. Please check our website and/or send us email for further details.

Latchman, David

2005 Eye St. Suite #6
Bakersfield, CA 93301
+1 518-951-8786
Email: `david.latchman (at) texnical-designs.com`
Web: `http://www.texnical-designs.com`
LATEX consultant specializing in the typesetting of books, manuscripts, articles, Word document conversions as well as creating the customized LATEX packages and classes to meet your needs. Contact us to discuss your project or visit the website for further details.

Monsurate, Rajiv

Web: `https://www.rajivmonsurate.com`
`https://latexwithstyle.com`
I offer: design of books and journals for print and online layouts with LATEX and CSS; production of books and journals for any layout with publish-ready PDF, HTML and XML from LATEX (bypassing any publishers' processes); custom development of LATEX packages with documentation; copyediting and proofreading for

English; training in LaTeX for authors, publishers and typesetters.

I have over two decades of experience in academic publishing, helping authors, publishers and typesetters use LaTeX. I've built typesetting and conversion systems with LaTeX and provided TeX support for a major publisher.

Sofka, Michael

Email: `michael.sofka (at) gmail.com`

Personalized, professional TeX and LaTeX consulting and programming services.

I offer 30 years of experience in programming, macro writing, and typesetting books, articles, newsletters, and theses in TeX and LaTeX: Automated document conversion; Programming in Perl, Python, C, R and other languages; Writing and customizing macro packages in TeX or LaTeX, `knitr`.

If you have a specialized TeX or LaTeX need, or if you are looking for the solution to your typographic problems, contact me. I will be happy to discuss your project.

Veytsman, Boris

132 Warbler Ln.
Brisbane, CA 94005
+1 703-915-2406
Email: `borisv (at) lk.net`
Web: `http://www.borisv.lk.net`

TeX and LaTeX consulting, training, typesetting and seminars. Integration with databases, automated document preparation, custom LaTeX packages, conversions (Word, OpenOffice etc.) and much more.

I have about two decades of experience in TeX and three decades of experience in teaching & training. I have authored more than forty packages on CTAN as well as Perl packages on CPAN and R packages on CRAN, published papers in TeX-related journals, and conducted several workshops on TeX and related subjects. Among my customers have been Google, US Treasury, FAO UN, Israel Journal of Mathematics, Annals of Mathematics, Res Philosophica, Philosophers' Imprint, No Starch Press, US Army Corps of Engineers, ACM, and many others.

We recently expanded our staff and operations to provide copy-editing, cleaning and troubleshooting of TeX manuscripts as well as typesetting of books, papers & journals, including multilingual copy with non-Latin scripts, and more.

Warde, Jake

90 Resaca Ave.
Box 452
Forest Knolls, CA 94933
+1 650-468-1393
Email: `jwarde (at) wardepub.com`
Web: `http://myprojectnotebook.com`

I have been in academic publishing for 30+ years. I was a Linguistics major at Stanford in the mid-1970s, then started a publishing career. I knew about TeX from editors at Addison-Wesley who were using it to publish beautifully set math and computer science books.

Long story short, I started using LaTeX for exploratory projects (see website above). I have completed typesetting projects for several journal articles. I have also explored the use of multiple languages in documents extensively. I have a strong developmental editing background in STEM subjects. If you need assistance getting your manuscript set in TeX I can help. And if I cannot help I'll let you know right away.

Hello from TeXnology!

Macro writing, Design, Data Visualization, E-Publishing, Innovations, and more

We've been writing macro files and teaching LaTeX for more than 30 years.

Now offering LaTeX classes via zoom
See www.texnology.com/train.htm for class dates, description of course and sample of hyperlinked course notes.

See you on-line!

– Amy Hendrickson
amyh@texnology.com

And, come visit our site for many and diverse examples of our LaTeX projects, with perhaps some ideas for projects you'd like to develop:

WWW.TEXNOLOGY.COM

Calendar

2021

Nov 13	TeXConf 2021 (Japan; online) texconf2021.tumblr.com
Dec 2 – 4	ATypI 2021 All Over (virtual). atypi.org/home/atypi-all-over-2021
Dec 14 – 17	SIGGRAPH Asia 2021, Tokyo, Japan. sa2021.siggraph.org/en

2022

Feb 12	Centre for Printing History & Culture, CPHC, Broadside Day, Birmingham, UK. www.cphc.org.uk/events
Mar 31	*TUGboat* **43**:1, submission deadline.
Apr 10 – 13	CODEX VIII, "Words on the Edge", Richmond, California. www.codexfoundation.org
Jun 8 – 10	Grapholinguistics in the 21st century — From graphemes to knowledge, Paliseau, France. grafematik2022.sciencesconf.org
Jun 20 – 22	International Society for the History and Theory of Intellectual Property (ISHTIP), 13th Annual Workshop, "Machines of Law and Intellectual Property as Legal Machinery", University of Gothenburg, Sweden. www.ishtip.org/?p=1210
Jun 20 – 22	Twentieth International Conference on New Directions in the Humanities, "Data, Media, Knowledge: Re-Considering Interdisciplinarity and the Digital Humanities", University of the Aegean, Rhodes, Greece. thehumanities.com/2022-conference
Jul 11 – 15	SHARP 2022, "Power of the written word", Society for the History of Authorship, Reading & Publishing. University of Amsterdam, The Netherlands www.sharpweb.org/main/conferences
Jul 25 – 29	Digital Humanities 2022, Alliance of Digital Humanities Organizations, "Responding to Asian Diversity", Tokyo, Japan, and Online. dh2022.adho.org

Owing to the COVID-19 pandemic, schedules may change. Check the websites for details.

Status as of 1 November 2021

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.

Introductory

224 *Barbara Beeton* / Editorial comments
- typography and *TUGboat* news

226 *Barbara Beeton* / Michael D. Spivak, 1940–2020
- memoriam for the author of AMSTEX, *The Joy of TEX*, and much more

228 *Barbara Beeton* / How to keep your sanity when preparing a transcript of an online interview for publication
- difficulties with interview transcriptions, and suggestions for easing them

311 *Hans Hagen* / A new unit for LMTX: The `dk`
- the potrzrebie unit (6.43985pt) now available in LMTX

223 *Boris Veytsman* / From the president
- on stability, licensing, and continuing maintenance of packages

Intermediate

312 *Karl Berry* / The treasure chest
- new CTAN packages, September–October 2021

231 *Charles Bigelow* / Form, pattern & texture in the typographic image
- the interaction of typographic size and combination leading to form, pattern, and texture

305 *LATEX Project Team* / LATEX news, issue 34, November 2021
- hooks; new or improved commands and code; changes in `amsmath`, `graphics`, `tools`

284 *Frank Mittelbach* / Preventing tofu with pdfTEX and Unicode engines
- input and font encodings, missing characters, Unicode, `\tracinglostchars`

287 *Frank Mittelbach* / The `unicodefonttable` package
- customizable font tables for OpenType/TrueType Unicode fonts

Intermediate Plus

310 *Karl Berry* / Production notes
- custom fonts, engines, page counts

258 *Hans Hagen* / Rendering open street maps
- generating graphics from `openstreetmap.com` data with MetaPost and ConTEXt

267 *Herbert Voß* / Controlling captions, fullpage and doublepage floats: `hvfloat`
- extensive examples of floating layouts, complex and simple

Advanced

247 *Amine Anane* / Arabic text justification using LuaLATEX and the DigitalKhatt OpenType variable font
- towards high-quality Arabic typesetting using Lua, OpenType, and METAFONT

Reports and notices

222 Institutional members

313 From other TEX journals:
La Lettre GUTenberg 41–44 (2020–2021);
Les Cahiers GUTenberg 58 (2021);
ConTEXt Group Journal: 14th meeting (2020)

316 *John Atkinson* / Comic: Fontzie

317 TEX consulting and production services

318 TEXnology Inc.

319 Pearson | Addison-Wesley

320 Calendar