

TUGBOAT

Volume 37, Number 3 / 2016

General Delivery	255	President's note / <i>Jim Hefferon</i>
	256	Editorial comments / <i>Barbara Beeton</i> R.I.P. Kris Rose, 1965–2016; A book fair... and another passing; Some typography links to follow; Another honor for Don Knuth; A fitting memorial for Sebastian Rahtz; Second annual Updike Prize for student type design; Talk by Fiona Ross
	259	Interview with Federico Garcia-De Castro / <i>David Walden</i>
Typography	264	Typographers' Inn / <i>Peter Flynn</i>
Software & Tools	267	LuaTeX version 1.0.0 / <i>Hans Hagen</i>
	269	LuaTeX 0.82 OpenType math enhancements / <i>Hans Hagen</i>
Electronic Documents	275	Introducing LaTeX Base / <i>Gareth Aye</i>
	277	Computer Modern Roman fonts for ebooks / <i>Martin Ruckert</i>
Graphics	281	When (image) size matters / <i>Peter Willadt</i>
Survey	284	A survey of the history of musical notation / <i>Werner Lemberg</i>
Fonts	305	Colorful emojis via Unicode and OpenType / <i>Hans Hagen</i>
	306	Cowfont (koeiletters) update / <i>Taco Hoekwater and Hans Hagen</i>
	311	Corrections for slanted stems in METAFONT and METAPOST / <i>Linus Romer</i>
	317	GUST e-foundry font projects / <i>Bogusław Jackowski, Piotr Strzelczyk, Piotr Pianowski</i>
IATEX	337	Localisation of TeX documents: <code>tracklang</code> / <i>Nicola Talbot</i>
	352	Glisternings: Index headers; Numerations; Real number comparison / <i>Peter Wilson</i>
Macros	357	Messing with endnotes / <i>David Walden</i>
	358	Tracing paragraphs / <i>Udo Wermuth</i>
Hints & Tricks	374	The treasure chest / <i>Karl Berry</i>
Cartoon	376	An asterisk's lament / <i>John Atkinson</i>
Abstracts	377	Die <i>TEXnische Komödie</i> : Contents of issues 2–3/2016
TUG Business	254	TUGboat editorial information
	254	TUG institutional members
	378	TUG 2015 election
Advertisements	379	TeX consulting and production services
News	380	Calendar

T_EX Users Group

TUGboat (ISSN 0896-3207) is published by the T_EX Users Group. Web: <http://tug.org/TUGboat>.

Individual memberships

2017 dues for individual members are as follows:

- Regular members: \$105.
- Special rate: \$75.

The special rate is available to students, seniors, and citizens of countries with modest economies, as detailed on our web site. Also, anyone joining or renewing before March 31 receives a \$20 discount:

- Regular members (early bird): \$85.
- Special rate (early bird): \$55.

Members also have the option to receive *TUGboat* and other benefits electronically, for an additional discount.

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

Journal subscriptions

TUGboat subscriptions (non-voting) are available to organizations and others wishing to receive *TUGboat* in a name other than that of an individual. The subscription rate for 2017 is \$110.

Institutional memberships

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

Trademarks

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

T_EX is a trademark of American Mathematical Society.

METAFONT is a trademark of Addison-Wesley Inc.

PostScript is a trademark of Adobe Systems, Inc.

[printing date: November 2016]

Printed in U.S.A.

Board of Directors

Donald Knuth, *Grand Wizard of T_EX-arcana*[†]

Jim Hefferon, *President*^{*}

Boris Veytsman^{*}, *Vice President*

Klaus Höppner^{*}, *Treasurer*

Susan DeMeritt^{*}, *Secretary*

Barbara Beeton

Karl Berry

Kaja Christiansen

Michael Doob

Steve Grathwohl

Steve Peter

Cheryl Ponchin

Geoffrey Poore

Norbert Preining

Arthur Reutenauer

Michael Sofka

Raymond Goucher, *Founding Executive Director*[†]

Hermann Zapf (1918–2015), *Wizard of Fonts*

*member of executive committee

†honorary

See <http://tug.org/board.html> for a roster of all past and present board members, and other official positions.

Addresses

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

Electronic Mail

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

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

Telephone

+1 503 223-9994

Fax

+1 815 301-3568

Web

<http://tug.org/>
<http://tug.org/TUGboat/>

Technical support for
T_EX users:
support@tug.org

Contact the
Board of Directors:
board@tug.org

Copyright © 2016 T_EX Users Group.

Copyright to individual articles within this publication remains with their authors, so the articles may not be reproduced, distributed or translated without the authors' permission.

For the editorial and other material not ascribed to a particular author, permission is granted to make and distribute verbatim copies without royalty, in any medium, provided the copyright notice and this permission notice are preserved.

Permission is also granted to make, copy and distribute translations of such editorial material into another language, except that the T_EX Users Group must approve translations of this permission notice itself. Lacking such approval, the original English permission notice must be included.

“Editors are ghouls and cannibals.”
Harriet Vane to Salcombe Hardy

Dorothy L. Sayers
Busman’s Honeymoon (A love story
with detective interruptions) (1937)

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP
EDITOR BARBARA BEETON

VOLUME 37, NUMBER 3 • 2016
PORTLAND • OREGON • U.S.A.

TUGboat editorial information

This regular issue (Vol. 37, No. 3) is the last issue of the 2016 volume year.

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

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

Submitting items for publication

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

The first issue for 2017 will be a regular issue, with a deadline of February 24, 2017. The second 2017 issue will be the proceedings of the TUG'17 conference (tug.org/tug2017). The third issue deadline is September 1.

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

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

TUG Institutional Members

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

American Mathematical Society,
Providence, Rhode Island

Aware Software, Inc.,
Midland Park, New Jersey

Center for Computing Sciences,
Bowie, Maryland

CSTUG, *Praha, Czech Republic*

Fermilab, *Batavia, Illinois*

Institute for Defense Analyses,
 Center for Communications

Research, *Princeton, New Jersey*

Maluhy & Co., *São Paulo, Brazil*

Marquette University,
Milwaukee, Wisconsin

Masaryk University,
 Faculty of Informatics,
Brno, Czech Republic

MOSEK ApS,
Copenhagen, Denmark

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

Overleaf, *London, UK*

River Valley Technologies,
Trivandrum, India

ShareLaTeX, *United Kingdom*

Springer-Verlag Heidelberg,
Heidelberg, Germany

StackExchange,
New York City, New York

Stanford University,
 Computer Science Department,
Stanford, California

Stockholm University,
 Department of Mathematics,
Stockholm, Sweden

TNQ, *Chennai, India*

University College, Cork,
 Computer Centre,
Cork, Ireland

Université Laval,
Ste-Foy, Québec, Canada

University of Cambridge,
 Centre for Mathematical Sciences,
Cambridge, United Kingdom

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

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

TUGboat editorial board

Barbara Beeton, *Editor-in-Chief*

Karl Berry, *Production Manager*

Boris Veytsman, *Associate Editor, Book Reviews*

Production team

William Adams, Barbara Beeton, Karl Berry,
 Kaja Christiansen, Robin Fairbairns, Robin Laakso,
 Steve Peter, Michael Sofka, Christina Thiele

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, send the information to the attention of the Publications Committee at tug-pub@tug.org.

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

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. The following list of trademarks which commonly appear in *TUGboat* should not be considered complete.

METAFONT is a trademark of Addison-Wesley Inc.

PostScript is a trademark of Adobe Systems, Inc.

TeX and *AMS*-TeX are trademarks of the American Mathematical Society.

President's note

Jim Hefferon

Lots of things are happening in the \TeX world, some new and some older, some great to hear and some otherwise.

Elections

This is an election year for TUG. There are ten seats up for a vote this year, including mine as President.

We invite nominations for these openings. I hope that you will consider serving if you are able. It is a truism but it is nonetheless true that the work doesn't get done unless somebody does it.

The page tug.org/election/ has the full information. If you have any questions, please contact the election committee, whose address is on that page. The submission deadline, which will be strictly kept, is February 1.

Conferences

Next year's TUG conference will take place in conjunction with the yearly Bacho \TeX conference. We are excited to be joining the Polish group GUST to celebrate their 25th birthday. A big thank you to them for agreeing to host!

The dates are April 29–May 3, 2017 (note that this differs from the midsummer time we have often used in the past). The monthly online TUG newsletter and the web page tug.org/tug2017/ will have more information as it becomes available.

I've also just heard the dates and location for the 11th International Con $\text{\TeX}t$ meeting have been set: September 11–17, 2017 in Butzbach-Maibach, Germany. More information on this will be at meeting.contextgarden.net/2017/.

New projects

We have activated two working groups.

The PDF accessibility group addresses the very important work of implementing accessibility standards in \TeX . It has a web page tug.org/twg/accessibility/ and mailing list lists.tug.org/accessibility. You can direct a portion of your TUG donations, which are tax deductible in the US, to this group.

The education working group is aimed at helping to teach and promote \TeX in schools and universities. The mailing list is lists.tug.org/edutex and the freshly minted web site at tug.org/twg/edutex/.

This second group is near to my heart. I would dearly like to see more undergraduates using \TeX and friends. I'm a mathematics professor and I teach an introduction to proofs class. As part of

the class I require students to learn enough \LaTeX to do their homework. It is a professional standard in mathematics and I think it is to their benefit to know it.

If you are interested in these issues, please join!

Membership drive continues

We are continuing our campaign *Members Bring Members*. We ask all TUG members to help spread the word about our community by inviting new individuals and organizations to join.

If you know someone who uses \TeX and friends, someone who is interested in high quality electronic documents, then please urge them to consider joining TUG.

On the TUG signup form at tug.org/forms/current/memberapp.html, the first question is for new members: "Who invited you to join TUG?" We acknowledge each such sponsor with a small gift, a postcard made by long-time TUG member Peter Wilson on his letterpress specially for this campaign (or, if preferred, any physical item from TUG store). Moreover, we will recognize participants in the campaign at the TUG meeting, in *TUGboat*, and on the web site. At the end of 2016, we will also hold a drawing and the person selected will receive a copy of the limited edition book *Manuale Zapficum: Typographic arrangements of the words by and about the work of Hermann Zapf & Gudrun Zapf von Hesse* (2009).

Helping

We all benefit from the work of others, and that includes those of us in the \TeX world. If you are able to give back a bit but are not sure how, you could consider running for the TUG Board; see above. Alternatively, the technical committee maintain a list of project ideas for anyone interested to tackle, at tug.org/help.html. Have a look; they are all worthwhile.

Kristoffer Rose

We learned with sadness that Kris Rose has passed away in September. He is one of those people whose work has helped me personally. He was the author of the widely used Xy-pic package. And, he was a member of the TUG Board from 1997–2003, as well as Vice President from 1997–2001. He was also a contributor to Debian. Barbara will say more elsewhere in this issue but we were sorry to hear the news and our thoughts go out to his loved ones.

◊ Jim Hefferon
Saint Michael's College
[jhefferon \(at\) smcvt dot edu](mailto:jhefferon(at)smcvt.edu)

Editorial comments

Barbara Beeton

R.I.P. Kris Rose, 1965–2016

Kristoffer Høgsbro Rose was a native of Denmark, born 5 April 1965. He discovered computer science when he was very young, and spent the rest of his life involved in this pursuit. He was an early contributor to AUC-*TeX*, and the author of *Xy-pic*.

In Aarhus in the mid-1990s, after receiving his degree from the University of Copenhagen, he was teaching at and working with the Basic Research in Computer Science (BRICS) center; he would from time to time visit with Kaja Christiansen and have a chat. Kaja reports:

We'd talk *TeX*, *Xy-pic*, Debian or Emacs, or he would sit down and read my copies of *TUGboat*. In 1997 we happened to talk about TUG; the same year I decided to join the board at TUG'97.

Kris joined the TUG board at the same time, and was elected Vice President for a term through 2001; he remained on the board until 2003. In addition to his TUG participation, he was active in the Debian open source and free software community.

Also in 1997, Kris moved with his family from Denmark to France, taking a teaching position at the École Normale Supérieur (ENS) de Lyon, from which position he was invited in 2000 by IBM to join the TJ Watson facility in New York as a Research Scientist. It was at this point that he left the *TeX* community, but continued to be active in Debian. In 2013, while still at IBM, he joined the adjunct faculty of New York University, where he taught compiler construction. In 2014, he left IBM to become a research scientist in the financial industry, at Two Sigma Investments, while continuing to teach at NYU, and becoming more active as a Debian contributor. Late in 2015 he was diagnosed with a very aggressive form of leukemia, which took his life on 17 September 2016. He was far too young.

A book fair...and another passing

The first weekend in October, my husband and I attended the Oak Knoll Fest XIX in New Castle, Delaware, hosted by Oak Knoll Press.¹ (Some attendees at the 2001 TUG meeting, held at the University of Delaware, may remember my recommendation to visit the bookshop.) This book fair is held every other year, and I look forward to it eagerly. So it surprised and saddened me to learn of the death, just

a week earlier, of Oak Knoll's proprietor and guiding spirit of the Fest, Robert Fleck; nonetheless, the Fest went on as planned, following Bob's admonition to his son Rob, "Hell no! We've already paid for it!" (Rob and his mother, Millie, intend to continue the work Bob started, Bob's plans are solidly in place for the next several years.)

Oak Knoll is both a bookshop and a publisher, with a very specialized focus—books about books. I first became familiar with Oak Knoll in the early years of *TeX* when I was looking for some of the books listed in the bibliography of Don Knuth's Gibbs lecture, "Mathematical Typography".² In addition to the (very few) publications devoted to math composition, the shop is full of publications about fonts, composition and printing, bibliography, book-binding, papermaking, fine press books, A most valuable resource for information on the history of type and printing.

Bob Fleck also recognized an interest in contemporary hand-set and artists' books, and in 1996 encouraged the founding of the Fine Press Book Association (FPBA).³ The biennial Oak Knoll Fest comprises a symposium on book-related topics as well as a book fair where book-makers, most of them FPBA members, exhibit their creations and works in progress. The array of books and ephemera to be seen is dizzying in its variety.

The topic for this year's symposium was the question: what are the most important criteria for a private press when selecting texts to print? While this matter is undoubtedly secondary for most *TeX* users, the opinions and experience of the participants were interesting and enlightening for any active or prospective book collector.

For anyone who loves books and is in the Delaware vicinity around the beginning of October (in even-numbered years; it alternates years with the Oxford Book Fair, in the UK), attending the Fest is a recommended activity.

Another honor for Don Knuth: the SIAM John von Neumann Lecture

On 12 July 2016, the John von Neumann Lecture prize was awarded to Don Knuth "for his transformative contributions to mathematics and computer science". Knuth delivered the associated prize lecture, "Satisfiability and Combinatorics" on that day to the

² *Bulletin of the American Mathematical Society (new series)*, 1:2, 337–372 (March 1979), <https://www.ams.org/bull/1979-01-02/S0273-0979-1979-14598-1>; republished in *Digital Typography*, pp. 19–65.

³ <http://fpba.com>

¹ <http://www.oakknoll.com>

Some typography links to follow

On the illegibility of street signs in New York, in verse:
<http://flip.it/HLIMY>

How typeface designers made room in the *New York Times* for President Eisenhower's long last name:
<http://www.theatlantic.com/technology/archive/2016/06/eisenhower-and-the-skinny-s/486965/>

Selections from the blog of St Brigid Press, in the Blue Ridge Mountains of Virginia:

How type is made, in two parts:
<http://www.stbrigidpress.net/blog/how-type-is-made-part-1>
<http://www.stbrigidpress.net/blog/how-type-is-made-part-2>

A letterpress lexicon, in (so far) three parts:
<http://www.stbrigidpress.net/blog/a-letterpress-lexicon-part-1>
<http://www.stbrigidpress.net/blog/a-letterpress-lexicon-part-2>
<http://www.stbrigidpress.net/blog/a-letterpress-lexicon-part-three>

The blog itself:
<http://www.stbrigidpress.net/blog>

Videos from Type@Cooper—Lectures presented in conjunction with the Cooper Union typeface design program, in New York and San Francisco:
<https://vimeo.com/coopertype/videos>

Donald Knuth, “32 Years of Metafont” (Type@CooperWest talk):
https://www.youtube.com/watch?v=0LR_1BEy7qU

Announcements of upcoming lectures:
<http://coopertype.org/>

annual meeting of the Society for Industrial and Applied Mathematics (SIAM) in Boston, Massachusetts. This is the highest honor awarded by SIAM; “the flagship lecture recognizes outstanding and distinguished contributions to the field of applied mathematical sciences and the effective communication of these ideas to the community.”

A fitting memorial for Sebastian Rahtz

The Text Encoding Initiative (TEI) has announced the creation of the Rahtz Prize for TEI Ingenuity. The prize is described in part as follows:

The TEI Consortium has created the Rahtz Prize for TEI Ingenuity in memory of Sebastian Rahtz (13 February 1955–15 March 2016). The award is intended to honour Sebastian’s major technical and philosophical contributions to the TEI, and to encourage TEI innovation by the TEI community.

The full announcement can be read at <http://www.tei-c.org/Activities/rahtz.xml>; nominations for the first award are due 1 April 2017.

Second annual Updike Prize for student type design

On October 17, safely outside of the winter storm season,⁴ the award ceremony for the second annual Updike Prize for student type design was held at the Providence Public Library. The invited speaker was Dr. Fiona Ross of the University of Reading.

Four finalists were announced, and their entries were on exhibit, along with information about the sources they had consulted for inspiration. Here are their names, and the names of their typefaces.

- June Shin, *Ithaka* (First Prize)
- SooHee Cho, *The Black Cat*
- Cem Eskinazi, *Mond*
- Íñigo López Vázquez, *Erik Text*

A brief announcement is at <https://pplspcoll.wordpress.com/2016/10/20/congratulations-to-june-shin-winner-of-the-2016-updike-prize/>, and includes several related links.

⁴ Last year’s presentation, on 19 February 2015, was accompanied by a fierce snowstorm. The event was reported in my column in *TUGboat* 36:1, <http://tug.org/TUGboat/tb36-1/tb112beet.pdf>.

Talk by Fiona Ross

Fiona Ross is on the faculty of the University of Reading, where she lectures on non-Latin typeface design in the MA Typeface Design program, and is curator of the Non-Latin Type Collection. (She is also an Associate Designer for Tiro Typeworks, the organization which is polishing version 2 of the STIX fonts.) Her talk, on the occasion of the Updike Prize ceremony, entitled “Collections-based research for contemporary typeface design—with special reference to non-Latin scripts”, dealt with the resources necessary when designing fonts for languages in which one is not a native speaker, and how to make most effective use of them.

Dr. Ross used the Bengali script as her main example. Bengali has a long history, longer even than Latin, with the oldest representations being carved in stone, and more recent, though still old, examples produced with a broad-edge pen that has the writing edge slanted in the opposite direction from that of the broad-edge pen used for italic script. Although the Bengali script is strongly alphabetic, the glyphs are based on consonant clusters, with vowels relegated almost to diacritic status. The order of written phonemes is not necessarily the same as how the phonemes occur in the spoken word. Wide elements at the top or (less frequently) bottom traditionally overlap what occurs next to them; the overlap can occur on either side. The setting of these features in type is strongly influenced by what is possible with the available technology.

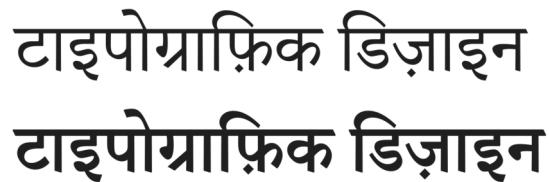
Dr. Ross’ studies in Sanskrit prepared her for her first assignment at Linotype (UK), where she undertook to redesign the Bengali font for use with a filmsetter. The existing Linotype Bengali font was designed for use on a hot-metal typesetter, which had no real ability to kern adjacent characters. Without this ability, the only alternative would be to provide ligatures, which for Bengali would increase the number of glyphs to several hundred; with a physical capacity of only 90 characters at a time, the Linotype was incapable of accommodating this requirement. For this reason, many shapes were restricted to a width narrower than tradition would dictate. But the desire for printed material (India is still devoted to reading the daily newspaper) was stronger than the requirement for typography that embodied traditional elegance.

The design of a new font, even for a new technology, should not be simply a clone of an existing font, even if it is meant to fill the same niche. Especially if a new technology provides possibilities that were not available under previous technologies, the opportunity should be taken to create something that

matches the expectations of the culture whose language it will be used to exemplify. So it was possible, with the enhanced capabilities of the filmsetter, to ignore the limitations that had heretofore restricted the font design.

The UK Linotype company held a collection of manuscripts and printed materials in the relevant script, as well as having a branch in India with personnel willing to supply not only more examples, but also the expertise of native language speakers. Together, these resources fulfilled the three criteria that are required for development of a new font (besides the efforts of a skilled designer): relevance, significance, and reliability. For Bengali, the available materials covered a broad period as well as a significant variety of likely applications. The staff of the office in India were enthusiastic about the project, allowing work on the new font to be a true team venture. Despite Dr. Ross’ lack of native competency in Bengali, the ability to ask the right questions and attention to the opinions of those native speakers resulted in a product that was readily adopted by the major Bengali newspapers, and even today, more than thirty years later, it is still the predominant font used by the newspapers.

The image below was kindly provided by Dr. Ross to illustrate this report. It says, in Hindi transliteration, “Typographic Design” in Adobe Devanagari Regular and Bold, designed by Tim Holloway, Fiona Ross, and John Hudson.



Two slides showing the Hindi transliteration of “Typographic Design” in Adobe Devanagari Regular and Bold. The top slide shows the word in Regular weight, and the bottom slide shows it in Bold weight.

Every slide illustrating Dr. Ross’ talk included the Bengali letter “ka” (Unicode U+0995) as an icon. An inquiry elicited the information that “ka” is auspicious; this is the first glyph that she designs in every script.

On 17 July 2016, Dr. Ross presented a lecture on a related topic at Typographics 2016, held at The Cooper Union in New York City. A video of that talk can be viewed at https://www.youtube.com/watch?v=3_MbN_pBuy0. In her slides, starting at 3:29, the iconic Bengali “ka” can be seen, usually in the lower left-hand corner.

Interview: Federico Garcia-De Castro

David Walden



Erin Gallagher Pesa

Federico Garcia-De Castro is a composer of music, passionate for chess, and a lover of TeX.

Dave Walden, interviewer: Please tell me a bit about yourself.

Federico Garcia-De Castro, interviewee: I was born and raised in Bogota, Colombia. I was a pretty normal child, but I pursued a couple of interests from an early age—I went to the conservatory for a couple years, before that conflicted with math Olympiad training and I decided to do math. I'd return a couple of years later, as I remember out of curiosity and a vague remembrance that it (music) had actually been interesting. Math still figured at the top, though, but then eventually I left math to devote myself to chess, which was my life pretty much between 11 and 16. And, studying chess, I started listening to music (while studying openings, tactics, etc.), and that's when I got hooked on music. In the end, at 16 I decided music—composition—was my thing.

So my mom, who had been the one pushing for my early music training (and hated that I quit for math, and later hated much worse that I quit math for chess!), ended up winning that one. The one who lost, however, was my uncle Rodrigo De Castro. A mathematician, uncle Luli was in Chicago for seven years before returning to Colombia in 1993, when I was 15. He's also a huge music aficionado and connoisseur, and was the one who triggered my teenage interest in music . . . always trusting that I clearly was going to be a mathematician. He's most relevant here, however, because in 1993 he brought TeX to Colombia.

Now, a digression for context: some four years ago I was visiting Bogota, and went through some notebooks and papers and things my mom had kept



Federico conducting a composition of his at the 2015 MusicArte Festival in Panama

from our childhood years. (That is, mine and my brother's; Nicolás is two years younger, and now a German philologist; he writes amazing poetry on the side, and I have set a couple of his poems to chamber works.) In those childhood archives I found a small slip of paper, which had, written clearly in a child's handwriting, some 10 lines of a program—in BASIC. This discovery, and some things it made us all remember, helped me, as an adult, understand a lot of my life in a new light. Whatever I did as a child, whatever I pursued, whatever courses I enrolled, I was always, and have always been, foremost a programmer. In the wide sense: what else is a music score? No more and no less than a program—a series of instructions, written in a code (with its own syntax that you have to learn, etc.), that will be executed. And then debugged. A musical work is not exactly analogous to an algorithm, but even so, what a composer does is programming.

My relationship with composition has always been tense (as probably anyone's is—nothing special there). When I came to the United States, for PhD in composition at Pitt in 2001, I also “discovered” musicology, and I did much more musicology than composition. From that time stems the motivation for my first L^AT_EX packages.

DW: Let's talk about them later.

FGDC: Then eventually between 2005 and 2009 I was *really* devoting myself to chess—in 2009 I won, amazingly, the Pittsburgh Chess Club championship and the informal “state” tournament at Carlisle, PA. That's where the TeXmate package stemmed from. Soon after my graduation in 2006 I co-founded a contemporary music organization, which has since grown stronger and stronger. That meant for years I flirted seriously with a career in conducting . . .

It's like I always looked for something to do, something to be, *other than* a composer.... That's why it was so important to me to understand, as mentioned above, that I am mainly a programmer, and that my interest in composing is of the same nature. Composing, I've since also figured out, is just much harder and much scarier (among other things, you don't have the luxury of constant feedback through test runs). But it is what I am, period.

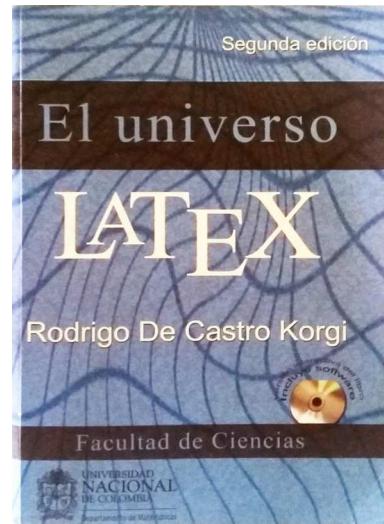
DW: How did you first come in contact with \TeX ?

FGDC: So in 1993 I ran into \TeX . Then I only had an Atari 130 XE, which I got after months and months of pleading to my parents because I wanted finally to be able to program on a computer, not on slips of paper ... (also, my parents couldn't readily afford a computer right away, that was the reason for the wait). I had no PC to run \TeX on in 1993, but I have the vivid memory of being in a class in school and writing out the \TeX document that would produce whatever the textbook we were looking at was. I'd write it on the left side of the notebook; on the right, I'd calligraph out what would have been \TeX 's output. This was in 1993, just when I got to know \TeX and was hooked enough that I had to do it even on paper :) That summer a colleague of my uncle hired me to typeset his math book. I remember too the frustrating moments with ! `Extra` messages, and my uncle coming to my rescue (I worked at my uncle's place—I did not have a PC!) when the tears were starting to come.

Back then, AMS- \TeX was the thing. (In fact, my uncle had brought with him the \LaTeX manual, but he had never used \LaTeX and only vaguely told me what it was. I borrowed the book. I was the one who told my uncle (!) that \LaTeX was onto something....) The point is that, for me, names like Spivak, Mittelbach, Rahtz, etc., were the same as names like Mozart, Beethoven, Brahms.

This last one with an h; Braams would also become like that when my uncle kept writing \LaTeX manuals for Colombian (and Spanish-speaking) \TeX users, with which I helped him. In 2002, I think, *El Universo \LaTeX* featured a CD with examples and summaries—in PDF, done by me, with color syntax highlighting and plenty of hyperlinks—a lot of fun with \verb+. Years and years later, when working seriously on my most complicated program (\TeXmuse), I looked back at this code, when I wanted to add syntax highlighting to the \LaTeX docstrip system—what eventually became the `colordoc` package.

In 2011, I think, Frank Mittelbach (Beethoven!) got in touch with me to include \TeXmate in the second volume of the *\LaTeX Graphics Companion*.



One of the \LaTeX books written by Federico's uncle in 2002; Federico prepared the PDF summary included in the CD.

And then I met him, in Boston in 2012. A couple of weeks ago (as of this writing) I had multiple beers and chats with him in Toronto.

DW: When you say that your uncle brought \TeX to Colombia, do you mean to you personally or basically to all Colombia?

FGDC: To all Colombia. He basically brought \TeX with him when returning from his time in the US (PhD in math at Champaign Urbana). At around the same time \TeX was also brought into the community centered on another person (I can't remember his name), who was associated with the Math Olympiads. But my uncle brought it to the National University, much more central in the math community. And then he immediately started writing booklets, manuals, etc., that people needed in order to learn. I remember he even did one on \PiCTeX . And a full booklet for tables, which of course were a problematic subject in \TeX . And so on.

DW: Is your uncle still alive and promoting \TeX ?

FGDC: He's still active at the math department. Demand for \TeX manuals declined once most people know how to use it and with the availability of everything online these days. But I'm sure copies of that book are still sold every once in a while today.

DW: You mentioned \TeXmate . What other things have you implemented in \TeX . (I know you have written about some of them.)

FGDC: Here's a list:

- `subfiles`—an alternative to the `\include` mechanism that allows the subsidiary

documents to be typeset as stand-alone documents.

- **todo**—utilities to add a to-do list at the end of a document.
- **texmate**—comprehensive chess annotation.
- **opcit**—support for footnote-style bibliographies in L^AT_EX.
- **colordoc**—modification of the doc package that provides syntax highlighting.

And there's **TeXmuse**, of course

DW: Yes, I have read your papers on **TeXmuse**: On musical typesetting: Sonata for **TeX** and **META-FONT**, Op. 2¹, and **TeX** and music: An update on **TeXmuse**².

I'd like to come back to **TeXmuse**. How about **TEXcel**, which you presented at TUG 2016³?

FGDC: **TEXcel** is a set of macros I developed this spring for financial tracking and reporting for my contemporary music company. It was a fun project, a nice excuse to program (go back to hacking **TeX**), with a clear and manageable goal. It is a complete and robust system, but it simply does what it does: helps in reporting financial information for my organization. It is certainly the backbone of what conceivably would be a more "public" and general purpose package; and, inasmuch as it was a kind of discovery (in the sense that who would have thought that **TeX** was more appropriate for this than a spreadsheet), it was an interesting presentation for the TUG meeting. But I would not count it next to the packages above that are meant for general use by general users without much further hacking.

DW: Let's return to your composing. Is it possible to label the kind of music you write, and who are your musical influences (in addition to your uncle)?

FGDC: The genre is usually called "new music". But for someone who doesn't already know what that means, it doesn't mean much. Lately I've said "contemporary chamber music". That gets it across.

I think Mahler and Beethoven are the greatest composers in history. At one point I was a devoted Bachian (I even published a scholarly article on his Italian Concerto), but I have since matured, hehe.

All that said, it is when I listen to Luciano Berio, Witold Lutosławski, and George Crumb that I remember why I am a composer.

DW: Might you give us an example of each of these composers that is in any sense representative of their compositions? Perhaps I can listen to them via YouTube.

FGDC: Crumb's Makrokosmos series (the first one is for solo piano, then he wrote one for amplified

piano, then the masterpiece volume III for two pianos and two percussionists, and finally one for amplified piano four hands) is in my view among the highest creations of the human mind in any area. Crumb's music is unmistakable, but in a very open way: it's prolific in his influence on other composers (in the 80s, for example, it swept through Latin America and everyone was writing like him for a while, in the good sense), unlike, for example, Messiaen's, who sounds like Messiaen to the point that it came to be a kind of dead end with no more to be explored there.

Berio's Laborintus II is a piece that also fills me with admiration. This coming spring (I can barely believe it), I am producing and conducting the Pittsburgh premiere of the piece (which is very ambitious: 17 instruments, 8 actors/speakers, 3 solo sopranos, narrator, and electronics) as part of a residency with Chicago-based vocal ensemble Quince. This has been more of a dream than a goal for me; until it became a possibility last year, it was simply too unthinkable.

Lutosławski's music is certainly my biggest influence. I think the piano concerto is second only, maybe, to Beethoven's Emperor. His cello concerto, his third symphony, and his string quartet are towering exemplars of each genre. His music is relatively conservative in format (most of it for orchestra, which is a 19th-century instrument), if not in its amazing content; and this has meant that he's a little less "sexy" than more fashionable contemporaneous names, notably Ligeti. But everyone agrees Lutosławski is a giant.

DW: On your website⁴, I see full scores⁵ for some of your compositions. How do you do your composing—musical instrument, music paper and pencil, instrument and music notation program, or directly into a music notation program (e.g., **TeXmuse**)?

FGDC: It really depends on the piece, the project, the kind of idea I'm struck by. I just finished a guitar piece, that's inspired by a Cuban trova song (Pedro Luis Ferrer's Romanza de la niña mala). You can think of it almost as a translation into a new language (in my piece, the guitar is tuned in microtones; and in addition it has no lyrics, it's just the guitar part). The process was: passage by passage, listen to the original, then find the right "translation" on my guitar; I'd then lock the left hand in position, so as to not lose the chord, and quickly scribble it down with my right hand on my notebook. But not all the details—those would be added simply when (next step) I'd type the notes into the computer. On the other hand, I've been composing a piece for six harps placed around the audience; in this case I have no



Federico at the piano strings for his composition *Livre Pour Deux Pianos*, August 2014, New Hazlett Theater Community Supported Art Performance Series, Pittsburgh, PA (photo by Renee Rosensteel)

instrument to find chords on and the piece is really about the surround effect, so exactly what notes I write is really a minor point. For this there's mainly a huge "research" stage that's really about discovering what the piece is about, and I am a long way away from even knowing what the best notation will be for it—so the computer is useless at this point.

The question, I think, touches on the relationship between the notebook and the computer. Do musicians have it all in their heads? Do they figure it out on paper? Do they use the computer for it? All understandable questions, but in a way they are illusory: would you ask a novelist, or a scholar, whether they write first on paper and then type it up? The answer in that case is not only obvious (take notes as needed on paper, but go ahead and use the computer for the actual text), but also not very interesting. Well, I hate to say it, but writing music doesn't carry any more mystery about it than writing words!

DW: Please compare your *TeXmuse* music transcription with other programs, for example, Finale, Sibelius, MusiXTEX, or Lilypond.

FGDC: The most important aspect and uniqueness in *TeXmuse*'s approach is that it is all about keeping the act of typesetting music as close as writing it by hand as possible. In several respects: if you're, say, in c-minor, you don't write the flat next to each e, even though you mean it (because that's part of what being in c minor means). This is the major deficiency of Lilypond—a very complete system, but with a syntax and an approach so autonomous from the musician's mind that it's a chore more than a help.

TeXmuse also features some algorithms that

were pursued in the belief that mechanical tasks are exactly what computers are for. In music, in particular, pitch spelling (including transposition) and line breaking are areas that are largely mechanical but that have been left alone by music typesetting software. Among the two of them they consume the biggest (and unacceptably big) portion of a composer's time these days.... It was this frustration that led me to think of *TeX* for music typesetting. And it was so high on my priority list that I tackled those two algorithms very early on: if they were not going to be possible, then the rest would not be worth it.

In recent times there have been advances. I follow from a distance, but from what I have seen MusicXML is very good, there are serious people doing serious research on all of this. (I'm still not aware of anyone tackling the spelling problem, though.) It's part of the reason why I haven't continued working on *TeXmuse* (in addition to time, my "real" responsibilities, etc.): that space is in a sense crowded now, and crowded with good efforts. As a system, and thinking of the user's experience, *TeX* (sadly) is not really the most promising environment; I don't see the whole musician community installing and deciphering *TeX*.

TeXmuse can still be a contribution at some point: as a front end to the other systems that have been developing. At this point in *TeXmuse*, *TeX* takes the user's input (and this is its strength) and from it, it makes METAFONT programs that produce the music. There's no reason why *TeXmuse*'s input couldn't be something other than METAFONT (for example, Lilypond, or MusicXML). In fact, this is probably much much easier than programming METAFONT automatically!

DW: Please say something about Alia Musica⁶ and the new music festival⁷ it sponsors, e.g., how and why were they founded, how does it help you (or the world) for you to spend time as artistic director and obviously doing lots of less artistic work to keep them functioning?

FGDC: In the last 15 years or so (I lived here since 2001), Pittsburgh has been going through a major cultural renaissance. Even when I got here the city was far from its infamous past as a dirty steel mid-west town, and had already shifted to things like the health system (the biggest industry in Pittsburgh, through the University of Pittsburgh Medical Center), high-tech research (mainly at Carnegie Mellon University) and eventually high-tech industry. But culturally it's been a thing of the last decade or so. I founded Alia Musica with another 10 emerging

composers in 2006–7. The main reason was that there were really no professional outlets for the work of young composers in the city, and in fact there was little visibility, little funding, and little interest. In a word, we realized that since no one was playing our music we would have to do it ourselves. A lot has changed, and it's continuing to change. Some things lag behind (funding, media attention) but they're catching up. In any case, the contemporary chamber music scene is much larger, and not only in quantitative terms. More and more young people (and that includes, simply statistically, artists, and within them musicians) are either moving in or (perhaps more relevantly in a city with a long tradition of excellent music performance schools) staying after advanced studies.

I was reflecting on this during an interview I did for the new-music blog “I Care If You Listen” on the occasion of the 2016 edition of the Pittsburgh Festival of New Music⁸ (PFNM), a production of mine and of Alia Musica’s in May 2016. As I was saying, when we started there wasn’t much devoted to cultivating the creation and appreciation of new music in the city (and that’s why we started; and, by the way, we were not the only ones that started more or less at that same time). By contrast, at PPNM 2016 one of the performances was a showcase of current new-music activity in Pittsburgh, for which I had to select groups. I had “room” for seven, and I had to make a choice, and it wasn’t easy. New initiatives pop up, some die, some endure, but the general sense is of vibrant activity. Alia Musica itself has in a way grown out of its original mission of performing Pittsburgh composers; other younger initiatives have taken that role. Alia Musica has been able now to shift its focus, from making an impact on the careers of emerging musicians, to making an impact on the actual life experience of its audiences. So, for example:

- In 2014 we produced a piece that was written for 9 to 99 percussionists by Pulitzer Prize winner John Luther Adams; we gathered 67 percussionists at a park.
- We presented one of the most epic works of the 20th century, the variations for piano on The People United will Never Be Defeated, performed by the composer himself, the legendary Frederic Rzewski, at a fish market!
- In 2015 we were able to bring a residency with cutting-edge California composer/vocalist Ken Ueno⁹ to perform his own concerto for overtone/throat-singing and orchestra (excerpt¹⁰).
- As part of the May 2016 festival we produced a flashmob of Stravinsky’s Firebird¹¹ at the central Market Square in Pittsburgh.

These are epic, unforgettable events—music at its most relevant, the proof, in fact, that music can still be relevant. I mean, who cares that the counterpoint is well crafted or that the pianist has good technique? When you’re seeing (as in the third bullet point above) a symphony orchestra re-creating and elaborating on the sound of a low throat-singing note, with muted trombones and string glissandos, who cares really whether the tempo is correct? Who cares, even, who the composer is? More and more in the mind of the general audience, Alia Musica is the folks who bring these experiences to our lives.

I should mention one more thing, namely, one of the young ensembles just founded, NAT 28¹². I have been in touch with them in regards to my 10 years of professional activity as a composer in Pittsburgh. As part of the celebration, NAT 28 is going to devote a full concert to my music this November. A “portrait concert”, the dream of any composer. They are bright young musicians, recent graduates in performance, and beyond (or before) plans for this concert I have had nothing to do with their formation. I take this as a sign that the seeds planted by Alia Musica (among many seeds planted by others as well) are in fact taking root and producing on their own. Personally, it is also one more gift I get from Pittsburgh—like the right grant that I’ve gotten at the right time for the right new idea, effectively the mechanism by which I’ve stayed, year after year, and based my career in what is an increasingly central city.

DW: Thank you for taking the time to do this interview. You’ve mentioned a lot of new music I need to listen to.

[Interview completed 2016-08-17]

Links

- ¹ <http://tug.org/TUGboat/tb24-2/tb77garcia.pdf>
- ² <http://tug.org/TUGboat/tb33-2/tb104garcia.pdf>
- ³ <http://tug.org/TUGboat/Contents/contents37-2.html>
- ⁴ <http://garciadecastro.net/composer>
- ⁵ <http://tinyurl.com/garciadecastroscores>
- ⁶ <http://aliamusicapittsburgh.org>
- ⁷ <http://icareifyoulisten.com/2016/05/5-questions-federico-garcia-de-castro-pittsburgh-new-music-festival-artistic-director>
- ⁸ <http://pghnewmusic.com>
- ⁹ <http://newmusicusa.org/projects/spring-2016-a-ken-ueno-premiere-2>
- ¹⁰ http://youtu.be/A_4nfxShyGM
- ¹¹ <http://youtu.be/bz80VFygnQA>
- ¹² <http://www.nat28.org>

◊ David Walden
<http://tug.org/interviews>

Typographers' Inn

Peter Flynn

Dashing it off

I recently put up a new version of *Formatting Information* (<http://latex.silmariel.ie>), and in the section on punctuation I described the difference between hyphens, en rules, em rules, and minus signs.

In particular I explained how to type a spaced dash — like that, using ‘`dash~---`’ to put a tie before the dash and a normal space afterwards, so that if the dash occurred near a line-break, it would never end up at the start of a line, only at the end. I somehow managed to imply that a spaced dash was preferable to an unspaced one (probably because it’s my personal preference, but certainly not an absolute).

The ensuing discussion on `comp.text.tex` revealed some curious inconsistencies. Petros Travioli very kindly directed me at the Oxford Dictionaries web site at <http://www.oxforddictionaries.com/words/punctuation#dash>, which gives examples of an unspaced em-rule, but in the ‘Read more’ link on that page, we discovered that the examples are actually spaced *en*-rules. This is the practise recommended by Wikipedia’s style guide, which says that the em-rule is not spaced; and the APA style guide agrees with them [1, p 97]. Strunk & White use it unspaced, but don’t actually mention it; but the Associated Press style guide disagrees and says to use spaces, which is what *TUGboat* does (`\thinspace`, in fact).

So what authorities say they think is right, and what publishers actually do, can be very different. I just picked up five books I read since the start of the year:

- Sansom, Ian (2012) *Paper: An Elegy*. Fourth Estate (Harper Collins), London, 2012 (**spaces around en rule**).
- Wilson, Bee (2013) *Consider The Fork*. Penguin, London (**spaces around em rule**).
- Jones, Terry and Alan Ereira (2005) *Mediaeval Lives*. BBC Books, London (**spaces around em rule**).
- Sayers, Dorothy (1942) *The Nine Tailors*. Victor Gollancz, London (**unspaced em rule**).
- Banks, Iain M (2003) *The Player of Games*. Orbit (Macmillan), London (**spaces around en rule**).

I suspect that, as with many points of typographic style, you should follow the conventions of your discipline; but if you have free rein, choose whichever style you think best — but be consistent.

Peter Flynn

X_ELATEX

Back at the ranch, we have been experimenting with X_ELATEX in our workflow, spurred on by two recent requests to use a specific set of OpenType fonts for some GNU/Linux documentation. X_ELATEX offers two major improvements on pdfLATEX: the use of OpenType and TrueType fonts, and the handling of UTF-8 multibyte characters.

Font packages. You can’t easily use the font packages you use with pdfLATEX because the default font encoding is EU1 in the `fontspec` package which is key to using OTF/TTF fonts, rather than the T1 or OT1 conventionally used in pdfLATEX. But late last year Herbert Voß kindly posted a list of the OTF/TTF fonts distributed with TeX Live which have packages of their own for use with X_ELATEX [6].

Table 1: List of font packages supporting X_ELATEX (as of 25 Dec 2015)

accanthis	some Accanthis with CMR
Alegreya	some Alegreya with CMR
AlegreyaSans	some Alegreya Sans with CMSS
cabin	some Cabin with CMSS
caladea	some Caladea with CMR
carlito	some Carlito with CMSS
cinzel	some CINZEL with CMR
ClearSans	some Clear Sans with CMSS
ebgaramond	some EB Garamond with CMR
FiraMono	some Fira Mono with CMTT
FiraSans	some Fira Sans with CMSS
gillius	some Gillius with CMSS
gillius2	some Gillius2 with CMSS
imfellEnglish	some IM FELL English with CMR
libertine	some Libertine with CMR
librebaskerville	some Libre Baskerville with CMR
librecaslon	some Libre Caslon with CMR
LobsterTwo	some Lobster Two with CMR
merriweather	some Merriweather with CMR
mintspirit	some Mint Spirit with CMSS
mintspirit2	some Mint Spirit2 with CMSS
PlayfairDisplay	some Playfair Display with CMR
quattrocento	some Quattrocento with CMR
raleway	some Raleway with CMSS
roboto	some Roboto with CMSS
sourcecodepro	some Source Code Pro with CMTT
sourcesanspro	some Source Sans Pro with CMSS
sourceserifpro	some Source Serif Pro with CMR
universalis	some Universalis with CMSS

These packages, shown in Table 1, work with the `\usepackage` command in the normal way. The

L^AT_EX Font Catalogue has a separate page at <http://www.tug.dk/FontCatalogue/opentypefonts.html> for fonts with OpenType support.

So what do you do about all those other font packages not yet adapted to detect that they are being used in X_EL^AT_EX? Individual font specification in X_EL^AT_EX is slightly different to that in pdfL^AT_EX: it requires the `fontspec` package. This provides three commands which let you specify the three basic families expected by L^AT_EX: `\setmainfont`, `\setsansfont`, and `\setmonofont` (there are also commands for loading individual fonts). The argument is *either* a full fontname like `Times New Roman` or a font filename like `Lato-Hairline.ttf`, and this is where the fun starts, because both methods have advantages and disadvantages for ease of use and portability.

Full fontnames. There are OTF/TTF replacements for many of the pdfL^AT_EX-oriented package fonts, which can be loaded using the full font name, *including spaces*. The T_EX Gyre project has created a set of fonts which work with `fontspec`, equivalent to the old Adobe ‘35’ which have been a mainstay of desktop publishing for many years (see Table 2).

Table 2: Font names for equivalents of the Adobe PostScript ‘35’ fonts

TeX Gyre Adventor	Avant Garde
TeX Gyre Bonum	Bookman
TeX Gyre Chorus	Zapf Chancery
TeX Gyre Cursor	Courier
TeX Gyre Heros	Helvetica
TeX Gyre Pagella	Palatino
TeX Gyre Schola	Century Schoolbook
TeX Gyre Termes	Times

For example, `\setsansfont{TeX Gyre Adventor}`.

If you need the Microsoft Windows Core Fonts which come with most Windows and Mac systems (available as RPM/DEB packages for GNU/Linux), the names are shown in Table 3.

However, the downside with all the other OTF or TTF fonts already installed on a computer is that the user may not know how to find out the full fontname—it’s usually the one shown when you display the system font folder or pull down the font menu in a wordprocessor. The upside is that the full fontname is usually fairly clear and descriptive, and stays built into the font even if you change the filename.

On GNU/Linux systems, where there is no strict rule about where such fonts get installed, you must

Table 3: Font names for using the Microsoft Windows Core Fonts in X_EL^AT_EX

Andale Mono	Andale Mono
Arial	Arial
Arial Black	Arial Black
Comic Sans MS	Comic Sans MS
Courier New	Courier New
Georgia	Georgia
Impact	Impact
Times New Roman	Times New Roman
Trebuchet MS	Trebuchet MS
Verdana	Verdana
Webdings	

For example, `\setsansfont{Comic Sans MS}`.

use your system’s font cache to find a fontname. You create the cache by running the command

```
sudo fc-cache -f -vv
```

You can then use the `fc-list` command to list the fonts it found (the second colon-separated field) and use a filter like `grep (1)` to find the name you want.

Font filenames. You can also load the font by filename, providing the filetype (extension), location (path), and naming pattern for the font variants as options (see Figure 1).

```
\setmainfont[Lato-Hairline][Extension=.ttf,
Path=/usr/share/fonts/truetype/lato/,
UprightFont=*, BoldItalicFont=*BoldItalic,
ItalicFont=*Italic, BoldFont=*Bold]
```

Figure 1: Loading a font by filename

The naming pattern uses an asterisk to represent the filename you gave as the main argument to the command; so in the example, the italic variant would be the file named `Lato-HairlineItalic.ttf`, but you can add whatever punctuation (hyphens, spaces, etc) your filenames require. This method is probably better than the fontname if you know the exact places and names of all your font files, because you can tailor the command to suit your own setup.

The disadvantage is that filenames can be different for the same font file across systems, so there is less portability; but the advantage is that it’s usually easier to find filenames than to find full fontnames. However, you can also create a `.fontspec` file which provides the mapping from filename to fontname, so frequent users can make life much easier for themselves.

Logos

Logotypes are typographic designs or glyph variants of more than one letter, but available as a single glyph (originally, cast as a single piece of metal type). *The* in ATF Garamond is one example, and *T_EX* is another.

The *T_EX* and related logos work fine in Computer Modern, but in other faces the spacing needs adjustment, and this is tedious in the extreme when dealing with many fonts in several faces.

There have been several articles on the problem of adjustments [4, 5], not least from Don Knuth [3]; but Karl Berry and Robin Fairbairns parameterised the macros for *\TeX* and other commands in the *tugboat* document class, which were tuned for CM by Barbara Beeton, the redoubtable editor of *TUGboat*.

Table 4: Example adjustments to logo settings

Typeface	Font	Normal	Italic	Bold	Bold Italic
Computer Modern	lmr	<i>L^AT_EX</i>	<i>L^AT_EX</i>	<i>L^AT_EX</i>	<i>L^AT_EX</i>
Bitstream Charter	bch	<i>L^AT_EX</i>	<i>L^AT_EX</i>	<i>L^AT_EX</i>	<i>L^AT_EX</i>
Avant Garde	pag	<i>L^AT_EX</i>	<i>L^AT_EX</i>	<i>L^AT_EX</i>	<i>L^AT_EX</i>
Lobster Two	LobsterTwo	<i>L^AT_EX</i>	<i>L^AT_EX</i>	<i>L^AT_EX</i>	<i>L^AT_EX</i>
Caladea	Caladea	<i>L^AT_EX</i>	<i>L^AT_EX</i>	<i>L^AT_EX</i>	<i>L^AT_EX</i>
Raleway	Raleway	<i>L^AT_EX</i>	<i>L^AT_EX</i>	<i>L^AT_EX</i>	<i>L^AT_EX</i>

I have borrowed this code and extended it so that you can specify the kerning either side of the letter E in *\TeX* and the letter A in *\LATEX*, as well as their vertical displacement and their scale. Three other arguments let you give the font name, font series, and font shape, so that each combination can have its own set of adjustments.

I'll upload a draft of the package (*flexlogo*) to CTAN once I have a few buglets sorted out.

Afterthought

In this column in *TUGboat* 33:1 [2], I described the problems raised by poorly-broken centered material, especially headings. Hardly a day goes by without me seeing another example, and Figure 2 shows the label from a very fine cheese I brought as a gift to an elderly relative recently. She wanted to know what a ‘cheese beech’ was... see Figure 2.

**SEMI FIRM CHEESE BEECH
SMOKED ON THE FARM**

Maybe they prepared the text in Word and sent it to the designer, who followed it without question (which is unprofessional), or that the vendor has no clue, and the designer neither (which is not unusual, alas). Would it have killed them to rearrange it?

**SEMI-FIRM CHEESE
BEECH-SMOKED ON THE FARM**

But most likely, the designer wanted the lines to get shorter so that they followed the curvature of the label.

**SEMI-FIRM CHEESE
BEECH-SMOKED
ON THE FARM**

Three lines was probably too many, but I still think they made a mistake putting vanity of the design above the usability of the text.



Figure 2: Poorly-broken lines in a description

References

- [1] American Psychological Association. Publication Manual of the American Psychological Association. Technical report, APA, Washington, DC, Jan 2010.
- [2] Peter Flynn. Typographers’ Inn—Formatting and centering. *TUGboat*, 33(1):8–9, Jan 2012. <http://tug.org/TUGboat/tb33-1/tb103inn.pdf>.
- [3] Donald E. Knuth. The *\TeX* Logo in Various Fonts. *TUGboat*, 7(2):101, Jan 1986. <http://tug.org/TUGboat/tb07-2/tb15knutlogo.pdf>.
- [4] Grzegorz Murzynowski. *\LATEX* vs. *\LATEX*—a modification of the logo. *TUGboat*, 29(1):180, Jan 2008. 17th European *\TeX* Conference (EuroBachotEX), Bachotek, Poland. CSTUG, GUST. <http://tug.org/TUGboat/tb29-1/tb91murzynowski-logo.pdf>.
- [5] Jacek Rużycka. *texlogos.sty*: *\LATEX* package for *\LATEX* logos. <http://ctan.org/pkg/texlogos>, Jan 2016.
- [6] Herbert Voß. Re: XeLaTeX/biblatex - anything missing? *comp.text.tex*, Dec 2015. de55baFddkaU1@mid.uni-berlin.de.

◊ Peter Flynn
Textual Therapy Division,
Silmaril Consultants
Cork, Ireland
Phone: +353 86 824 5333
[peter \(at\) silmaril dot ie](mailto:peter(at)silmaril(dot)ie)
<http://blogs.silmaril.ie/peter>

LuaTeX version 1.0.0

Hans Hagen

1 The release

After some ten years of development and testing, on September 9, 2016, we released LuaTeX 1.0.0! It happened at the tenth meeting of the ConTeXt users and developers group in the Netherlands.

Instead of staying below one and ending up with versions like 0.99.1234, we decided that the moment was there to show the TeX audience that LuaTeX is stable enough to lose its beta status. Although functionality has evolved and sometimes been replaced, we have been using LuaTeX ourselves in production right from the start. Of course there are bugs and for sure we will fix them.



Our main objective was and still is to provide a variant of TeX that permits user extensions without the need to adapt the inner workings. We did add a few things here and there but they mostly relate to opening up the inner parts and/or the wish to influence some hard-coded behaviour. Via Lua we managed to support modern functionality without bloating the code or adding more and more dependencies on foreign code. In the process a stable and flexible MetaPost library became part of the engine.

The functionality as present now will stay. We might open up some more parts, we will stepwise clean up the code base while staying as close as possible to the Knuthian original, we will try to document bits and pieces. We might also experiment a bit with better isolation of the backend, and simplify some internals. For that we can use the experimental version but if we diverge too much we may need to give that another name.

We want to thank all those who have tested the betas and helped to make LuaTeX better.

Hans Hagen
Hartmut Henkel
Taco Hoekwater
Luigi Scarso

2 The past

Originally we planned to release the first version a few years ago but our ambitions didn't work out well with that schedule so we finally took a decade to get there. For the record it is good to summarize what happened during those years.

- Around 2005, after we talked a bit about extending TeX in a flexible way and Hartmut added the Lua scripting language to pdfTeX as an experiment. This add-on was inspired by the Lua extension to the SciTE editor that I (still) use.
- At that time one could query counter registers and box dimensions and print strings to the TeX input buffer.
- The Oriental TeX project then made it possible to go forward and come up with a complete interface. For this, Taco converted the code base from Pascal to C, a quite impressive effort.
- We spent more than a year intensively discussing, testing and implementing the interface between TeX and Lua. Many binaries and lots of test code were flying between Taco and my machine as we progressed and decided what directions to go. These were really interesting times.
- In successive years we polished and extended things; in recent years, we cleaned up interfaces, polished more code, filled in gaps and reached the point where we were more or less satisfied.
- The core is still traditional TeX, but has been extended with pdfTeX protrusion and expansion (re-worked) and directional features from Aleph (cleaned up). We did add some extensions (in ε -TeX fashion) but removed most of the ones that we inherited from pdfTeX because Lua could do better.
- The backend and extension interfaces are now mostly separated and although we don't expect to add more backends, it makes the code somewhat cleaner because all kinds of PDF-related issues are no longer mixed with front-end mechanisms.
- The font subsystem is no longer limited to 8-bit fonts. It must be noted that for instance OpenType support is done in Lua, which provides a lot of flexibility. This also serves as an example of extensibility. A small TeX core, independent of libraries, was definitely an objective and it works out well.
- The (rewritten but compatible) hyphenation machinery can use runtime loaded (and extended) patterns. There are a few extensions and of course one can revert to Lua for more.
- Already at an early stage, hyphenation, ligaturing and kerning were separated, which was one step in adding callbacks to nearly every stage in the typesetting process.
- Math supports wide (more than 8-bit) characters too so that one can implement Unicode math easily. The machinery has OpenType math code paths because there are some fundamental differences with traditional TeX math fonts.

- Although the kpse library is still the default interface to the file system, all in- and output can be controlled and intercepted, for instance for input filtering or re-encoding on the fly.
- The token scanner has been opened up so that one can write (simple) parsers. Experimental interception code didn't prove to be useful so that interface has been dropped. We kept it simple and efficient.
- During callbacks related to the node lists, individual nodes can be accessed and manipulated at will. Of course one needs to know a bit about the internals and not mess up the lists to the extent that TeX will choke on it: things that 'can't happen' now can. Most of the original documentation of the code by Don Knuth still applies (which was another objective) but of course directional support and such go beyond that. And it's surprisingly fast.
- Images and reusable boxes are now native nodes; they travel through the system as special kinds of rules instead of whatsits with dimensions. Users can define their own rule types too.

There is more to say but much has been reported already in articles in this and other journals. In the ConTeXt distribution there are four documents describing aspects of the development and choices we have made (`mkiv.pdf`, `hybrid.pdf`, `about.pdf` and `still.pdf`) and we keep writing (`onandon.pdf`). One thing will hopefully be clear by now: the choice of Lua was a good one.

3 The future

The project is driven by ConTeXt users and ConTeXt development which is why we found it proper to release version one at the tenth meeting. Right from the start ConTeXt supported LuaTeX and this means that most mechanisms have been tested in production. There is some risk in this as users then are always forced to update the binary with the macros but the ConTeXt garden provides easy ways to deal with this. In fact, most users switched to the new engine pretty soon after we started rewriting ConTeXt. We greatly appreciate their patience.

Raw performance of LuaTeX is of course less than 8-bit pdftEX but in practice and on modern machines LuaTeX behaves well. In fact, many mechanisms, like native XML handling and MetaPost processing are way faster in ConTeXt MkIV than in the now frozen MkII version. Given the fact that we're using Unicode and more complex fonts, one can safely assume that in ConTeXt the overhead due to delegation to Lua has no real drawbacks.

We will continue development, but functionality will stay stable within versions. The code will be further streamlined and documented. We deliberately postponed some cleanup till after version one. And of course bugs will be fixed. We hope to stepwise improve the manual too. So what will the future bring?

- So far we managed to avoid extensions beyond those needed as part of the opening up. We stick close to Don Knuth's concepts so that existing documentation still conceptually applies. We keep our promise of not adding to the core. But, we might open up (make configurable) some of the remaining hardcoded properties.
- Some node lists could use a bit of (non-critical) cleanup, for instance passive nodes, localpar nodes, and other leftovers. Maybe we should add missing left/right skips.
- We can optimize some callback resolution (more direct) so that we can gain a little performance.
- Inheritance of attributes needs checking and maybe we need to permit some more explicit settings.
- We will move some more code to the API file and plan to update the global PDF and Lua states consistently (there are some leftovers from the early days). Some C macros can probably go away.
- We can possibly minimize some return values of Lua functions and only return nil when we expect multiple calls in line. This might be more efficient. We plan to look into Lua 5.3 but we might well conclude that it's better to stick to 5.2.
- We have to figure out a way to deal with literals in virtual characters. This relates to font switching in the result.
- Maybe we will reorganize some code so that documentation is easier. We hope to continue to stick close to what Don Knuth documents.
- We can clean up and isolate the backend a bit more. We also could add a few more options to delegate actions to Lua and we should get rid of some historic PDF artifacts.

Of course we have some ideas of what to do next but these don't need an extension to the engine because we can use Lua for that.

In that perspective it is tempting to think of a (lean and mean) LuaTeX variant for ConTeXt: a close to traditional core with many hooks and a minimal number of dependencies on libraries and such. In a ConTeXt setup a user needs LuaTeX because all (workflow) related scripts are written in Lua and if additional functionality (like graphic conversions) is needed, it can easily be provided by external programs.

We will not touch the stable version unless it concerns bug fixes and/or simple extensions, but we will keep exposing ConTeXt users to the experimental branch (as we do now). Of course users of other macro packages can pick up binaries from the compile farm that has been set up by Mojea and friends.

So ... be prepared.

◊ Hans Hagen
Pragma ADE
<http://pragma-ade.com>

LuaTeX 0.82 OpenType math enhancements

Hans Hagen

Abstract

LuaTeX 0.82 (and later) have had improvements in OpenType math typesetting.

1 Introduction

When TeX typesets mathematics it makes some assumptions about the properties of fonts and dimensions of glyphs. Due to practical limitations in the traditional eight-bit fonts, such as the number of available characters in a font and a limited number of heights and depths, some juggling takes place. For instance, TeX sometimes uses dimensions as a signal to treat some characters as special. This is not a problem as long as one knows how to make a font and in practice that was done by looking at the properties of Computer Modern to implement similar shapes. After all, there are not that many math fonts around and basically there is only one engine that can deal with them properly.

However, when Microsoft set the standard for OpenType math fonts it also steered the direction of their use in rendering mathematics. This means that the LuaTeX engine, which handles OpenType fonts, has to implement some alternative code paths. At the start, this involved a bit of gambling because there was no real specification; since then we now have a better picture. One of the more complex changes that took place is in the way italic correction is applied. A dirty way out of this dilemma would be to turn the math fonts into virtual ones that match traditional TeX properties, but this would not be a nice solution.

It must be noted that in the process of implementing support for the new fonts, Taco (Hoekwater) turned some noad types (see below) into a generic noad with a subtype. This simplified the transition. At the same time, a lot of detailed control was added in the way successive characters are spaced.

In LuaTeX before 0.85, the italic correction was always added when a character got boxed (a frequently used preparation in the math builder). Now this is only done for the traditional fonts because, concerning italic correction, the OpenType standard states:¹

1. When a run of slanted characters is followed by a straight character (such as an operator or a delimiter), the italic correction of the last glyph is added to its advance width.

¹ Recently version 1.8 has been published on the Microsoft website.

2. When positioning limits on an N-ary operator (e.g., integral sign), the horizontal position of the upper limit is moved to the right by half of the italic correction, while the position of the lower limit is moved to the left by the same distance.
3. When positioning superscripts and subscripts, their default horizontal positions are also different by the amount of the italic correction of the preceding glyph.

And, with respect to kerning:

4. Set the default horizontal position for the superscript as shifted relative to the position of the subscript by the italic correction of the base glyph.

I must admit that when the first implementation showed up, my natural reaction to unexpected behaviour was just to compensate for it. One such solution was simply not to pass the italic correction to the engine and deal with it in Lua. In practice, that didn't work well for all cases; one reason was that the engine saw the combination of old fonts as a new one and followed a mixed code path.² Another approach I tried was a mix of manipulated italic values and Lua, but finally, as specifications settled I decided to leave it to the engine completely, if only because successive versions of LuaTeX behaved much better.

So, as we were closing in on the first stable release of LuaTeX (1.0.0 was released on September 27, 2016; this note was mostly written in the early part of 2016), I decided to fix the pending issues and sat down to look at the math-related code. I must admit that I had never looked in depth into that part of the machinery. In the next sections I will discuss some of the outcomes of this exercise.

I will also discuss some extensions that have been on the agenda for years. They are rather generic and handy, but I must also admit that the MkIV code related to math has so many options to control rendering that I'm not sure if they will ever be used in ConTeXt. Nevertheless, these generic extensions fit well into the set of basic features of LuaTeX.

2 Italic correction

As stated above, the normal code path included italic correction in all the math boxes made. This meant that, in some places, the correction had to be removed and/or moved to another place in the chain. This is a natural side effect of the fact that TeX runs over the intermediate list of math nodes

² ConTeXt employed Unicode math right from the start of LuaTeX.



Figure 1: Italic correction examples (1): superscripts shifted right and subscripts left.



Figure 2: Italic correction examples (2): plain integral vs. integral with limits

(noads) and turns them into regular nodes, mostly glyphs, kerns, glue and boxes.

The complication is not so much the italic corrections themselves, because we could just continue to do the same, but the fact that these corrections are to be interpreted differently in case of integrals. There, the problem is that we have to (kind of) look backward at what is done in order to determine what italic corrections are to be applied.

The original solution was to keep track of the applied correction via variables but that still made some analysis necessary. In the new implementation, more information is stored in the processed noads. This is a logical choice given that we have already added other information. It also makes it possible to fix cases that will (for sure) show up in the future.

In figure 1 we show two examples of inline italic correction. The superscripts are shifted to the right and the subscripts to the left. In the case of an integral sign, we need to move half the correction. This is triggered by the `\nolimits` primitive. In figure 2 we show the difference between just an integral character and one tagged as having limits.³

The amount of correction, if present at all, depends on the font, and in this document we use DejaVu math. Figure 3 shows a few variants. As you can see, the amount of correction is highly font dependent.

3 Vertical delimiters

When we go into display math, there is a good chance that an integral has to be enlarged. The integral sign in Unicode has slot 0x222B, so we can define a bigger one as follows:

```
\def\standardint{\Umathchar "1 "0 "222B }
\def\wrappedint{\mathop{\Umathchar "1 "0 "222B}}
\def\biggerint{\mathop{
  \Uleft height3ex depth3ex axis
  }
```

³ We show some boxes so that you can get an idea what TeX is doing. Essentially, TeX puts superscripts and subscripts on top of each other with some kern in between and then corrects the dimensions.

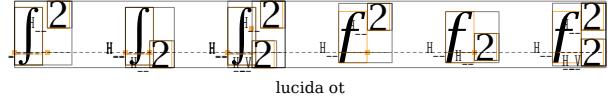
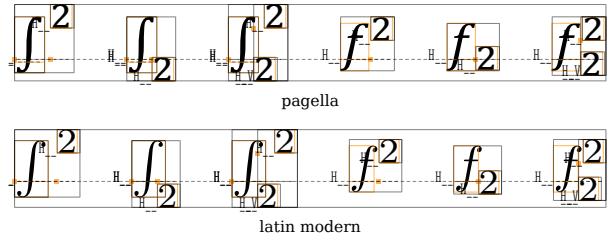
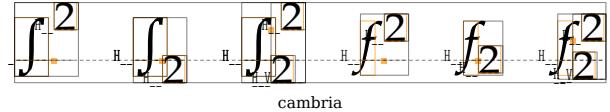


Figure 3: Italic correction examples (3): correction amounts are font-dependent.

```
\Udelim "0 "0 "222B \Uright .}
\def\evenbiggerint{\mathop{
  \Uleft height 6ex depth 6ex axis
  \Udelim "0 "0 "222B \Uright .}
```

The `axis` keyword will apply a shift up over the size of the current styles math axis. We use this in some examples as:

```
$
\displaystyle\standardint ^a_b\enspace
\displaystyle\wrappedint ^a_b\enspace
\displaystyle\biggerint ^a_b\enspace
\displaystyle\evenbiggerint ^a_b\enspace
$
```

In figure 4 you can see some subtle differences. The wrapped version doesn't shift the superscript and subscript. The reason is that the operator is hidden in its own wrapper and the scripts attach at an outer level. So, unless we start analyzing the innermost noad and apply that to the outer, we cannot know the shift. Such analyzing is asking for problems: where do we stop and what slight variations do we take into account? It's better to be predictable.

Another observation is that Latin Modern does not provide (at least not yet) large integrals at all.

The following four cases are equivalent:

```
\Uleft height 3ex depth 3ex axis
\Udelim "0 "0 "222B
\Uright .
```



```
\Uleft .
\Uright height 3ex depth 3ex axis
\Udelim "0 "0 "222B
```



```
\Uleft .
\Umiddle height 3ex depth 3ex axis
```

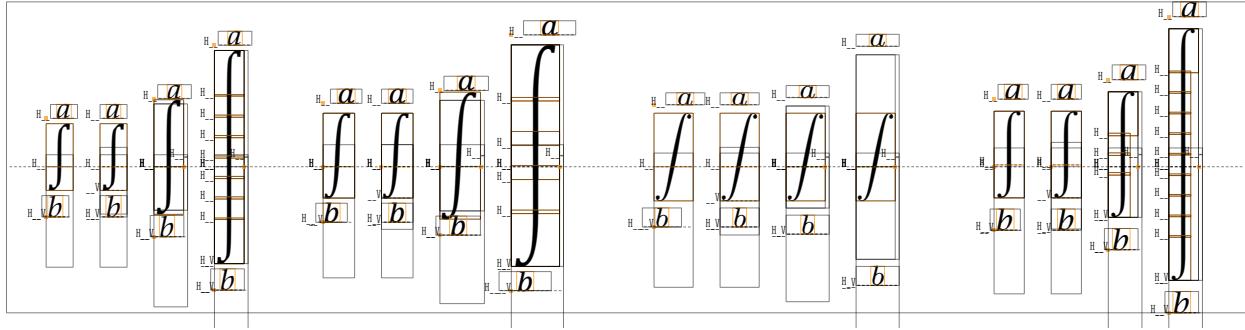


Figure 4: Comparison of integral variants (standard, wrapped, bigger, even bigger) among fonts: TeX Gyre Pagella, Cambria, Latin Modern, and Lucida OT.

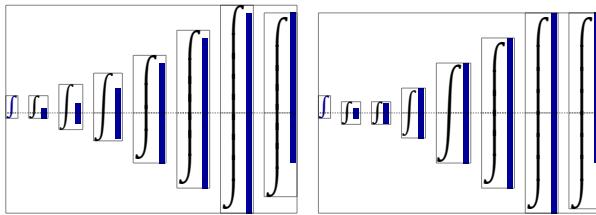


Figure 5: Cambria integrals, adaptive; `axis` left, `noaxis` right.

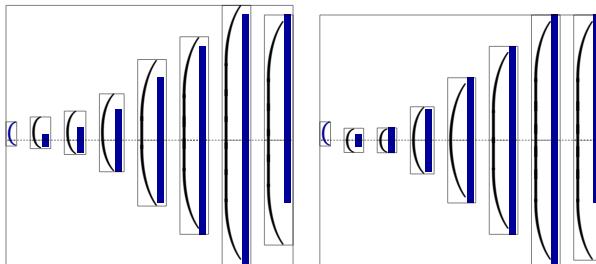


Figure 6: Cambria left parenthesis, adaptive; `axis` left, `noaxis` right.

```
\Udelim "0 "0 "222B
\Uright .
\Uleft .
\Umiddle height 3ex depth 3ex axis
\Udelim "0 "0 "222B
\Uright .
```

However, because this all looks a bit clumsy, we now provide a new primitive:

```
\Uvextensible
  height <dimension>
  depth <dimension>
  [no]axis
  exact
  <delimiter>
```

The symbol to be constructed will have size `height` plus `depth`. When an `axis` is specified, the symbol will be shifted up, which is normally the case

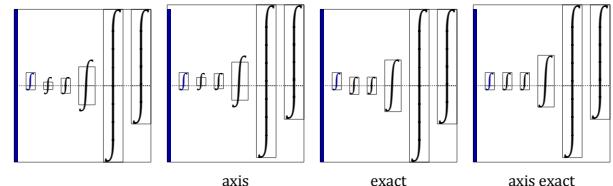


Figure 7: Cambria integrals, with dimensions.

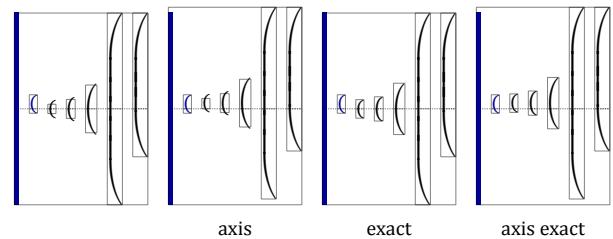


Figure 8: Cambria left parenthesis, with dimensions.

for such symbols. The keyword `exact` will correct the dimensions when no exact match is made, and this can be the case as long as we use the stepwise larger glyphs and before we end up using the composed shapes. When no dimensions are specified, the normal construction takes place and the only keyword that can be used then is `noaxis` which keeps the axis out of the calculations. After about a week of experimenting and exploring options, this combination made most sense, read: no fuzzy heuristics but predictable behaviour. After all, one might need different solutions for different fonts or circumstances and the applied logic (and expectations) can (and will, for sure) differ per macro package. Figures 5–8 show some examples.

4 Horizontal delimiters

Horizontal extenders also have some new options. Although one can achieve similar results with macros, the following might look a bit more natural. Also,

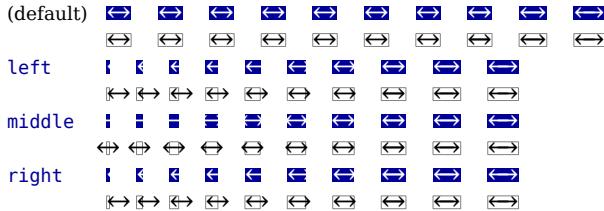


Figure 9: Stepwise wider `\Uhexxtensible` with options (Cambria).

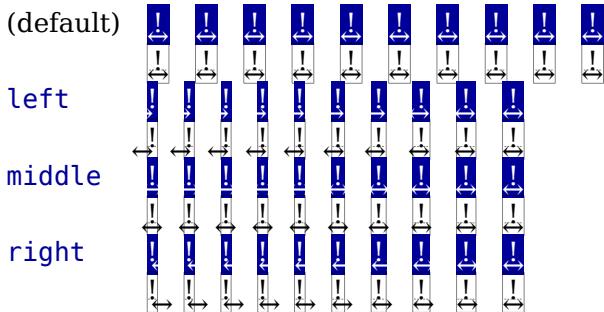


Figure 10: Stepwise wider `\Udelimiterunder` with options (Cambria).

some properties are lost once the delimiter is constructed, so macros can become complex when trying to determine the original dimensions involved.

We start with the new `\Uhexxtensible` primitive that accepts a dimension. It's just a variant of the over and under delimiters with no content part.

```
\Uvextensible
  height <dimension>
  depth <dimension>
  left | middle | right
  <family>
  <slot>
```

So for example you can say:

```
$\Uhexxtensible width 30pt 0 "2194$
```

The `left`, `middle` and `right` keywords are only interpreted when the requested size can't be met due to stepwise larger glyph selection (i.e., before we start using arbitrary sizes made of snippets). Figure 9 shows what we get when we step from 2–20 points by increments of 2 points in Cambria.

The dimensions and options can also be given to the four primitives:

```
\Uoverdelimiter \Uunderdelimiter
\Udelimiterover \Udelimiterunder
```

Figure 10 shows what happens when the delimiter is smaller than requested. The source for the samples looks like this:

```
$\Udelimiterunder width 1pt 0 "2194
  {\hbox{\strut !}}
```

$\Sigma_{800} \Sigma_{900} \Sigma_{1000} \Sigma_{1100} \Sigma_{1200} \Sigma_{1300} \Sigma_{1400} \Sigma_{1500}$
xits - has variants

$\Sigma_{800} \Sigma_{900} \Sigma_{1000} \Sigma_{1100} \Sigma_{1200} \Sigma_{1300} \Sigma_{1400} \Sigma_{1500}$
cambria - lacks variants

$\Sigma_{800} \Sigma_{900} \Sigma_{1000} \Sigma_{1100} \Sigma_{1200} \Sigma_{1300} \Sigma_{1400} \Sigma_{1500}$
pagella - lacks variants

Figure 11: Using `overlay` in `\Umathaccent`.

When no dimension is given the keywords are ignored as it makes no sense to mess with the extensible in that case.

5 Accents

Many years ago, I observed that overlaying characters (which happens when we negate an operator which has no composed negation glyph) didn't always give nice results and, therefore, a tracker item was created. When going over the todo list, I ran across a suggested patch by Khaled Hosny that added an overlay accent type. As the suggested solution fits in with the other extensions, a variant has been implemented.

The results definitely depend on the quality and completeness of the font, so here we will use XITS. The placement of an `overlay` also depends on the top accent shift as specified in the font for the used glyph. Instead of a fixed criterion for trying to find the best match, an additional `fraction` (numerator) parameter can be specified. A value of 800 means that the target width is 800/1000.

The `\Umathaccent` command now has the following syntax:

```
\Umathaccent
  [top | bottom | overlay]
  [fixed]
  [fraction <number>]
  <delimiter>
  {{<content>}}
```

When we have an overlay, the fraction concerns the height; otherwise it concerns the width of the nucleus. In both cases, it is only applied when searching for stepwise larger glyphs, as extensibles are not influenced. An example of a specification is:

```
\Umathaccent
  overlay "0 "0 "0338
  fraction 950
  {{\Umathchar"1"0"2211}}
```

Figure 11 shows what we get when we use different fractions (from 800 up to 1500 with a step of 100). We see that `\overlay` is not always useful.

Normally you can forget about the factor because overlays make most sense for inline math, which

	$x + \frac{a}{b} + x$	$x + \frac{1}{2} + x$	$x + \left(\frac{a}{b}\right) + x$	$x + \left(\frac{1}{2}\right) + x$
exact	$x + \frac{a}{b} + x$	$x + \frac{1}{2} + x$	$x + \left(\frac{a}{b}\right) + x$	$x + \left(\frac{1}{2}\right) + x$
noaxis	$x + \frac{a}{b} + x$	$x + \frac{1}{2} + x$	$x + \left(\frac{a}{b}\right) + x$	$x + \left(\frac{1}{2}\right) + x$
exact noaxis	$x + \frac{a}{b} + x$	$x + \frac{1}{2} + x$	$x + \left(\frac{a}{b}\right) + x$	$x + \left(\frac{1}{2}\right) + x$

Figure 12: Skewed fraction results in Latin Modern.

uses relatively small glyphs, so we can get $\mathbf{x} \mathbf{x} \mathbf{x} \mathbf{x}$ with the following code:

```
$\Umathaccent overlay "0 "0 "0338 {x}$
$\Umathaccent overlay "0 "0 "0338 {\tf x}$
$\Umathaccent overlay "0 "0 "0338 {\tf xxx}$
```

A normal accent can also be influenced by `fraction`:

$\widehat{a \times b}$ $\widehat{a \times b}$ $\widehat{a \times b}$ $\widehat{a \times b}$ $\widehat{a \times b}$

6 Fractions

A normal fraction has a reasonable thick rule but as soon as you make it bigger you will notice a peculiar effect:

$x + \left(\frac{a}{b}\right)$				
1pt	2pt	3pt	4pt	5pt

Such a fraction is specified as:

```
x + { {a} \abovewithdelims () 5pt {b} }
```

A new keyword `exact` avoids the excessive spacing:

```
x + { {a} \abovewithdelims () exact 5pt {b} }
```

Now we get:

$x + \left(\frac{a}{b}\right)$				
1pt	2pt	3pt	4pt	5pt

One way to get consistent spacing in such fractions is to use struts:

```
x + { {\strut a} \abovewithdelims () exact 5pt
      {\strut b} }
```

Now we get:

$x + \left(\frac{a}{b}\right)$				
1pt	2pt	3pt	4pt	5pt

Yet another way to increase the distance between the rule and text a bit is:

```
\Umathfractionnumvgap \displaystyle4pt
\Umathfractiondenomvgap\displaystyle4pt
```

This looks quite consistent:

$x + \left(\frac{a}{b}\right)$				
1pt	2pt	3pt	4pt	5pt

Here we use code like:

```
\displaystyle x +
{{a} \abovewithdelims() exact 2pt {b}}
```

Using struts, it is best to zero the gap:

$x + \left(\frac{a}{b}\right)$				
1pt	2pt	3pt	4pt	5pt

Here we use code like:

```
\displaystyle x +
{{\strut a} \abovewithdelims()
 exact 2pt {\strut b}}
```

7 Skewed fractions

The math parameter table contains values specifying horizontal and vertical gaps for skewed fractions. Some guessing is needed in order to implement something that uses them, so we now provide a primitive similar to the other fraction related ones but with a few options that one can use to influence the rendering. Of course, a user can mess around directly with the parameters `\Umathskewedfractionhgap` and `\Umathskewedfractionvgap`.

The syntax used here is:

```
{ {1} \Uskewed / <options> {2} }
{ {1} \Uskewedwithdelims / () <options> {2} }
```

The options can be `noaxis` and `exact`, a combination of them or just nothing. By default we add half the axis to the shifts and also by default we zero the width of the middle character. For Latin Modern, the results are shown in figure 12.

8 Side effects

Not all bugs reported as such are really bugs. Here is one that came from a misunderstanding: In Eijkhout's *TeX by Topic*, the rules for handling styles in scripts are described as follows:

- In any style superscripts and subscripts are taken from the next smaller style. Exception: in display style they are taken in script style.

- Subscripts are always in the cramped variant of the style; superscripts are only cramped if the original style was cramped.
- In an $\dots \backslash over \dots$ formula in any style the numerator and denominator are taken from the next smaller style.
- The denominator is always in cramped style; the numerator is only in cramped style if the original style was cramped.
- Formulas under a \sqrt or \overline are in cramped style.

In LuaTeX, one can set the styles in more detail, which means that you sometimes have to set both normal and cramped styles to get the effect you want. If we force styles in the script using \scriptstyle and \crampedscriptstyle we get the following (all render the same):

```
default      bx=xx  
script       bx=xx  
crampedscript bx=xx  
              x=xx
```

This is coded as follows:

```
$b_{\{x=xx\}}^{\{x=xx\}}$  
$b_{\{\scriptstyle x=xx\}}^{\{\scriptstyle x=xx\}}$  
$b_{\{\crampedscriptstyle x=xx\}}^{\{\crampedscriptstyle x=xx\}}$  
`{\crampedscriptstyle x=xx}$
```

Now we set the following parameters:

```
\Umathordrelspacing\scriptstyle=30mu  
\Umathordordspacing\scriptstyle=30mu
```

This gives:

```
default      bxx=xx = x x  
script       bxx=xx = x x  
crampedscript bx=xxx=xx
```

Since the result is not what is expected (visually), we should say:

```
\Umathordrelspacing\scriptstyle=30mu  
\Umathordordspacing\scriptstyle=30mu  
\Umathordrelspacing\crampedscriptstyle=30mu  
\Umathordordspacing\crampedscriptstyle=30mu
```

Now we get:

```
default      bxx = x x  
script       bxx = x x  
crampedscript bxx = x x
```

mode	down	up	
0	dynamic	dynamic	$\text{CH}_2 + \text{CH}_2^+ + \text{CH}_2^2$
1	<i>d</i>	<i>u</i>	$\text{CH}_2 + \text{CH}_2^+ + \text{CH}_2^2$
2	<i>s</i>	<i>u</i>	$\text{CH}_2 + \text{CH}_2^+ + \text{CH}_2^2$
3	<i>s</i>	<i>u+s-d</i>	$\text{CH}_2 + \text{CH}_2^+ + \text{CH}_2^2$
4	<i>d+(s-d)/2</i>	<i>u+(s-d)/2</i>	$\text{CH}_2 + \text{CH}_2^+ + \text{CH}_2^2$
5	<i>d</i>	<i>u+s-d</i>	$\text{CH}_2 + \text{CH}_2^+ + \text{CH}_2^2$
$\text{CH}_2 + \text{CH}_2^+ + \text{CH}_2^2$	$\text{CH}_2 + \text{CH}_2^+ + \text{CH}_2^2$	$\text{CH}_2 + \text{CH}_2^+ + \text{CH}_2^2$	
0	1	2	
$\text{CH}_2 + \text{CH}_2^+ + \text{CH}_2^2$	$\text{CH}_2 + \text{CH}_2^+ + \text{CH}_2^2$	$\text{CH}_2 + \text{CH}_2^+ + \text{CH}_2^2$	
3	4	5	

Figure 13: The effect of setting $\mathsf{mathscriptsmode}$.

9 Fixed scripts

We have three parameters that are used for anchoring superscripts and subscripts, alone or in combinations.

```
d \Umathsubshiftdown  
u \Umathupshiftup  
s \Umathsubsupshiftdown
```

When we set $\mathsf{mathscriptsmode}$ to a value other than zero, these are used for calculating fixed positions. This is something that is needed in, for instance, chemical equations. You can manipulate the mentioned variables to achieve different effects, and the specifications are shown in figure 13, with enlarged examples below the table.

10 Remark

The changes that we have made are hopefully not too intrusive. Instead of extending existing commands, new ones were introduced so that compatibility should not be a significant problem. To some extent, these extensions violate the principle that extensions should be done in Lua, but TeX being a math renderer and OpenType replacing old font technology, we felt that we should make an exception here. Hopefully, not too many bugs were introduced.

◊ Hans Hagen
Pragma ADE
<http://pragma-ade.com>

Introducing LaTeX Base

Gareth Aye

LaTeX Base (<https://latexbase.com>) is a web-based L^AT_EX editor that provides many useful features such as

- live, compile-as-you-type document preview,
- one-click document publishing and sharing,
- offline mode,
- integrations with file storage services like Google Drive and Dropbox,
- syntax highlighting,
- and familiar keyboard shortcuts for Vim and Emacs users.

This article focuses on the technical side of the interactions between LaTeX Base and L^AT_EX. However, we're eager to hear from users about their experiences using the service; please feel free to reach out to us at team@latexbase.com with suggestions or if you'd like to beta test new features. Many features do require a premium membership.

Figure 1 shows the first page presented after loading, with simple LaTeX source on the left and the preview output on the right.

1 Offline mode

One unique aspect of LaTeX Base amongst web applications is that it can be used with or without an Internet connection! This is possible thanks to “service workers”: a recent development in the web platform that allows applications to intercept and cache network requests. That means that you can write your papers on a plane, in a park, or anywhere in between. While LaTeX Base isn't the first web-based L^AT_EX editor, it is the only one with this capability.

If you're a programmer as well as a L^AT_EX enthusiast, you may suspect that there's a bit more to the story. Service workers allow us to cache the editor, but (L)TeX doesn't run in the browser (it's ordinarily compiled to machine code). How can LaTeX Base compile documents without an Internet connection if it's limited to executing JavaScript?

2 Enter Emscripten

The answer lies in a fascinating tool that came out of Mozilla's research group a few years ago called Emscripten (wikipedia.org/wiki/Emscripten), which compiles LLVM bytecode to JavaScript. Many codebases such as Unreal Engine, Bullet Physics, and the Lua programming language (which appears to be of some interest to the greater L^AT_EX community) have been ported from C/C++ to JavaScript using Emscripten. In building LaTeX Base, pdf_La_TE_X was

compiled to LLVM bytecode using clang and from LLVM to JavaScript using Emscripten.

Whereas similar services send users' documents to servers with installed L^AT_EX compilers and packages, LaTeX Base sends the compiler and packages to the browser. In addition to making offline mode possible, compiling in the browser also allows LaTeX Base to compile documents quickly and often—so much so that we can offer a real-time preview instead of requiring the user to compile manually.

3 Packages

Our design goal when considering the issue of packages was to make a large number of packages (eventually anything hosted on the CTAN registry) available to users while only ever downloading the packages needed to compile their documents. What we came up with is lazy package loading. Every time you include a new package in a document with LaTeX Base, you'll download it from our servers. When you use that package in the future it'll be cached in your browser. For this reason, using packages that a user hasn't previously used while offline will not work. You can also expect compiling to take slightly longer the very first time you use a package.

For the time being, we only support a small number (around 25) of the most commonly used packages, but our roadmap includes extending support to arbitrary hosted packages.

4 Images

The only way that our implementation of L^AT_EX differs from a standard compiler is in how we handle external files (like images). We don't currently give users direct access to the virtual Emscripten filesystem that L^AT_EX sees when it's running on <https://latexbase.com>. Instead, we support calling \includegraphics with an image url that we'll fetch and preload in Emscripten's virtual filesystem. When you download your documents, we automatically convert these url identifiers to simple file names and bundle the downloaded images so that no changes are necessary to compile documents elsewhere.

Our roadmap also includes allowing users to upload images and other local resources rather than supplying urls.

5 Conclusion

Web applications are great. They allow users to use software without permanently installing it. The abilities and permissions they're granted by default are very limited compared to native applications, so

The screenshot shows the LaTeX Base interface. At the top, there's a navigation bar with links for IMPORT, EXPORT, LANGUAGE, OVERVIEW, PRICING, and SIGN UP. The word COUNT is displayed as WORDS 135, and a green button labeled READY is visible. Below the navigation is a toolbar with icons for document, search, refresh, settings, and preview. The main area is divided into two sections: LATEX (left) and PREVIEW (right). The LATEX section contains the source code for 'Getting started.tex'. The PREVIEW section shows the rendered document, which includes a title 'Getting started', a date 'September 11, 2016', and a welcome message with a mathematical expression and an image of a raptor.

```

1 \documentclass[12pt]{article}
2 \usepackage{amsmath}
3 \usepackage{hyperref}
4 \usepackage{graphicx}
5 \title{Getting started}
6 \date{\today}
7
8 \begin{document}
9 \maketitle
10
11 Welcome to LaTeX Base, a web-based LaTeX editor with live document preview!
12 Here are some things to try --
13
14 \begin{itemize}
15   \item edit the document name above by typing in the input field
16   \item make changes to the body on the left and see the preview update
17   \item check the compiler output by clicking the log button
18   \item format a mathematical expression like
19     
$$\frac{1}{2}\int_{-\infty}^{\infty} e^{-\frac{x^2}{2}} dx$$

20   \item download the document as a pdf by selecting Save To > Local Filesystem
21     (or by clicking the cloud download button)
22   \item include an image by url like this one
23     \hspace*{0.5in}
24     \includegraphics[alt="A raptor image."]{https://latexbase.com/images/raptor.jpg}
25   \item export your work to Dropbox or Google Drive
26 \end{itemize}
27
28 Editing single page documents online is free. View premium plans and
29 pricing at
30 \url{https://latexbase.com/static/pricing} to enjoy unlimited document
31 editing
32 (online or offline) and a variety of other useful features. Thanks for
33 trying
34 out our service and don't hesitate to get in touch at
35 \url{support@latexbase.com}!
36
37 \end{document}

```

Figure 1: Getting Started page of LaTeX Base.

they are preferable from a practical security perspective. They're built on open standards, so they run anywhere and don't need to be sanctioned by any organization or app store. Using cutting edge tools, LaTeX Base is able to offer many advanced capabilities right in the browser. In this author's (absolutely biased) opinion, it's on its way to becoming the best way to write \LaTeX documents.

In closing, I want to recognize Mozilla, not only for their tremendous standards work that's made the web the wonderful thing it is today, but also for their work on components that made LaTeX Base possible including Ace, Emscripten, PDF.js, and localforage.

About the author

Gareth is a New York native living in Portland, OR, with his wife Alison and toddler Albee. He received a BA in Computer Science with a Math minor from Middlebury College and worked in software development, most recently as an engineering lead at Mozilla, before building LaTeX Base. In his free time, he enjoys playing jazz piano and chess.

◊ Gareth Aye
<https://latexbase.com>

Computer Modern Roman fonts for ebooks

Martin Ruckert

How it all started

Last year on February 19, I looked at the first version of my first ebook and I was shocked. I had just finished the printed version of “The MMIX Supplement for The Art of Computer Programming” [5] and Donald Knuth had provided extensive help to make its appearance match the books in his series. Donald Knuth had developed **TEX**, **METAFONT**, and the Computer Modern Roman (CMR) type faces especially to be able to typeset “The Art of Computer Programming” [3] in the best possible quality. But when I looked at my newly bought Kindle Paperwhite—not the most expensive, but still a decent ebook reader—what I saw (Figure 1) did not resemble even remotely what you would expect from **TEX** and friends.

I studied the specification of the epub format and found that TrueType or OpenType fonts should work with it. My first attempt with the TrueType versions of the CMR fonts failed because I had overlooked the few instances where the characters in the book were not pure ASCII. So I switched to the Computer Modern Unicode (CMU) version of the fonts [4], and mailed the publisher the first long list of change requests including the request to use these fonts. When I received the next version of my ebook, Dayna Isley, the digital development editor responsible for the ebook, wrote: “I’m finding that embedded fonts are not well supported across Kindle apps and devices. In most cases, the fonts default to the standard Kindle fonts. Kindle for PC and Paperwhite support embedded fonts, but the body font is difficult to read (very faint) and therefore not effective.” And see for yourself (Figure 2), she was right.

My schedule was tight, I was teaching 18 credit hours that semester, and aside from the fonts there were more and bigger problems to be solved before the ebook could be released. So we settled for a selection of standard ebook fonts and moved on. The final ebook uses Baskerville fonts for the main text body. It is no match for the printed version, but it was a good compromise given the limitations of time and technique.

Now, a year later, I decided to get back to the problem of ebook production with more time to my disposal: I plan to use my sabbatical in 2017 to build a prototype ebook renderer that uses the algorithms of **TEX** for ebook layout and a front-end

The multiplicative hashing scheme is equally easy to do, but it is slightly harder to describe because we must imagine ourselves working with fractions instead of with integers. Let w be the word size of the computer, so that w is usually 2^{32} or 2^{64} for MMIX; we can regard an integer A as the fraction A/w if we imagine the radix point to be at the left of the word. The method is to choose some integer constant A relatively prime to w , and to let

$$h(K) = \left\lfloor M \left(\left(\frac{A}{w} K \right) \bmod 1 \right) \right\rfloor. \quad (4)$$

Fig. 1: First version of my ebook

The multiplicative hashing scheme is equally easy to do, but it is slightly harder to describe because we must imagine ourselves working with fractions instead of with integers. Let w be the word size of the computer, so that w is usually 2^{32} or 2^{64} for MMIX; we can regard an integer A as the fraction A/w if we imagine the radix point to be at the left of the word. The method is to choose some integer constant A relatively prime to w , and to let

$$h(K) = \left\lfloor M \left(\left(\frac{A}{w} K \right) \bmod 1 \right) \right\rfloor. \quad (4)$$

Fig. 2: Second version of my ebook

that translates **TEX** input to an intermediate representation that can be used by the new rendering engine. In preparation for this project, I started to identify those subproblems which I would need to ignore in order to have a reasonable sized project. This brought me back to investigating the font issue.

ebook versus Preview

One of the good programs to view **TEX** output on-screen is YAP, which comes with MiKTEX [6]. YAP is an acronym standing for Yet Another Previewer. The word “previewer” indicates that it is the purpose of the program to give the user an advance view, an approximation of what one should expect to see on paper. The paper version is the “real thing” and the electronic version is only an intermediate step in its production. An indication of this attitude is the selection of fonts. In its default configuration, YAP uses **METAFONT** in **ljfour** mode to generate bitmap fonts. These bitmaps are optimized for a (once) popular 600 dpi laser printer. YAP scales them down to display the **TEX** output on the low resolution computer screen, and if you reduce down-scaling, you can see precisely, down to the last pixel, what you are supposed to get on paper.

The situation has changed significantly with the introduction of ebooks. Now books are produced

specifically for reading on some kind of computer screen. The electronic rendering is no longer an approximation of something yet to come, it is the final product.

The built-in font rendering software (and hardware) of computers usually does not support METAFONT generated bitmapped fonts. It supports TrueType or OpenType outline fonts. These are now the de facto standards. The emerging universal standard in character encoding is, perhaps due to the World Wide Web, the UTF-8 encoding. Outline fonts can be scaled to any resolution desired, but as we will shortly see, this is not sufficient for optimal on-screen reading.

Rendering Computer Modern Roman fonts

To investigate the rendering of the available CMR fonts on current electronic devices, a reference rendering is needed. I choose (somewhat but not completely arbitrarily) my personal copy of *The METAFONTbook* and picked the second paragraph of the preface [2, page v]. It reads, typeset below in 10pt Computer Modern Roman as in the printed book:

“Modern printing equipment based on raster lines—in which metal “type” has been replaced by purely combinatorial patterns of zeroes and ones that specify the desired position of ink in a discrete way—makes mathematics and computer science increasingly relevant to printing. We now have the ability to give a completely precise definition of letter shapes that will produce essentially equivalent results on all raster-based machines. Moreover, the shapes can be defined in terms of variable parameters; computers can “draw” new fonts of characters in seconds, making it possible for designers to perform valuable experiments that were previously unthinkable.”

I then took a photograph of this paragraph from the book (Figure 3) which I will use as my reference for the Computer Modern Roman 10pt font from now on. Apart from the plain text, the photograph contains an insert with the first two letters magnified six times. In comparing the different font renderings, one should pay special attention to the thickness of the different strokes of the “M” and the rounding of the “o”. As a first observation you might notice that the font as shown in figure 3 appears to be much heavier than the same font in the previous paragraph—depending of course on how you printed or rendered this article in order to read it. While it is impossible for me to avoid the effects that your rendering software and your output device

Modern printing equipment based on raster has been replaced by purely combinatorial patterns ify the desired position of ink in a discrete way—n puter science increasingly relevant to printing. We r completely precise definition of letter shapes that w alent results on all raster-based in terms of variable paramete in seconds, making it possible for were previously unthinkable.



Fig. 3: From *The METAFONTbook*

Modern printing equipment based on raster has been replaced by purely combinatorial patterns ify the desired position of ink in a discrete way—n puter science increasingly relevant to printing. We n completely precise definition of alent results on all raster-based in terms of variable parameters; seconds, making it possible for were previously unthinkable.

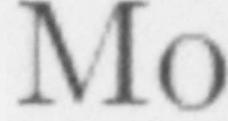


Fig. 4: CMU font, ebook

Modern printing equipment based on raster has been replaced by purely combinatorial patterns ify the desired position of ink in a discrete way—n puter science increasingly relevant to printing. We n completely precise definition of letter shapes that w alent results on all raster-based in terms of variable parameters; seconds, making it possible for were previously unthinkable.



Fig. 5: LM font, ebook

will have on the appearance of fonts, I can reasonably hope that the reproduction of the photographs shown in this article preserve the relative differences which I observed on my output devices.

I have tried my best to take all the photographs in this article under identical conditions, for the sake of comparison. Using a good camera (Canon EOS 60 D, 18Mpixel, 18mm–135mm lens, 1/15s, 5300K, ISO320, 56mm focal length, 16 aperture), I took all photographs under identical light conditions and post-processed the raw images in the same way, trying to reproduce the differences in appearance as well as possible.

TrueType, OpenType, and Unicode fonts

As a first example, lets look at the rendering on my ebook reader (Kindle Paperwhite 2, 1024x758 pixels, 212dpi) using the OpenType Computer Modern

Modern printing equipment based on raster has been replaced by purely combinatorial patterns ify the desired position of ink in a discrete way—nputer science increasingly relevant to printing. We n completely precise definition of letter shapes that walent results on all raster-based in terms of variable parameters: seconds, making it possible for were previously unthinkable.



Fig. 6: CMU Font, smart-phone

Modern printing equipment based on raster has been replaced by purely combinatorial patterns ify the desired position of ink in a discrete way—nputer science increasingly relevant to printing. We n completely precise definition of letter shapes that walent results on all raster-based in terms of variable parameters: seconds, making it possible for were previously unthinkable.

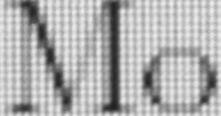


Fig. 7: CMU Font, laptop

Modern printing equipment based on raster has been replaced by purely combinatorial patterns ify the desired position of ink in a discrete way—nputer science increasingly relevant to printing. We n completely precise definition of letter shapes that walent results on all raster-based in terms of variable parameters: seconds, making it possible for were previously unthinkable.



Fig. 8: METAFONT at 142dpi, laptop

Modern printing equipment based on raster has been replaced by purely combinatorial patterns ify the desired position of ink in a discrete way—nputer science increasingly relevant to printing. We n completely precise definition of letter shapes that walent results on all raster-based in terms of variable parameters: seconds, making it possible for were previously unthinkable.



Fig. 9: METAFONT at 4 x 142dpi, laptop

Unicode (CMU) fonts that I had tried already for my ebook (Figure 2). Figure 4 shows how the ebook renders this outline font using the built-in rendering engine. Comparing it with the printed book (Figure 3), it is obvious that the rendering lacks contrast and looks significantly lighter. It turns out that the initial observation that the font is “very faint” is not a general property of the CMR fonts but a property

Modern printing equipment based on raster has been replaced by purely combinatorial patterns ify the desired position of ink in a discrete way—nputer science increasingly relevant to printing. We n completely precise definition of letter shapes that walent results on all raster-based in terms of variable parameters: seconds, making it possible for were previously unthinkable.


Fig. 10: METAFONT with *blacker* = 0.6, laptop

Modern printing equipment based on raster has been replaced by purely combinatorial patterns ify the desired position of ink in a discrete way—nputer science increasingly relevant to printing. We n completely precise definition of letter shapes that walent results on all raster-based in terms of variable parameters: seconds, making it possible for were previously unthinkable.


Fig. 11: METAFONT with *blacker* = 1.6, smart-phone

Modern printing equipment based on raster has been replaced by purely combinatorial patterns ify the desired position of ink in a discrete way—nputer science increasingly relevant to printing. We n completely precise definition of letter shapes that walent results on all raster-based in terms of variable parameters: seconds, making it possible for were previously unthinkable.


Fig. 12: METAFONT with *blacker* = 2.4, ebook

of a specific font implementation on a specific output device. Another popular choice are the OpenType Latin Modern (LM) fonts [1]. The native rendering on the Kindle Paperwhite (Figure 5) is comparable to that of the CMU fonts. It seems that the eInk technology used on the Kindle Paperwhite just needs heavier fonts.

When considering reading text on an electronic device, two other choices come to mind: laptop computers and smart phones (or tablets), which typically have a smaller screen but higher resolution. Figures 6 and 7 show the reference text as displayed on my smart-phone (Motorola Moto G, 1280x720 pixels, 329dpi) and my laptop (Dell Latitude E6530, 1920x1080 pixels, 142dpi).

It is clearly visible that, due to high resolution and good contrast, the font rendering on the smart-phone already approaches the rendering on traditional paper, whereas the laptop screen falls short of

our expectations. METAFONT was designed to produce good looking fonts at low resolution. Donald Knuth writes: “However, it will always be less expensive to work with devices of lower resolution, and we want the output of METAFONT to look as good as possible on the machines that we can afford to buy.” [2, page 195] Of course, we can hope that the ever-increasing resolution of our computer screens will make those techniques dispensable within the next years. But for the time being and for affordable, low-cost devices, good font rendering will continue to be an issue.

METAFONT and bitmapped fonts

METAFONT is aware of rasterization and takes great care to round the outlines of the glyphs to the available raster, but it assumes an output device that places small dots of black ink on white paper. In contrast, my ebook is able to produce 16 gray levels and my smart-phone screen is, at least in theory, capable of 256 shades of gray. (Other font rendering engines use even more sophisticated subpixel rendering.) To overcome this limitation of bitmap fonts generated by METAFONT, one can render the bitmaps for a higher resolution and then scale down the result to a lower resolution, converting partially-black regions to gray pixels. The effect of this mechanism can be seen in figure 8 and figure 9. Clearly the downscaling gives superior results. So the following figures all show fonts that are scaled down by a factor of 4.

The METAFONT system for font design offers special parameters to adapt the generated bitmap fonts for any specific output device [2, Chapter 24, Discreteness and Discretion]. The main parameter, of course, is the resolution. Since we are dealing with fonts that are too light, we turn our attention to the parameter *blacker*. The variable *blacker* is a special correction intended to help adapt a font to the idiosyncrasies of the current output device [2, page 93]. Its effect can be seen when comparing figure 9 to figure 10, where the parameter *blacker* has been chosen so that the visual appearance of the font on screen would match as closely as possible the appearance in the printed book (Figure 3). Similar results can be obtained for the smart-phone (Figure 11) and the ebook (Figure 12) with appropriately chosen values of *blacker*. The illustrations show that with appropriate parameters, the glyphs as rendered by METAFONT look better than their counterparts produced by the built-in font rendering engines from standard outline fonts optimized for high-resolution printers.

Conclusion

Preparation of an ebook from a TeX source will always be more than just flipping a switch in the TeX file. Just as preparing a book for print is more than just adopting the publisher’s style file: it might require for example stretching a paragraph by rewriting it to get a good page break; repositioning and redesigning illustrations, so that they fall on the right page and fit the available space on the page. These things are no longer necessary nor possible with ebooks, but other problems appear: Now the author has to judge the appearance of tables or program listings at different sizes and optimize font sizes for good readability at various magnification levels. Still, I expect that the algorithms of TeX can help us to produce ebooks of much better quality than the ebooks we have today.

But even if I can get TeX to produce a beautiful page layout for the ebook reader, I still need—especially for the traditional look and feel of books like “The Art of Computer Programming”—a TrueType or OpenType version of the Computer Modern Roman font family using Unicode encoding that is specifically designed for ebooks (or other on-screen reading). My experiments indicate that such fonts are possible and I sincerely hope that one of the many font specialists takes on this project. If you do, please let me know!

References

- [1] Boguslaw Jackowski and Janusz M. Nowacki. The Latin Modern (LM) Family of Fonts. <http://www.gust.org.pl/projects/e-foundry/latin-modern/>, 2009.
- [2] Donald E. Knuth. *Computers & Typesetting, The METAFONTbook*. Addison-Wesley, 1986.
- [3] Donald E. Knuth. *The Art of Computer Programming*. Addison Wesley, 1998.
- [4] Andrey V. Panov. Computer Modern Unicode Fonts. <http://canopus.iacp.dvo.ru/~panov/cm-unicode/>, 2010.
- [5] Martin Ruckert. *The MMIX Supplement: Supplement to The Art of Computer Programming Volumes 1, 2, 3 by Donald E. Knuth*. Addison-Wesley, 2015.
- [6] Christian Schenk. MiKTeX. <http://www.miktex.org/>, 2016.

◊ Martin Ruckert
Hochschule München
Lothstrasse 64
80336 München Germany
ruckert (at) cs dot hm dot edu

When (image) size matters

Peter Willadt

Abstract

For space and performance reasons, scaling of images to be included into a PDF document down to a certain resolution is often desirable. This article describes a halfway automatic method to achieve this goal with pdfTeX.

1 Basics

TeX output is by default device independent, and this is fine. In ancient implementations, adoption to different previewers or printing devices resulted from the work of DVI processing software, in recent years TeX has been widely replaced by pdfTeX and other software that produces PDF output which can be processed by a wide range of devices (the *P* in PDF stands for *portable*, after all).

With scalable fonts and scalable inline graphics produced by software packages like TikZ or METAPOST, still everything is fine. Problems arise when raster graphics are embedded into a PDF file. pdfTeX includes raster graphics at their natural size. Especially with the megapixel mania of digital cameras this leads to bloated files. Download time and processing effort at the printing device increase; perhaps your printer will give up with an out-of-memory error for rendering a stamp-sized photograph.

Rescaling bitmap images to a size that suffices for usual post-processing help keep file size and processing complexity small while retaining expected image quality. The most important question is: what resolution will suffice? There are two answers: With pure black-and-white pictures ('line art'), the printing devices' native resolution (e.g. 1200 dpi) is fine. With grayscale or color pictures, a resolution of 1/4 of that will easily do. The reason is that colored 'pixels' are formed by combining several dots.¹ For 16 distinct gray tones, one 'pixel' consists in theory of a 4×4 matrix of black or white pixels. In practice, the real resolution may be even coarser, as effects like bleeding of ink or surface roughness of paper have also to be considered. With professional printing equipment, true output resolution is measured in lpi (lines per inch) and grayscale or color images scaled to their dpi equal to the printing devices' lpi should be fine. If you read this article in the print version of *TUGboat*, figure 1 gives you the chance to see what

¹ It does not matter if the device does halftoning by rastering or dithering. Special printing devices which are capable of producing ink drops of varying size or using thermal sublimation are not covered by these thoughts.

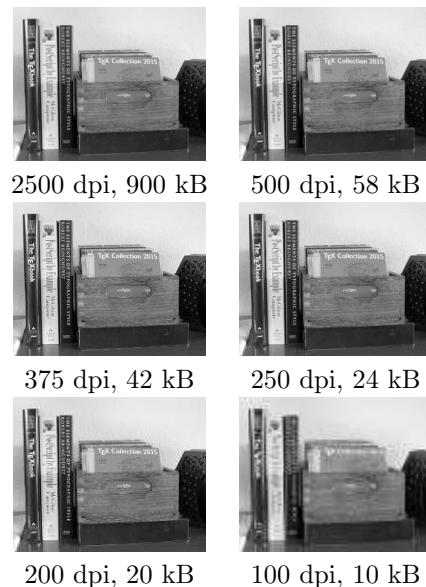


Figure 1: The same photograph in several resolutions. See for yourself where quality degradation becomes perceptible.

professional printing equipment can achieve on plain paper. Printing this page on your own device will probably be even more sobering: On my 1200 dpi laser printer, I can't see any difference among the images. With special paper and techniques like duotone printing, there is room at the top.

For online viewing, one pixel of an image corresponds to one pixel on screen. If you zoom in even more pixels may be needed. Considering high-res devices and moderate zooming, 300 dpi will probably suffice for the next few years.

2 Looking outside the box

In the early times of desktop publishing, when disk space was costly and memory size and local network bandwidth were seriously limited, layout designers often worked with low-resolution preview pictures and final images would be inserted on the way to the imagesetter. There even existed a standard called *open prepress interface* (OPI) [1] for automatic image replacement in PostScript files.

Adobe software (at least InDesign and Distiller) will rescale images during PDF production to an appropriate resolution chosen for the intended target you choose (e.g. print or web). OpenOffice and LibreOffice leave images untouched, while Microsoft Word treats pictures, without any chance to intervene, in a way that causes considerable grief to people in printing offices.

3 Including pictures for different output devices into a single PDF file

Probably for purposes similar to OPI, the PDF specification allows the inclusion of different images for viewing on screen and printing. Unfortunately, there are several restrictions and drawbacks:

- both images have to have the same dimensions
- both images should have the same resolution
- both images will be included within the PDF file, so file size gets bloated.
- Software support is rare.

Alternative print images are enabled as an experimental feature in `pdftex.def` and can be used with the `graphicx` package out of the box. You just say

```
\includegraphics
[print=imgPrint.jpg,...]{imgView}
```

instead of

```
\includegraphics[...]{myimgView}
```

and you're done.

It only works with bitmap graphics and you have to specify the full filename. Another drawback is that image reuse does not work with this option for screen images, so your file gets bloated even more. With Adobe Acrobat, I have been able to use it, but most other PDF processing software fails. On my GNU/Linux system, I ended up with a file I could view but not print. Considering all this, I can unfortunately see no good use for this technology apart from playing pranks.

4 Ways of attack

There are three possible hooks to scale images: before the pdftEX run; while pdftEX is processing pictures; and as a postprocessor on the finished PDF.

Googling for the problems aforementioned, you will find a postprocessing solution using Ghostscript on the final PDF file [2] and another one using a Python tool [3].

There also exists a L^AT_EX package and corresponding ConTeXt module, both called `degrade` [4], which shrink image files on the fly using ImageMagick in the background. Both of these packages require a Unix-ish operating system and `\write18` has to be enabled.

Beyond downscaling, there are other ways of getting smaller image files. For one, increasing JPEG compression allows drastic reductions in space. This can be done when there will be no further image processing involved, but it requires careful visual checking for compression artifacts.

Also, color depth can be decreased (by “posteriorization” or grayscale conversion). The author believes

that this technique is best carried out with interactive software and visual checks for the results. Unfortunately, reducing color depth does not yield large gains in space,² but it might be useful to do gray-scale conversion for material to be printed in black and white to get fine control over the results. Gamma correction and adjustment of black and white levels are often helpful to get better printing results, but for file size there is no benefit.

5 Proposal for a perfect solution

A presumably perfect solution would scale pictures on the fly while producing PDF, ideally triggered by a command like `\pdfFinalResolution=300` or `\destination=web` in the document preamble. This command would probably be supported by some bookkeeping to avoid unnecessary computations on already downscaled images. If black-and-white output was intended, all images might be converted to grayscale, also keyword-driven. Of course, when images were to be clipped, only the visible part of these images would be included.

I guess that this could be done with LuatEX almost out of the box, and as it has now (Sept. 2016) reached a stable state, there should be no obstacles to implementing it.

6 Implementation (less than perfect) and usage

I have resorted to external software that reads a T_EX log file to scan for filenames of images and required target resolutions and then builds up scaled images. As you can specify paths for graphic inclusion with the `graphicx` L^AT_EX package, you get a comparatively easy solution if you adopt to some conventions. You should avoid giving path names on individual `\includegraphics` commands and instead use the `\graphicspath` directive. In a first run you will comment the path to the final images out, having generated the downscaled pictures you will comment the original file path out.

```
\graphicspath{{my/hires/images/}}
%\graphicspath{{printimg/}}
% move comment up for final pdfLaTeX run
```

You will have to repeat this procedure as you change image sizes or as you add new images, so it is probably best to start generating scaled pictures when your document is almost done. Really fast previewing can—as you probably know—be done by specifying the `draft` option to the `graphics` package,

² With the example picture, only ten percent reduction of disk space was achieved by grayscale conversion.

where you get only frames instead of pictures in your PDF file.

So, your workflow will look like this:

- Run `LATEX` on your file, with `\graphicspath` pointing to the original files.
- Run `pdflatexpicscale` on your `LATEX` project.
- Run `LATEX` on your file, with `\graphicspath` pointing to your optimized files.
- Repeat if you change picture sizes, add new pictures, or choose a different target resolution.

If you cannot produce PDF files directly, the only change to the workflow will be that you have to additionally call your PDF producing software.

`pdflatexpicscale`³ is a Perl script. It depends on some standard Perl packages and the presence of ImageMagick software. As these prerequisites are quite common, it should run with your system. You call the script with the name of your `LATEX` project and optionally with the desired resolution and desired picture directory. If you omit arguments, reasonable defaults will be assumed. So a typical call would be:

```
pdflatexpicscale --printdpi=200 \
--destdir=medrespics myarticle
```

If your `LATEX` file is called `myarticle.tex`, you have done a `LATEX` run, so that the log file exists, you want 200 dpi output and the directory for the scaled pictures is an existing subdirectory of the current directory called `medrespics`, then you may copy the above command verbatim.

The software can be downloaded from CTAN [5], and is included in `TeX Live`. Documentation is included. You may probably want to read it, as it is not identical with this article.

7 Caveats, limitations and drawbacks

My PostScript printer prints some black-and-white images inverted. I could have inverted them with an image processor, but then they would look wrong on screen. As a workaround I converted them to grayscale. Some provision needs to be made to keep `pdflatexpicscale` from scaling them down like other halftone images. The easiest way is keep them in a separate directory and to include this directory at the beginning of the `\graphicspath` list.

The target resolution you choose may not truly meet the printing devices' needs, especially if you do not know who will print your document. Perhaps the printing device has got fantastic image scaling software that you replace by some inferior software on your computer. Also you probably will not want

³ This name was chosen because Google found no hits in July, 2016.

to recompile all of your documents just because you bought a new printer.

`pdflatexpicscale` changes image size and target resolution. This has serious consequences if you intend to use clipping, or to display pictures in a size dependent on their resolution. Also, anisotropic scaling is not supported.

When a file gets used several times at different sizes, only the largest will be included. The software reads the log file from beginning to the end and starts rendering immediately, so when an image is included at first in thumbnail size and then larger, it will be rendered several times.

The Perl script uses ImageMagick's `convert` software for scaling pictures, so quality of resampling and file compression (most important for lossy compression formats like JPEG) depend upon ImageMagick's algorithms.

Security concerns: ImageMagick has had several security flaws fixed in 2016. So it is probably not a good idea to provide scaling services to anonymous users that might upload a malicious image file.

The solution presented only deals with pure raster graphics (JPEG and PNG). If you include graphics in a mixed format like PDF, rasterization might be beneficial or disadvantageous, depending on the content. Rasterizing vector graphics is definitely not what you want. Treating your PDF with one of the postprocessing solutions mentioned might help.

A last remark: Having two projects share the same images is a recipe for dissatisfaction. It is quite common to keep, for instance, a presentation and the corresponding handout in the same folder, but graphic requirements are totally different. The best solution is to keep downscaled pictures in separate directories; `pdflatexpicscale` can easily cope with this.

References

- [1] http://wwwimages.adobe.com/www.adobe.com/content/dam/Adobe/en/devnet/postscript/pdfs/5660.OPI_2.0.pdf
- [2] <http://tex.stackexchange.com/questions/14429/pdftex-reduce-pdf-size-reduce-image-quality>
- [3] <http://tex.stackexchange.com/questions/2198/how-to-create-small-pdf-files-for-the-internet>
- [4] <http://ctan.org/pkg/degrade>
- [5] <http://ctan.org/pkg/pdflatexpicscale>

◇ Peter Willadt
willadt (at) t-online dot de

A survey of the history of musical notation

Werner Lemberg

Abstract

Music has been and still is an essential part of life. Similar to writing text there have been various ideas on how to notate music. This article tries to show, with many images, the solutions found in the course of more than 3000 years of history.

1 Introduction

Over the millennia, humanity has developed many different ways to notate music. The solutions can be roughly categorized visually as follows.

1. action description
2. words
3. letters
4. digits
5. graphics
6. stylized graphics
7. abstract symbols
8. combination of 1–7

This ordering also roughly corresponds to historical development across cultures: The oldest Chinese sources we know of are action descriptions, Mesopotamia seems to have started with letters and digits, while modern Western notation essentially uses all possible combinations.

Interestingly, preserving music itself has been much less important than preserving words – this is true for all ancient cultures throughout the world. We know many texts of hymns and songs, but their music did not survive. Another observation is that most cultures only developed *tablatures*. A tablature basically notates the fingering to play the music on an instrument, not the music itself. An exception to that was a notation system for songs in ancient Greek; however, it was lost with the fall of the Roman empire. Perhaps due to the century-long ban of instruments in the church music of early Christianity, musicians in western Europe developed new ways to notate sung music, which eventually led to the modern notation which is used all over the world today.

The emphasis in this survey is on the graphical representation of music, showing both the inventiveness and the beauty of the solutions discovered, via many images. Using some technical terms related to music for the descriptions is unavoidable; readers without a musical background, however, can simply skip them and enjoy the pictures for themselves.

This article is a greatly revised and extended version of a paper submitted to the MOTYF 2014 conference.



<https://www.youtube.com/watch?v=270pcKxcg1c>, with permission

Figure 1: A kinnor-like lyre.

ence proceedings.¹ Most images shown here are high-resolution scans; it is thus recommended that you have a look at the online PDF version so that you can zoom into the document for details!

The music examples were typeset with GNU LilyPond version 2.19.43.²

2 Mesopotamia – Hurrian songs

The oldest notation for music we currently know of was found in the Royal Palace at Ugarit, an ancient port city in northern Syria, today called Ras Shamra (رأس شمرا). These clay tablet shards date to approximately 1400 BCE (figs. 2 and 3); the notation uses words and digits (in Akkadian cuneiform) for a nine-string lyre (fig. 1).

There are several problems making it hard to interpret the data in a meaningful way.

- The text is written in a poorly understood Hurrian dialect.
- Words in the notation represent intervals, not pitches (fig. 4) – is this polyphonic? If not, what is the order of the pitches? Ascending? Descending?
- Should the intervals be filled with scales or something different? In other words, is this a one-to-one representation of what should be played, or is it just a mnemonic aid?
- What do the numbers mean? Repetition? Beats? Something completely different?

3 Greece – The Seikilos Column

The example shown in this section is the famous Seikilos column, found in one of the largest Aegean cities in antiquity, Tralleis (Τραλλεῖς, today Aydin, Anatolia, Turkey), with an approximate age of 2000 years (figs. 7, 5, 6). It is a tombstone depicting an epigram or a skolian (drinking song).

1. MOTYF. International Students' Moving Type Festival 2014: *Type in Music, the Rhythm of Letters*. Polsko-Japońska Akademia Technik Komputerowych, Warsaw 2015. ISBN 978-83-63103-76-7.

2. <http://www.lilypond.org>

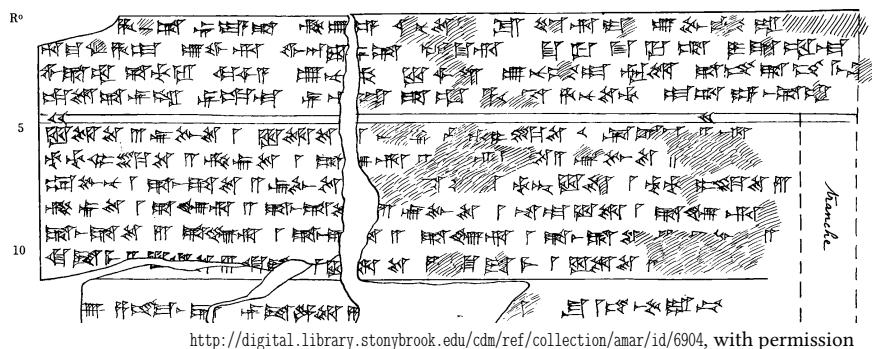


http://www.ramivitale.com/wp-content/uploads/2013/09/H6TabletsFront_lg.jpg



http://www.ramivitale.com/wp-content/uploads/2013/09/H6TabletsBack_lg.jpg

Figure 2: Hymn tablet h. 6 (front and back), consisting of shards RS 15.30, 15.49, 17.387 (Natl. Museum of Damascus).



R^o

5	qáb-li-te 3 ir-bu-te 1 qáb-li-te 3 ša-ah-ri 1 i-šar-te 10 uš-ta-ma-a-ri
6	ti-ti-mi-šar-te 2 zi-ir-te 1 ša-ah-ri 2 ša-aš-ša-te 2 ir-bu-te 2
7	um-bu-be 1 ša-aš-ša-te 2 ir-bu-te 1 [+X] na-ad-qáb-li 1 ti-tar-qáb-li 1 ti-ti-mi-šar-te 4
8	zi-ir-te 1 ša-ah-ri 2 ša-aš-ša-te 4 ir-bu-te 1 na-ad-qáb-li 1 ša-ah-ri 1
9	ša-aš-ša-te 4 ša-ah-ri 1 ša-aš-ša-te 2 ša-ah-ri 1 ša-aš-ša-te 2 ir-bu-te 2
10	ki-it-me 2 qáb-li-te 3 ki-it-me 1 qáb-li-te 4 ki-it-me 1 qáb-li-te 2

Figure 3: A transcription of the cuneiform text below the double line that represents musical notation, following Manfred Dietrich and Oswald Loretz (*Kollationen zum Musiktext aus Ugarit*, Ugarit-Forschungen 7, 1975).



Figure 4: The intervals and counters as used in the Hymn tablet. Depending on the order of strings (either ascending or descending), which is unknown, either the first or the second line is the correct one.



<https://commons.wikimedia.org/wiki/File:Seikilos2.tif>

ΕΙΚΩΝΗ ΛΙΘΟΣ
ΕΙΜΙ·ΤΙΘΗΣΙ ΜΕ
ΣΕΙΚΙΛΟΣ ΕΝΘΑ
ΜΝΗΜΗΣ ΑΘΑΝΑΤΟΥ
ΣΗΜΑ ΠΟΛΥ ΧΡΟΝΙΟΝ
ΟΣΟΝ ΖΗΣ ΦΑΙΝΟΥ
ΜΗΔΕΝ ΟΛΩΣ ΣΥ
ΛΥΠΟΥ· ΠΡΟΣ ΟΛΙ-
ΓΟΝ ΕΣΤΙ ΤΟ ΖΗΝ.
ΤΟ ΤΕΛΟΣ Ο ΧΡΟ-
ΝΟΣ ΑΠΑΙΤΕΙ.

Figure 5: The text of the Seikilos column; left a flattened image, right a version with modernized orthography. Note the final ‘T’ character: To get more symbols for music notation, Greek characters were both mirrored and rotated.

Είκωνή λίθος είμι.
Τίθησι με Σεικίλος ἐνθα
μνήμης ἀθανάτου
σῆμα πολυ χρόνιον.

I am a tombstone, an image.
Seikilos placed me here
as an everlasting sign
of deathless remembrance.

Οσον ζῆς φαίνου
μηδὲν ὄλως σὺ λυποῦ·
πρὸς ὄλιγον ἔστι τὸ ζῆν.
τὸ τέλος ὁ χρόνος ἀπαιτεῖ.

While you live, shine
have no grief at all
life exists only for a short while
and time demands its toll.



Hoson zēs, phai - nou mēden ho-lōs sy ly - pou pros ol-igone s - ti to zēn to te-los ho chronos a-pai-tei
"Οσον ζῆς, φαί - νου μη-δὲν ὄ-λως σὺ λυ - ποῦ πρὸς ὄλ-ιγον ἔσ - τι τὸ ζῆν τὸ τέλος ὁ χρό-νος ἀπαι-τεῖ

Figure 6: A text version of the Seikilos column using mixed-case Greek, together with an English translation and a representation with modern musical notation. A bar over letters represents two beats, a hooked bar three beats. A dot indicates an unstressed beat, a tie below letters marks a syllable to be sung with more than a single note.

In contrast to the Hurrian songs, this is the earliest known piece of music that can be *almost exactly* transcribed to today’s music notation, thanks to many scientific works of Pythagoras and others who introduced music notation for their theoretical treatises. Sadly, the number of music pieces that actually use the notation is

very small; as mentioned in the introduction, it was not considered important to be written down. Additionally, knowledge of this notation system was lost in the early middle ages; only the text of Greek songs has survived.



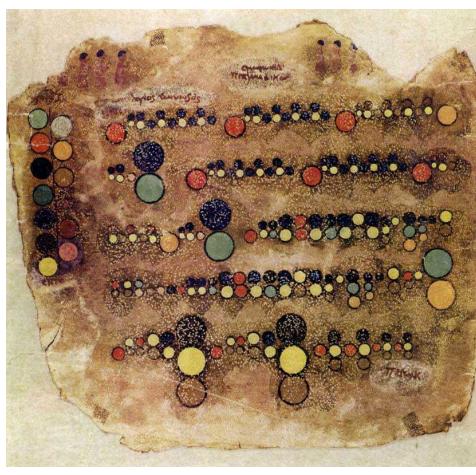
https://commons.wikimedia.org/wiki/File:2._Stèle_portant_l'inscription_de_Seikilos.jpg

Figure 7: The Seikilos column, National Museum of Denmark.

4 Egypt

No musical notation is known from the culture of ancient Egypt; apparently, there was only an oral tradition. Musical instruments are displayed in many images (and some have even survived); it is thus possible to reconstruct at least the range of possible sounds, but nothing more.

A Coptic document with coloured circles, dated to the 5th–7th centuries CE, might be related to notation (fig. 8); however, nobody really knows.



http://musicofthebiblerevealed.files.wordpress.com/2013/07/coptic_musical_notation.jpg

Figure 8: One of six Coptic parchments; colours might indicate pitch, size the duration. Metropolitan Museum, New York (?).

5 Far East

5.1 China

Similar to ancient Greece, music theory was highly developed in ancient China. The most important archaeological site, from a musicological point of view, is the tomb of Marquis Yi of Zeng (曾侯乙墓, Zēng hóu Yǐ mù, located in Léigǔdūn, 擂鼓墩, Hubei province, China), dated sometime after 433 BCE. Excavated chimestones and bells contain inscriptions related to pitches, scales, and transposition (fig. 9). However, no musical notation was found.

The oldest known notation from China dates from the 7th century, called wénzìpǔ (文字譜), a longhand tablature (figs. 10 and 11). It is a plain text description of how to play the gǔqín (古琴), a zither.

During the Tang dynasty (8th to 9th centuries) this system of verbal descriptions was greatly simplified, leading to the jiānzìpǔ (減字譜) tablature (fig. 12). In parallel, another tablature called gōngchěpǔ (工尺譜) was invented (fig. 13); both systems use Chinese characters, digits, and other symbols to notate fingerings.

Modern non-western notation (簡譜 jiǎnpǔ), introduced in the early 20th century, is most likely based on the French Galin-Paris-Chevé system, published 1818 (figs. 14 and 15 [after main text]). While not having



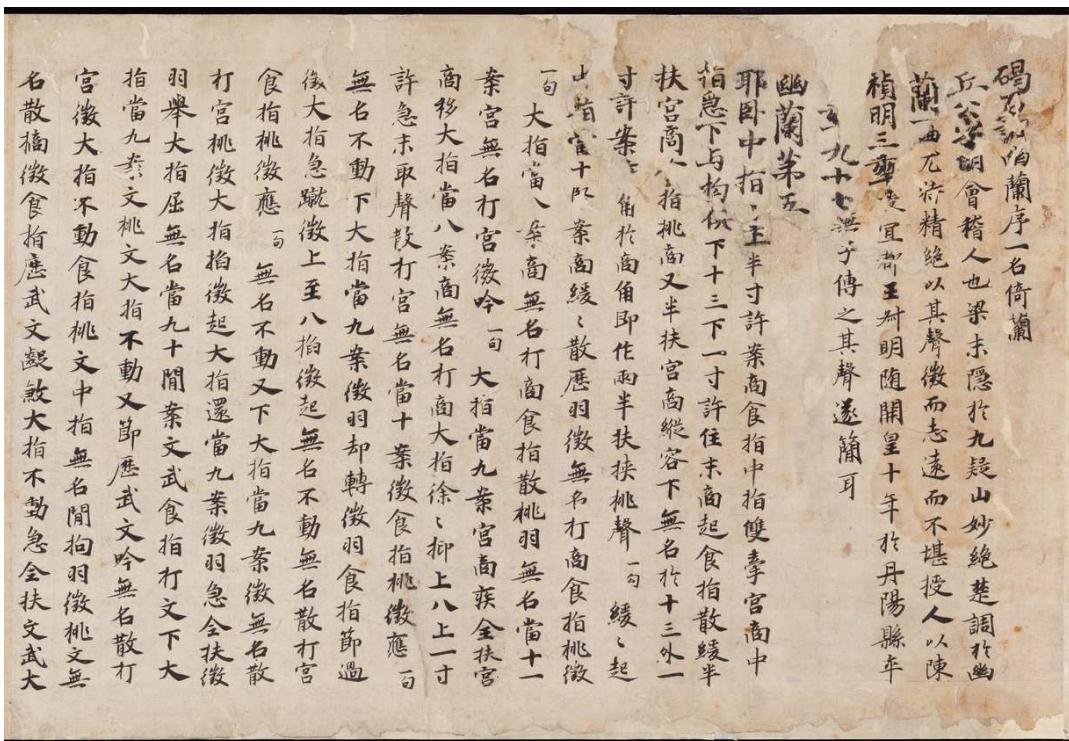
<http://herschelian.files.wordpress.com/2013/09/bianzhong-concert-2.jpg>, with permission

Figure 9: Bells from the tomb of Marquis Yi of Zeng, Hubei Provincial Museum, Wuhan.

耶臥中指 | 卅半寸許案商,
食指、中指雙牽宮商。
中指急下与拘俱下十三下一寸許。
住。末商起。食指散緩半扶宮商。
食指挑商。又半扶宮商。
縱容下無名於十三外一寸許案商角,
於商角即作兩半扶、挾挑聲。



Figure 10: The beginning of *Jiéshí diào yōu lán*, with John Thompson's transcription. While the pitches in the original are given to the quarter tone or better, one can only approximate the rhythm.



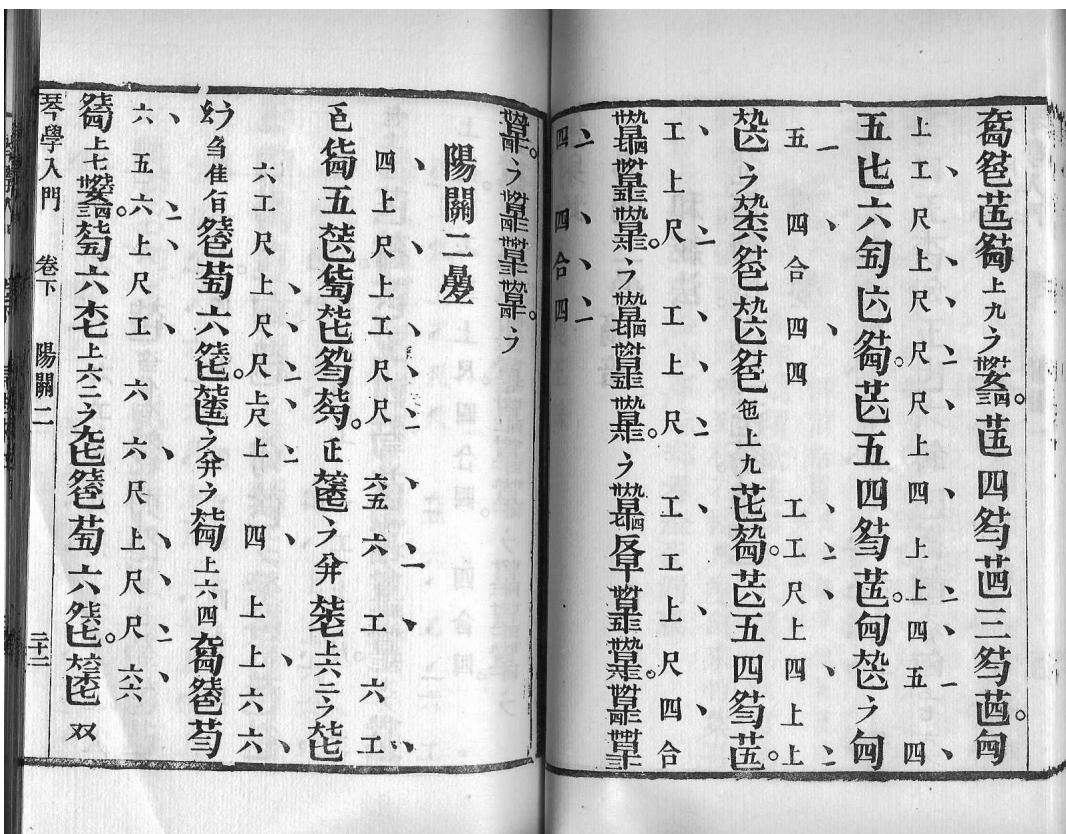
<http://www.emuseum.jp/detail/100229/000/000>

Figure 11: The beginning of the 4 m long scroll with wénzǐpǔ tablature of the piece *Jiéshí diào yōu lán* (碣石調幽蘭) “Secluded Orchid, in Stone Tablet Mode”, from the 7th century (Tōkyō National Museum, TB-1393).



https://en.wikipedia.org/wiki/File:Shenqi_Mipu_vol_3_pg_1.jpg

Figure 12: Two pages from *Shéngqí mǐpǔ* (神奇秘譜), dated 1425, an example of jiānzǐpǔ tablature for the qín zither.



https://en.wikipedia.org/wiki/File:Kam_Hok_Yap_Mun-Yeung_Kwan_Sam_Tip.jpg

Figure 13: Two pages from the score book *Qín xué rùmén* (琴學入門), dated 1864, showing gōngchěpǔ tablature for the gǔqín zither.

This figure shows a page from a German Catholic hymnal. The page number is 105. The title of the hymn is 'O heil'ge Seelenspeise'. The notation consists of a series of cipher-like numbers (1, 3, 1, 2, etc.) and corresponding lyrics in German. The lyrics are:

1 = F.
N° 217. || 3 1 2 3 5 4· 3 |
O heil'-ge See=le=n =spei =se
3 5 5 2 3 1 7 | 7 1 2 3
au die=fer Pil=ger rei=fe, o Man-na, Himm=
3 2 | 3 1 2 3 5 4· 3 |
mel=brot! Du la=best hier die Mü=den
3 5 5 2 3 1 7 | 5 1 2 3
mit Got=tes fü=hem Frie=den und stär=kest uns
4 3 2 1. ||
zum sel=gen Tod.

<http://sammlungen.ulb.uni-muenster.de/hd/content/pageview/1474276>

Figure 14: Cipher notation for the song *O heil'ge Seelenspeise*, contained in the book *Sursum corda*, a German Catholic hymnal from 1887. The displayed melody is based on *Innsbruck, ich muß dich lassen* (*Innsbruck, I Must Leave You*), a famous song by Heinrich Isaac composed in the second half of the 15th century.

a significant impact on the Western world, it became extremely popular in Asia since it is comparable to the gōngchē tablature. Even today, most traditional music scores and song books use jiānpǔ notation.

5.2 Japan

All traditional notation systems in Japan are tablatures, strongly influenced by China and Korea, which in turn was also influenced by China (fig. 16). In the course of time, many different, specialized notations were developed depending on the instrument. This was further specialized by competing music schools, trying hard to provide knowledge of playing the instrument only to members of the clan (fig. 17). In spite of the specialization, all notation systems are just mnemonic devices, making it impossible to interpret it correctly without additional oral tradition.

5.3 Korea

Similar to Japan, both music and music notation was strongly influenced from China.

In the 15th century, the jeongganbo mensural notation (정간보, 井間譜) was developed, providing a means to exactly specify the rhythm by positioning the musical information into a grid. This system, which was the first in Asia able to represent duration of notes, is still in use today (fig. 18); the idea of using a rhythm grid was also exported to Japan in the 18th century.

6 India

Ancient India is another major civilisation that did not develop explicit notation systems. Instead, only hints, usually small strokes above and below the text, were added. As is to be expected, such a system is not reproducible without the oral tradition from guru (गुरु, teacher) to shishya (शिष्य, student).

Modern notations were developed by Vishnu Narayan Bhatkhande (विष्णु नारायण भातखडे, 1860–1936) and Vishnu Digambar Paluskar (विष्णु दिगंबर पलुस्कर, 1872–1931) in northern India, mainly for teaching music and the preservation of traditional compositions. They are based on Devanagari characters and numbers with a small set of additional symbols (fig. 19).

7 Middle East

No music notation systems were developed in the Middle East after the fall of the Roman empire; there was only oral tradition, as far as we know. Starting around 1830 in Egypt, Western notation was introduced, but only in a very limited way.

أبو يعقوب بن إسحاق (Al-Kindi, ابو نصر محمد الفارابي, 9th century) and Al-Farabi (الفارابي, 10th century) used letters to denote strings of the oud (عود, an Arabic lute), together with finger positions. Safi

al-Din al-Urmawi (صنف الدين الارموي, 13th century) additionally used digits to indicate rhythm in his works. It must be noted, however, that none of these systems gained any practical importance for playing music.

The perhaps most remarkable contributor to middle eastern notation systems was Dimitrie Cantemir, Prince of Moldavia (Turkish: Kantemiroğlu), who published his letter notation around 1710 while in forced exile in Constantinople, collecting and preserving around 340 Ottoman instrumental pieces (figs. 20 and 21).

8 Europe

8.1 Neumes

Isidore of Seville, living in the early 7th century, states in his book *Etymologiae* (also known as *Origines*):

*nisi enim ab homine memoria teneantur soni,
pereunt, quia scribi non possunt³*

*(unless sounds are held by the memory of man,
they perish, because they cannot be written down)*

Music history soon provided counterexamples: Visigothic *neumes* (i.e., inflective marks to notate music) began to develop in northern Spain in the late 7th century (figs. 22 and 23).

Similarly, the first paleofrankish neumes appeared around 850 in Aurelian of Réôme's works (fig. 24).

In the 10th and 11th centuries, development and usage of neumes started to flourish in many places in Europe: St. Gallen (Switzerland), Laon, Brittany (France), to name just a few.

Neumes can be roughly classified as either *adiastematic* or *diastematic*. The older adiastematic neumes show the direction of a melody, but no pitches. On the other hand, rhythm and dynamics were quite precise (fig. 25).

Diastematic neumes were rather the opposite: Quite precise pitches, but lack of rhythm and dynamic hints (fig. 26).

It is probable that neumes were originally developed in the Byzantine Empire, based on Greek origins. The orthodox church still uses neumes today (with refined notation).

8.2 Staff lines

Another European invention was the use of staff lines. The first use of horizontal lines to indicate the pitch can be found in the theoretical work *Musica enchiriadis*, written in the 9th century (fig. 27).

Guido of Arezzo further developed the idea of staff lines; he recommended the use of lines in distances of a third in his book *Prologus in Antiphonarium* (around 1030), together with a clef (or coloured lines) to indicate pitches.

3. Isidorus Hispalensis, *Etymologiae*, book III, *De Musica*.

With the introduction of square-note neumes in the 12th century, the development of the notation of Gregorian chant was essentially completed (fig. 28), and is still in use today.

In the same century, polyphony started to develop, mainly in the Notre Dame school in Paris. It also introduced *modes* to control the rhythm (figs. 29 and 30).

8.3 Mensural notation

Perhaps the most important invention in Western musical notation was made by Franco of Cologne (around 1250, as documented in his book *Ars cantus mensurabilis*). He introduced note heads with different shapes to define their own duration.

Previously, rhythm was only implicitly defined by context and learned rules; the new class of note heads allowed the notation of arbitrary durations. This system is still basically the same as what we use today, with only minor changes and additions (figs. 31, 32, 33).

8.4 Tablatures

In the Western music, tablatures spread in the 15th century, mainly for instruments that can produce more than a single note at the same time. Either digits or letters were used to denote keys or fingering (figs. 34, 35, 36).

8.5 Printing

Printing music with movable type started around 1500. In the beginning, the layout was a copy of handwriting (figs. 37 and 38).

Bar lines, ties, slurs, and other marks were gradually introduced in the 16th and 17th centuries (fig. 39).

8.6 Engraving

In the 18th century, polyphonic music became too complicated to be printed with movable types. Instead, engraving techniques were introduced for music, starting with copper plates (chalcography, fig. 40).

Around 1730, the English music publisher John Walsh invented a new engraving technique: Staff lines were drawn with a 5-pronged ‘scoring tool’ onto a pewter plate (an alloy of mainly tin), fixed-size musical symbols were punched with dies, and everything else engraved manually (fig. 41).

In 1799, Johann André from Offenbach am Main, Germany, applied the newly invented lithography technique to music – images on zinc plates being mechanically transferred (fig. 42).

Around 1860, the aesthetics of classical music engraving as used today were completed (fig. 43).

9 The future

The last revolutionary step in the history of music notation to date is the introduction of computers to typeset

music – this is happening right now. Today, the job of a music typesetter working with pewter and dies is now essentially defunct.

However, until very recently, the results produced by computers were hardly adequate compared to manually engraved scores. This difficulty is mainly due to the two-dimensionality of the data, making it hard to automatically achieve visually pleasing scores. At the present time, this is going to change: Computers are steadily becoming more powerful, allowing for the mathematically expensive computations that are necessary for good automatic positioning of the notational elements.

Software is evolving, too: Programmers are learning from the errors and problems affecting the first-generation programs used for typesetting music, and also providing better GUIs with powerful templates for users – who thus need be less aware of the intricate details of correct music layout.

Sources

All images not tagged with a URL were created by the author. Here is a table with additional notes for selected images.

- 1 This image was extracted from a video; it shows Peter Pringle playing an ancient lyre.
- 3 On his website, Casey Goranson collects no less than ten different realizations of the tune that appeared in various scientific papers, see http://individual.utoronto.ca/seadogdriftwood/Hurrian_Website_article_on_Hurrian_Hymn_No._6.html. Most of them, if not all, are highly speculative due to lack of information.
- 6 The English translation of the Greek text was taken from Wikipedia, https://en.wikipedia.org/wiki/Seikilos_epitaph.
- 8 The blog entry <http://musicofthebiblerevealed.wordpress.com/2013/07/25/spiritual-harmony-coptic-musical-notation> gives more details on the possible interpretation of this parchment.
- 9 The photo belongs to Jo Michie’s blog on China, <http://herschelian.wordpress.com/2013/09/25/the-marquis-yi-of-zengs-musical-bells>.
- 10 Thompson’s complete transcription of the piece can be found at <http://www.silkqin.com/02qnpu/01yl/transpdf/jsdyl01.pdf>.
- 19 The image was taken from David Courtney’s site *Music of India*, http://chandrakantha.com/articles/inian_music/lippi.html.

◊ Werner Lemberg
wl@gnu.org

1=G

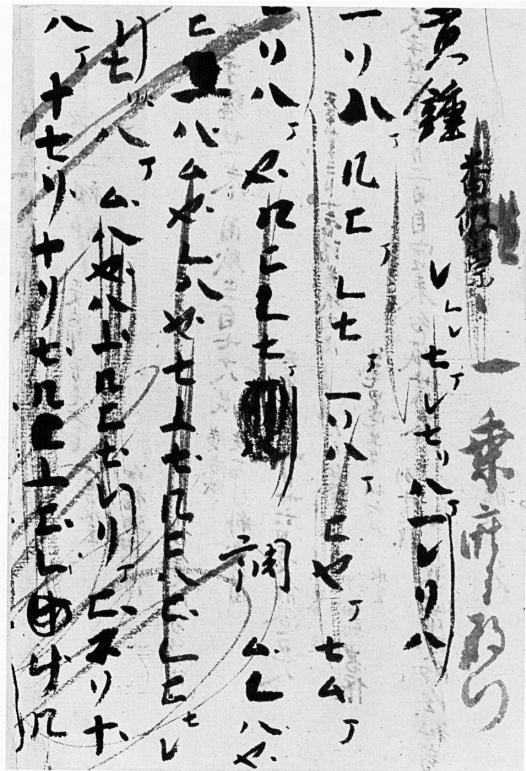
Top Part (Jiǎnpǔ Notation):

4/4 0. 6 5643 | 2 - 2. 3 1 12 | 3. 5 6 5 6561 |
 5. 3 5 5 3 2 6 5612 | tr 3. 5 2.351 6235 | 1 - i 61 3 32 |
 1. 6 i. 233 21. 1 6123 | 5 - 5035 6561 | tr 5. 3 551 6 6 5655 |

Bottom Part (Western Notation):

Three staves of Western musical notation in G major, 4/4 time, featuring eighth and sixteenth note patterns.

Figure 15: Beginning of the piece *The Moon Mirrored in the Second Spring* (二泉映月, Èrquán yìngyuè) for the èrhú (二胡, a Chinese fiddle); jiǎnpǔ and Western notation.



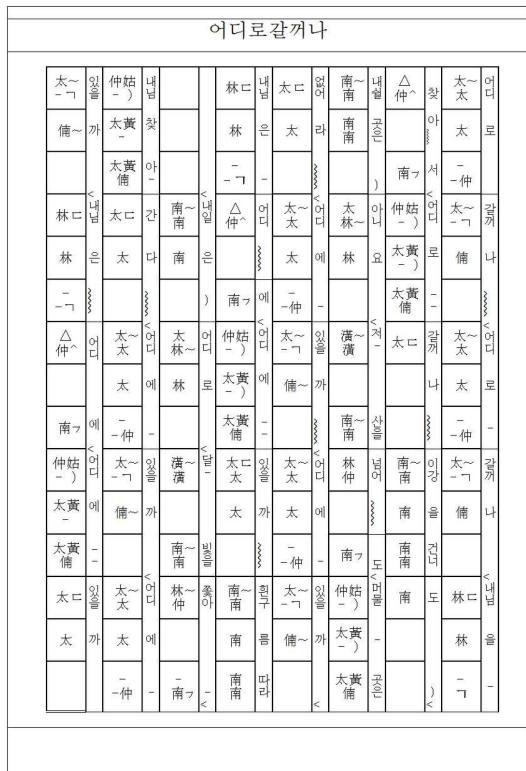
https://en.wikipedia.org/wiki/File:Tempyo_Biwa_Fu.jpg

Figure 16: The *Tempyo biwa fu* (天平琵琶譜, Tempyo lute score), dated ca. 738. This is essentially a Chinese piece using Chinese lute notation, preserved in the Imperial Storehouse (Shōsōin 正倉院) in Nara, Japan.



http://www.shaku-rus.com/Scores/Ch_score.jpg

Figure 17: An example of a tablature for the shakuhachi (尺八, an end-blown flute).



<http://cfile222.uf.daum.net/image/251EC74A5184633226C90A>

Figure 18: A modern example of jeongganbo notation, to be read from top to down, right to left. The square boxes contain the melody, the rectangles to the right hold playing instructions. In the square boxes, two rows split the beat into two half beats; two characters in a row further split a half beat into two quarter beats. A ‘-’ character indicates a rest.



http://www.musikwissenschaft.uni-wuerzburg.de/struktur/lehrstuhle_professuren_ressorts/projekte_und_materialien_des_lehrstuhls_fuer_ethnomusikologie/osman/elci_pesrevi_1_faks/

Figure 20: A page of Cantemir's treatise *Kitab-i 'Ilmü'l-Müsiķi 'alâ Vechi'l-Hurûfât* (Türkiyat Enstitüsü Library, İstanbul, Arel 2768).



http://chandrakantha.com/articles/indian_music/lippi_media/bhat_notation2.jpg, with permission

Figure 19: An example of two systems of Bhatkhande's notation, taken from *Hindustani Sangeet-Paddhati Kramik Pustak Malika* (हिन्दुस्तानी संगीत-पद्धति क्रमिक पुस्तक मालिका), Volume 4. Each system consists of four lines.



Figure 21: A zoom into the fourth line of the page showing the beginning of the song 'Irak Elçi Peşevi', *Usul Düyek*, with transcription. The lower part of the line in the facsimile with Arabic digits (to be read from right to left) gives the rhythm of the melody.



Figure 22: Two pages from the *Visigothic Antiphonal*, most probably an 11th century copy of a 7th century book (Archivo de la Catedral de León, Ms. 8). It depicts Mozarabic chant.

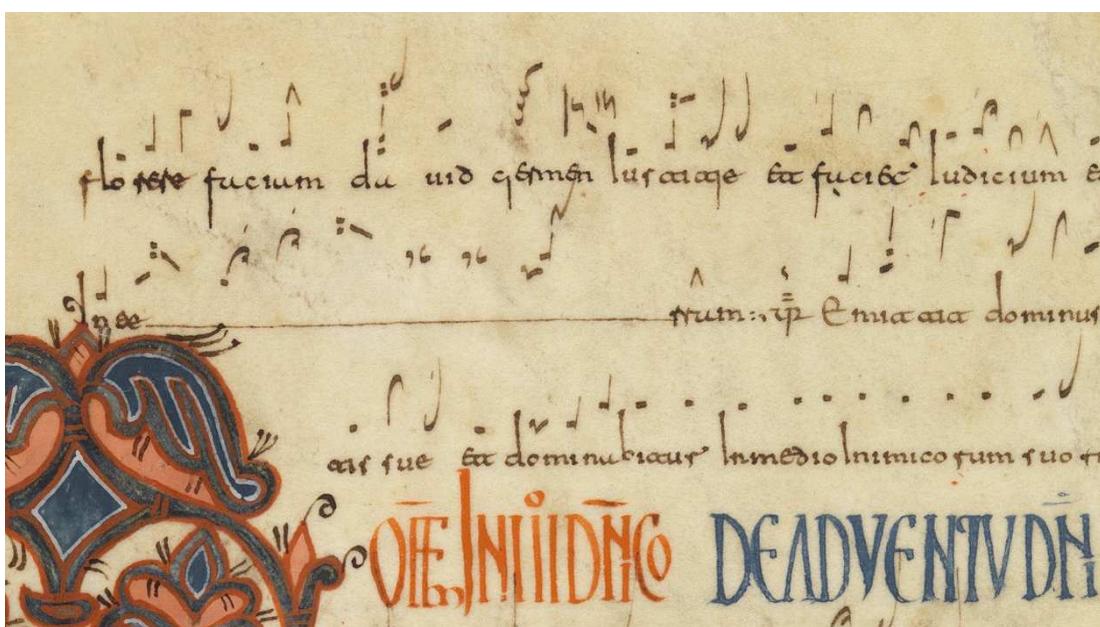
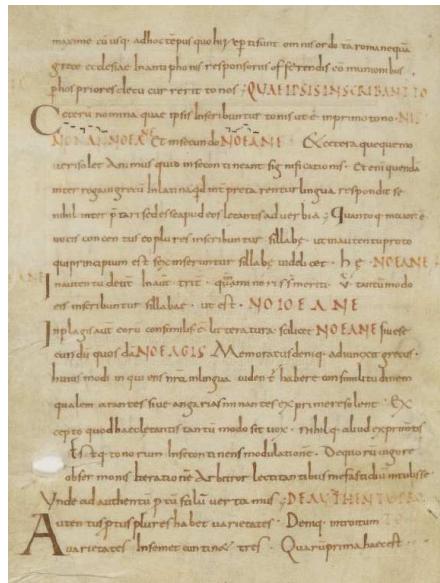
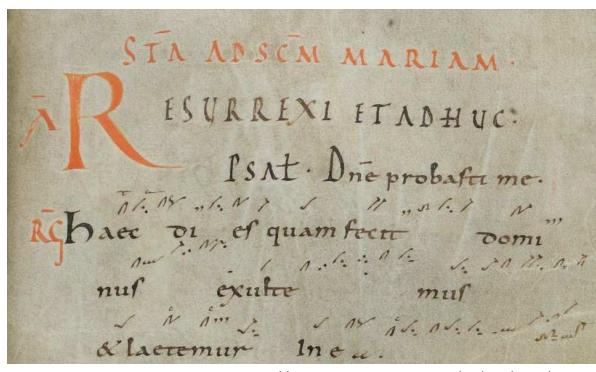


Figure 23: A detailed view of the Antiphonal, also called *Antifonario de León*. Today, these neumes are almost completely undecipherable – the Mozarabic Rite was forbidden in Spain around 1080 by Pope Gregory VII and replaced by the Roman Rite.



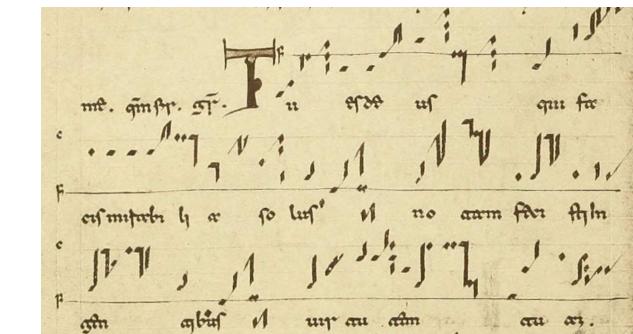
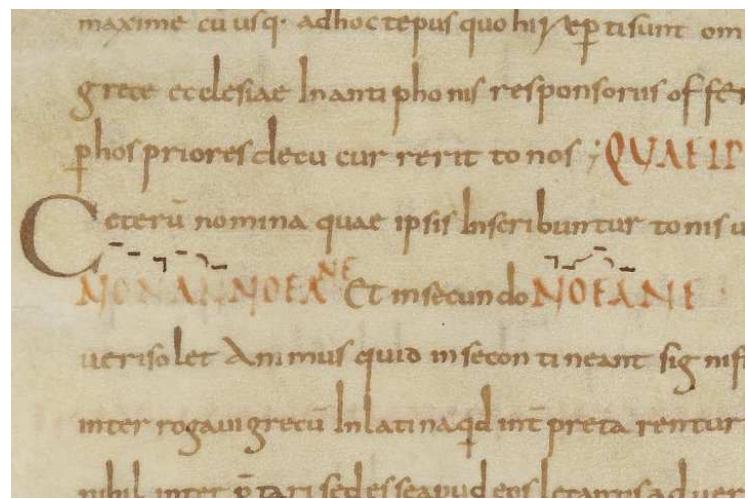
<http://gallica.bnf.fr/ark:/12148/btv1b8452635b/f147.item>

Figure 24: Two views of a page of Aurelian of Réôme's book *Musica disciplina*, a Carolingian chant treatise written around 850 – containing only text, no music. The neumes (undecipherable today) in this copy from ~880 were apparently added to this copy by an early reader (Bibliothèque municipale de Valenciennes, Ms. 148).



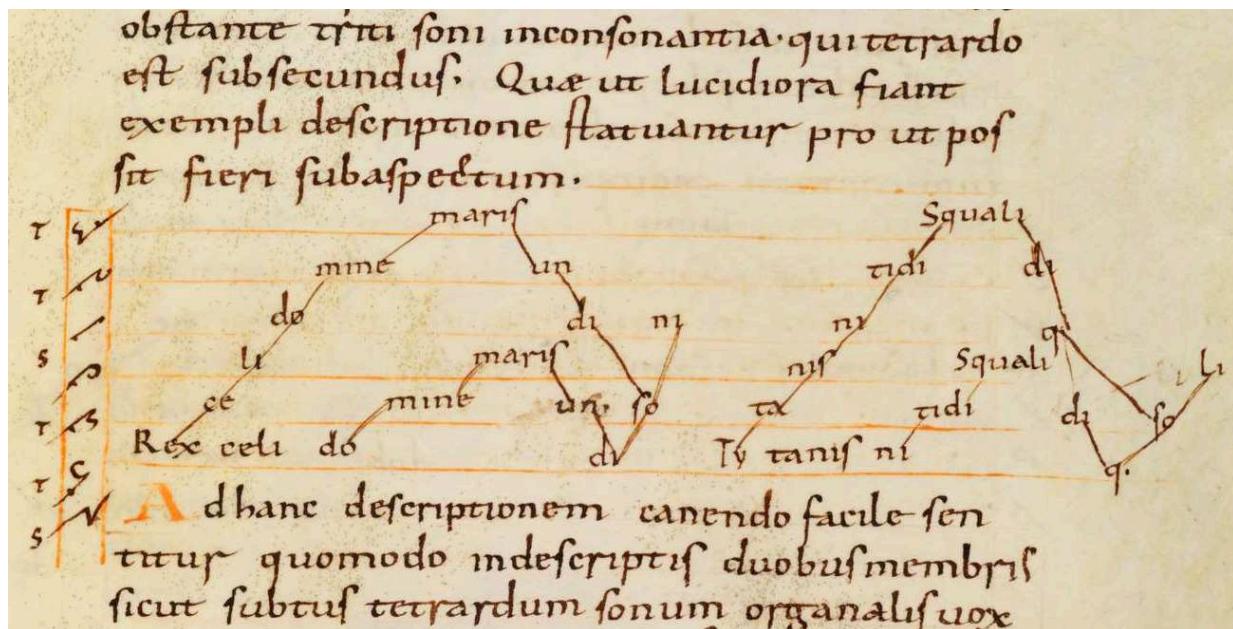
<http://www.e-codices.unifr.ch/de/csg/0359/107>

Figure 25: A detailed view of p. 107 of the Codex Sangallensis 359 manuscript from the early 10th century, showing adiastematic neumes (St. Gallen, Stiftsbibliothek).



<https://archive.org/stream/palographiemusic15macq#page/60/mode/1up>

Figure 26: The beginning of *Tu es deus* in the Codex Benevento VI.34 manuscript, f. 59v, written around 1100, showing diastematic neumes (Biblioteca capitolare, Benevento). The thicker line marks the pitch 'f', the thinner one pitch 'c', a fifth higher, as indicated by the letters at the beginning of the lines. Those letters eventually became the clefs in modern notation.



https://commons.wikimedia.org/wiki/File:Musica_enchiriadis_Rex_celi.png

Figure 27: An image from *Musica enchiriadis* (Staatsbibliothek Bamberg, Var. 1, fol 57r). Each line corresponds to a chord of a harp-like instrument.



<http://www.kalosconcentus.org/images/Musiche/Intr%20Rorate.jpg>, with permission

Figure 28: The *Graduale Triplex* (published by the Abbaye Saint-Pierre de Solesmes in 1979) shows both the diastematic neumes from Metz (above, black) and the adiastematic ones from St. Gallen (below, red), together with square-note neumes taken from the *Graduale Romanum*.



Figure 29: The beginning of Pérotin's *Viderunt Omnes* from the *Magnus liber organi*, composed around 1200, starting with syllable "Vi". This is the first known Quadruplum, a piece with four different voices (Biblioteca Medicea Laurenziana, Pluteus 29.1, f. 1, Florence).

Figure 30: Transcription of *Viderunt omnes* from fig. 29, using modern notation.



Figure 31: The beginning of the Gloria of Guillaume de Machaut's famous *Messe de Nostre Dame*, composed around 1360, in black mensural notation (ms. Machaut B, f. 283v–284r, Bibliothèque nationale de France).

The transcription below shows the musical structure with time signatures and bar lines:

Et in ter - ra pax ho - mi - ni bus bo - ne vo lun - ta -
tis. Lau - da - mus te. Be - ne - di - ci - mus te. Ad - o - ra - mus te. Glo - ri - fi - ca -
mus te. Gra - ti - as a - gimus ti - bi propter mag - nam glo - ri - am tu - am.

Figure 32: A zoom into the highest voice of the Gloria, together with a transcription to modern notation. Since the original manuscript does not have time signatures, the grouping into bars is rather arbitrary.



http://digi.vatlib.it/view/MSS_Chig.C.VIII.234/0384

Figure 33: The beginning of the *Missa L'homme armé* (*Kyrie eleison*), written by Josquin des Prez (~1452–1521), in white mensural notation (Biblioteca Apostolica Vaticana, Chig.C.VIII.234, f. 191v).

All elements of modern notation except bar lines are present. Using paper instead of parchment made it necessary to use less ink to avoid damage, thus the hollow ('white') note heads.

<http://riccar.cesr.univ-tours.fr/3-programmes/EMN/luth/sources/consult.asp?numnotice=4&ID=Capirola&index=94>

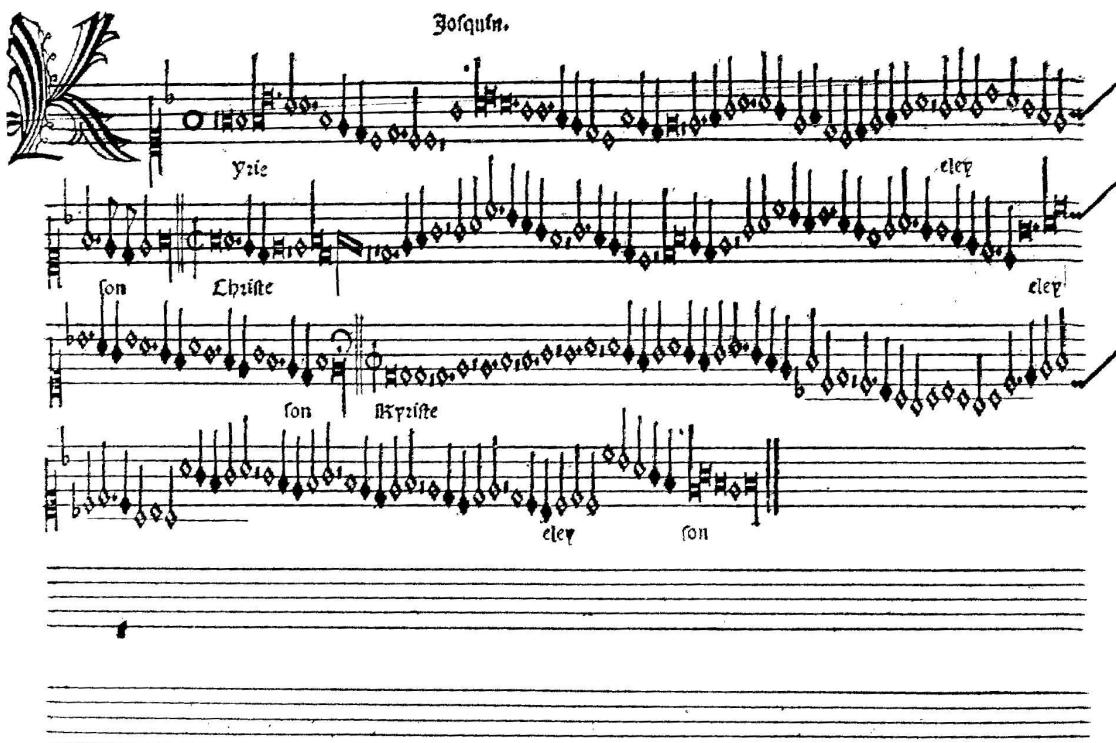
Figure 34: *Padoana a la francese* from Vincenzo Capirola's lute book, written around 1517 (Newberry Library, Chicago, MS VM C.25, f. 47r).



<http://imslp.org/wiki/Special:ImagefromIndex/111320>

Figure 35: The chorale *Wir Christenleut*, BWV 612, from J. S. Bach's *Orgelbüchlein* manuscript, written around 1715. The last 2½ bars are notated in German organ tablature (Staatsbibliothek zu Berlin Preussischer Kulturbesitz, Mus. ms. autogr. Bach P 283).

Figure 36: A zoom into BWV 512. In the autograph, each line with German Kurrentschrift letters represents a voice; uppercase letters denote pitches one octave lower, letters with a line above one octave higher. A sharp accidental is indicated by a trailing curved stroke below the baseline. Flats are not used, thus the note sequence 'g-f-e flat-d' is notated as 'g-f-d sharp-d', for example. Superscript digits and other symbols above the letters indicate duration (e.g., '4' for four semiquavers, '1' for a whole note).



[http://imslp.org/wiki/Missarum,_Book_1_\(Josquin_Desprez\)](http://imslp.org/wiki/Missarum,_Book_1_(Josquin_Desprez))

Figure 37: This page of a partbook, printed in Venice by Ottaviano Petrucci in 1502, shows the same mass from Josquin as figure 33.

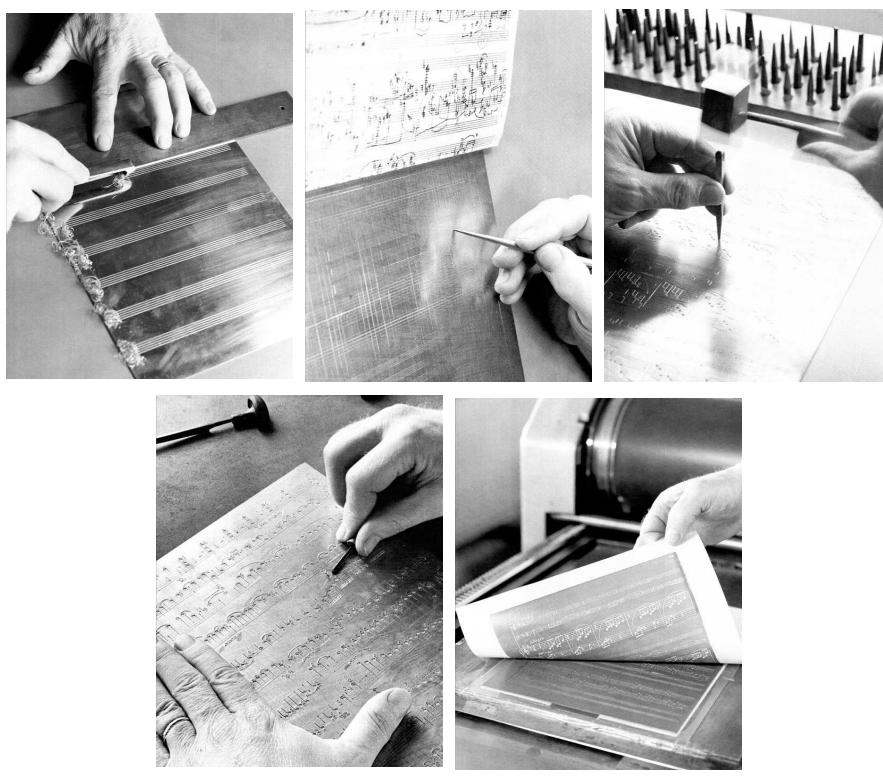
Figure 38: A direct comparison between the Josquin manuscript (fig. 33) and the printing (fig. 37). Note that the print uses a different clef. It also contains some errors and variants, probably due to a different manuscript copy.



http://www.bibliotecavirtualdeandalucia.es/catalogo/catalogo_imagenes/grupo.cmd?path=1000562&presentacion=pagina&posicion=29

Figure 39: A page from Miguel de Fuenllana's *Orphenica lyra* (printed 1554), a tablature for the vihuela (an early guitar), with bars. Red numbers indicate the melody. A rhythm indicator is only specified if a rhythm changes.

Figure 40: A copper plate engraving of the air *En fin la beauté* from Etienne Moulinie, published 1624.



<http://www.musicprintinghistory.org/music-engraving/13-about-music-engraving>

Figure 41: The process of manual music engraving. For complicated scores, it could easily take a day to finish a single plate.

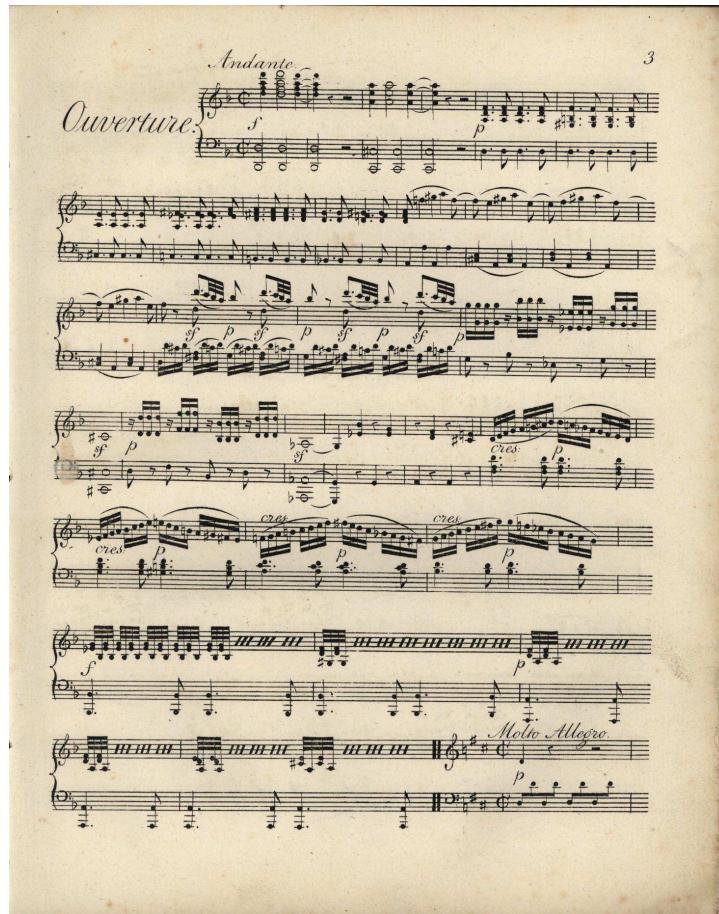


Figure 42: A lithography print from 1805 of a piano reduction of Mozart's overture to *Don Giovanni*, produced by Johann André's printing company in Paris (he held a patent on lithography).

<http://www.wurlitzerbruck.com/images/MUS/Mozart%20Don%20Giovanni%20overture%2010589.jpg>

An excerpt of Rachmaninoff's second piano sonata, engraved in 1914. The music is written for two staves. The top staff features a dynamic of *p* and a tempo marking of *Poco più mosso*. The bottom staff has a dynamic of *f*. The notation includes various note heads, stems, and rests, with some markings like *cresc.* and *decresc.*

<http://imslp.org/wiki/Special:ImagefromIndex/69058>

Figure 43: An excerpt of Rachmaninoff's second piano sonata, engraved in 1914.

Colorful emojis via Unicode and OpenType

Hans Hagen

A recent new (and evolving) addition to OpenType is colored glyphs. One variant (by Microsoft) uses overlays and this method is quite efficient.

```
\definefontfeature[colored] [colr=yes]
\definefontsynonym[Emoji]
  [file:seguiemj.ttf*default,colored]

\definesymbol[bug] [\getglyphdirect{Emoji}
  {\char"1F41B}]
\definesymbol[ant] [\getglyphdirect{Emoji}
  {\char"1F41C}]
\definesymbol[bee] [\getglyphdirect{Emoji}
  {\char"1F41D}]
```

Here we see a 🐛, 🐞 and 🐝, and they come in color! Since Unicode has started adding such symbols (and more in each release) the distinction between characters and symbols becomes even fuzzier. Of course one can argue that we communicate in pictograms but even then, given that mankind may last a while yet, the Unicode repertoire will explode.

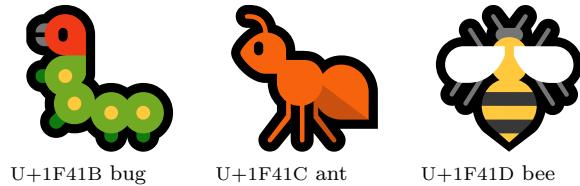


Figure 1: A few emojis from `seguiemj.ttf`.

Above we have used `seguiemj.ttf`, a font that comes with Windows. Colors are achieved by combining glyphs rendered in different colors. A variant font that uses SVG instead of overlays is

```
emojionecolor-svginot.ttf:
\definefontfeature[svg] [svg=yes]
\definefontsynonym[Emoji]
  [file:emojionecolor-svginot.ttf*default,svg]
```

This time we get 🐛, 🐞 and 🐝 and they look quite different. Both fonts also have ligatures and you can wonder what sense that makes. It makes it impossible to swap fonts and as there is no standard one never knows what to expect.



Figure 2: The same emojis from `emojionecolor-svginot.ttf`.

How do we know what faces add up to the ligature 🧑 and how are we supposed to know that there should be `zwj` between? When we input four faces separated by zero width joiners, we get a four face symbol instead. The reason for having the joiners is probably to avoid unexpected ligatures. The sequence man, woman, boy, boy gives family: 🧑 + zwj 🧑 + zwj 🧑 + zwj 🧑 = 🧑, but two girls also works: 🧑 + zwj 🧑 + zwj 🧑 + zwj 🧑 = 🧑, and so does a mixture of kids: 🧑 + zwj 🧑 + zwj 🧑 + zwj 🧑 = 🧑, although (at least currently): 🧑 + zwj 🧑 + zwj 🧑 + zwj 🧑 = 🧑 (not stacked). To add to the random fun, the official Unicode family U+1F46A has only three members (in this font): 🧑.

In our times for sure many combinations are possible, so: 🧑 + zwj 🧑 + zwj 🧑 + zwj 🧑 = 🧑 indeed gives a family, but I wonder at what point cultural bias will creep into font design. One can even wonder how clothing and hair styles will demand frequent font updates: 🧑🧑🧑🧑🧑🧑.

In the math alphabets we have a couple of annoying holes because characters were already present in Unicode, so now we forever have to deal with those exceptions. But not so with emojis because here eventually all variants will show up. Although a character A in red or blue uses the same code point, a white telephone (not in this particular font) and black telephone 📞 have their own. And because obsolete scripts are already supported in Unicode and more get added, we can expect old artifacts also showing up at some time. Soon the joystick 🕹 will be an unknown item to most, while the Microsoft hololens might get its slot.



Figure 3: Will all animals come in all stages of development?

For sure these mechanisms will evolve and to what extent we support them depends on what users want. At least we have the basics implemented.

◊ Hans Hagen
Pragma ADE
<http://pragma-ade.com>

Cowfont (koeiletters) update

Taco Hoekwater, Hans Hagen

Abstract

After ten years, the ‘koeiletters’ font is ready for an update. The new version uses OpenType technology to combine the existing four PostScript Type 1 fonts into a single TrueType font. It’s sort of a coincidence that at the tenth ConTeXt meeting, the font also celebrates its tenth birthday.

1 A bit of history¹

1.1 The artful beginnings

At TUG 2003 in Hawaii, Hans Hagen met with Duane Bibby. Hans was looking for some small images to enliven the ConTeXt manuals and Wiki. A cutout of a very early sketch can be seen in figure 1, but it was soon agreed that consecutive drawings were going to be an alphabet.

Nothing much happened after that initial meeting until the beginning of 2006 when Hans picked up the thread and got Duane started drawing. The alphabet quickly progressed. Starting in a rather naturalistic style like Duane’s ‘normal’ TeX drawings, but later progressing toward a much more cartoon-like style, as can be seen from the drawings in figure 2.

For ease of use, it was clear that these drawings should ideally become a computer font. Taco Hoekwater agreed to take care of the digitization, and luckily the drawings were already prepared for that. As can be seen from the leftmost closeup in figure 3, the cows are drawn inside a grid. This ensures that they are all the same size, which is a vital requirement for a font design. But of course this is a proportional font in the end; it even has kerning and ligatures!

The center drawing in figure 3 is a still rather roughly inked version of one of the in-between drawings (there were many). In this particular one you can see that the mouth of the cow was originally more or less oval, but in the final form (on the right) it became much more hexagonal.

1.2 Digitization

The original sheets were sent to Pragma ADE by regular mail in the beginning of March 2006. Hans scanned the original sheets at 1200 dpi and then forwarded the images to Taco. There were four sheets in all, containing an alphabet with some accents,



Figure 1: The first drawing

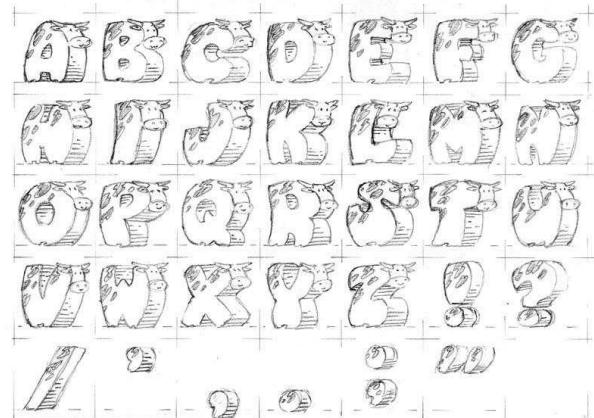


Figure 2: Rough design

Latin punctuation, and a number of TeX-related logos and a few (mathematical) symbols.

The four sheets were digitally cut up into many smaller pieces, each containing a single glyph for the font. This being intended as a decorative font, the character set does not even contain the complete ASCII range. Nevertheless, almost a hundred separate images were created.

These were then imported into FontForge. The autotracer in FontForge, which is actually the stand-alone `autotrace` program, does quite a good job of tracing the outlines. But, interestingly enough, only at a fairly low resolution. At higher resolutions it gets confused and inserts more than a quadratic amount of extra points as the resolution is increased. Based on empirical tests, the images were scaled to 40% of their original scanned size, resulting in bitmaps that were precisely 1000 pixels high.

¹ This section is an abbreviated version from our article ‘The making of a (TeX) font’, MAPS 34 (2006), pages 51–54. <http://www.ngt.nl/maps/34/11.pdf>

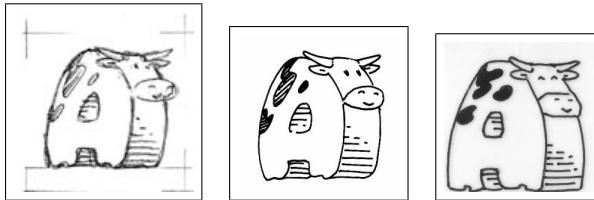


Figure 3: Closeups of the progressive design stages of the letter ‘A’.

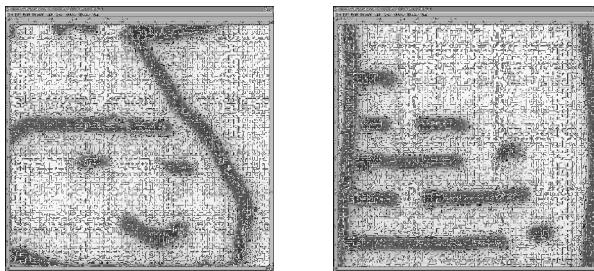


Figure 4: Close-ups of autotracer output

As was to be expected, the autotracer brought out many of the impurities in the original inked version, as you can see in the left image of figure 4. Luckily, the number of places where manual corrections like this were needed was not so great to force us to reconsider the digitization process.

A more severe problem can be seen in the right-hand image of figure 4. The drawings contain hardly any straight lines. For a font of this complexity, it turned out to be absolutely necessary to simplify the curves. Without simplification, the rendering speed in PDF browsers became unbearably slow. All of the near-horizontal stripes in the bellies were manually removed and replaced by geometric straight lines.

The final stage in the font editor is to add the PostScript hinting. A screenshot of a manually hinted letter is visible in figure 5.

1.3 Finishing the font

The font was saved as two separate PostScript Type 1 fonts, one with the text glyphs and one containing the logo glyphs. The text font is named ‘koeiletters’, the logo font ‘koeielogos’. ‘Koeiletters’ literally translates from Dutch to English as ‘cowcharacters’, but the word ‘koeiletter’ is also used to indicate an enormous character, as in a billboard, for instance.

Eventually it turned out that we needed a second set of two fonts. Sometimes you want to have text in the cowfont but on top of a colored background. The background would then shine right through the hide of the cow and that was of course unacceptable. Hence, we also have the fonts ‘koeiletters-contour’ and ‘koeielogos-contour’.

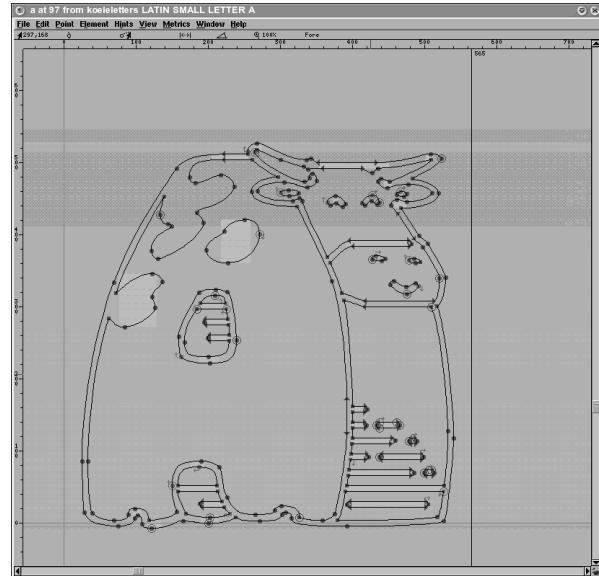
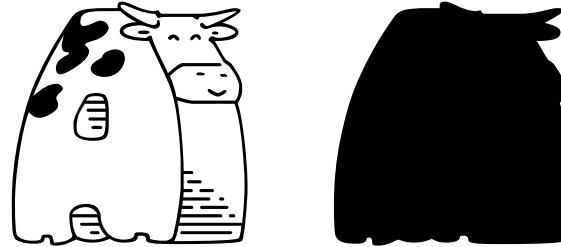


Figure 5: Finished outline

Here is the final ‘A’, in the normal and the contour version:



2 Updated version

In ConTeXt MkIV, we prefer not to use Type 1 fonts, and definitely not the tfm-based trickery that was needed to get the ‘koeiletters’ font performing at its best. Advances in font technology have made it possible to combine all glyphs into a single OpenType font, which goes by the name **koeilettersot**.

2.1 Mathematics

The original Type 1 font already had a math companion but the new font supports math via its ‘MATH’ table, allowing it to be used for math typesetting just like the other OpenType math fonts that ConTeXt uses, with only a few minor differences:

- There are far fewer glyphs, due to a lack of original artwork. You can imagine that providing the full repertoire of Unicode math would be a bit of a challenge.
- ConTeXt has to do some extra tweaking for the horizontal extensible rules, including those that are appended to radicals.

- There are no accented characters but much can be achieved by enabling the `compose` feature.

2.2 Ligatures for logos

In this font, there is no ‘fi’ ligature. In fact there are no ‘normal’ ligatures at all. However, there is a `dlig` feature in the font which replaces words by hand-drawn versions of those words, and the `ss02` feature can be used to convert these further, into nicer versions with a drop-shadow below.

2.3 Sheep

The numbers and plus and minus in the font can be replaced by versions that resemble a sheep instead of a cow, by enabling the `ss01` feature.

2.4 Colorization

In mid-2016, the ConTEXt font loader started supporting color fonts. Such fonts normally contain emoji characters and for achieving the desired effect two methods are available: overlays and SVG. The first method is cleaner and naturally fits ‘koeieletters’.

The trick is in splitting a glyph into overlaying snippets that each can have a color from a palette. Emoji fonts can provide multiple palettes so that culturally-based colors can be supported. So eventually we could have black Frisian cows and brown ones from the southern part of our country.

The implementation uses virtual fonts. This is straightforward but the current way to inject the needed color directives and information to cut-and-paste the right character can interfere with the way the backend flushes characters. As we managed it with some hackery eventually the virtual font technology might be extended a bit for this purpose.

More challenging was to get math working. Not so much math itself but where regular math fonts use rules for extending radicals, over- and underbars and fractions, we need to use something cowish. Possible solutions are:

- Build the radicals from scratch using snippets: this is cumbersome.
- Preroll with normal rules that get replaced in the node list later: one has to know in what ways TeX constructs glyphs because not every rule is a radical one.
- Patch the math engine to support complex radicals: after some experiments this was considered too dangerous and messy.
- Make the math rules pluggable: adding more callbacks makes no sense for this one exception.

- Make the math rules be (optional) user rules that can be postprocessed: this was relatively easy.

It should be clear that the last solution was chosen. Of course it was not as trivial as we make it sound. First, for radicals we need to register what font we are dealing with so that we can get the right snippets to construct a rule. For the other rules we need to know the font as well and it happens that no such information is available: rules don’t come from fonts. The solution is in two new primitives:

```
% use math specific user nodes:  
\mathrulenode = 1  
% the family to take rules from:  
\mathrulefamily = \fam\textstyle
```

When set, special rules will be constructed that carry the current size (text, script or scriptscript) and family-related font. In the backend the serialization of these rule nodes will trigger a callback (when set) that can inject whatever is reasonable. Of course these extensions are still somewhat experimental and should be used with care.

2.5 Using the font

So how is this new font used? Although it is a special kind of font that will seldom be used for a whole document, you need to load it anyway. The easiest way (in ConTEXt) is:

```
\loadtypescriptfile[koeielettersot]  
\setupbodyfont[cows,12pt]
```

Please take a look at `type-imp-koeielettersot` to see how these fonts get set up. The beginning of ConTEXt’s usual example Zapf quote (“Coming back to the use of typefaces . . .”) comes out as follows:

COMING BACK TO THE USE OF
TYPEFACES IN ELECTRONIC
PUBLISHING: MANY OF THE NEW
TYPEGRAPHERS . . .

If you want a colored variant a bit more work is needed. By default the cows are black and white. If you enable color you will see the difference when you show them on a background:

COMING BACK TO THE USE OF
TYPEFACES IN ELECTRONIC
PUBLISHING: MANY OF THE NEW
TYPEGRAPHERS . . .

When a font is loaded its color properties are frozen because the backend needs to deal with it.

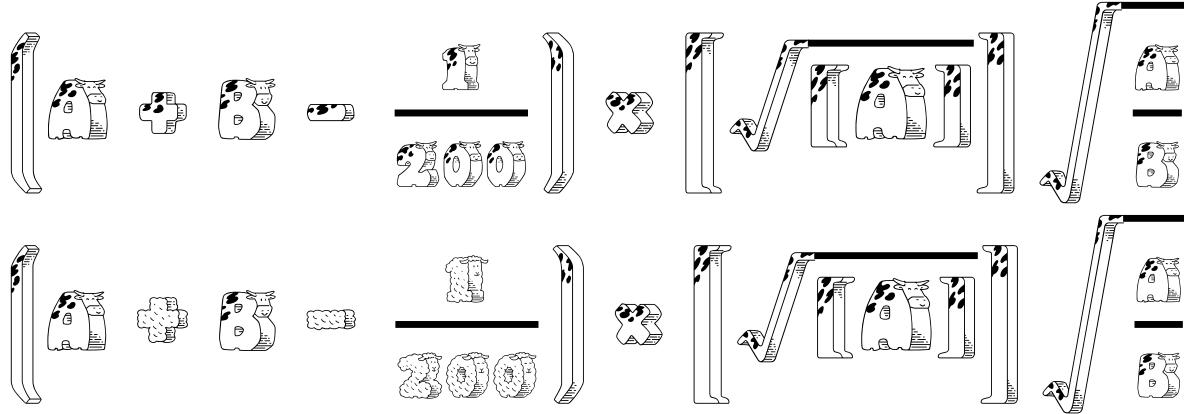


Figure 6: A math formula rendered in ‘koeieletters’; cows above, sheep below.
The standard black rules in fractions and radicals are fixed in the next figure.

You can, however, influence the color with the `colr` property before a font gets defined. This happens just after loading the typescript file.

```
\definecolor[cowred] [r=.50]
\definecolor[cowgreen] [g=.50]
\definecolor[cowblue] [b=.50]
\definecolor[cowyellow] [y=.25]
\definefontcolorpalette[cows]
  [cowgreen,cowyellow,cowblue,cowred]
\adaptfontfeature[sheepcolored] [colr=cows]
```

In the example below we show the sheep with colors because we already defined the cows as black and white. You can mix colors by defining fonts explicitly. Note that we only use the second and fourth color in these glyphs.

```
\usetypescript[all][cowsotf]

\definefontcolorpalette[cows-1][cowgreen,
  cowyellow,cowblue,cowred]
\definefontcolorpalette[cows-2][cowred,
  cowyellow,cowblue,cowgreen]
\definefontcolorpalette[cows-3][cowgreen,
  cowyellow,cowred,cowblue]

\definefontfeature[cows-1]
  [cowscolored][colr=cows-1]
\definefontfeature[cows-2]
  [cowscolored][colr=cows-2]
\definefontfeature[cows-3]
  [cowscolored][colr=cows-3]

\definedfont[Cows*cows-1 at 30pt]red\quad
\definedfont[Cows*cows-2 at 30pt]green\quad
\definedfont[Cows*cows-3 at 30pt]blue
```

RED GREEN BLUE

2.6 Math

As said, we can do math. Take this formula:

```
\left( a + b - \frac{200}{300} \right) \times
\left[ \sqrt{A} \right] \sqrt{\frac{a}{b}}
```

This renders as shown in figure 6, cows above, sheep below. The standard rules there don’t work well, but figure 7 shows we can do better (implemented with the `\mathrulesmode` mentioned above).

2.7 Logos

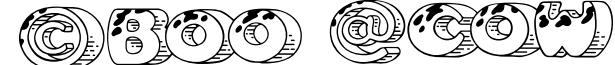
There’s a bunch of logos available. You can directly request them but they can also be set automatically.

```
\definefont      [CowsLogo]
  [koeielettersot*cowslogos    sa c]
\definefont      [CowsLigs]
  [koeielettersot*cowsligatures sa c]
\definefontsynonym[CowsOnly]
  [koeielettersot]
```

These definitions can be used to get the logos shown in 8. The last two columns in the table are typeset using:

```
\getnamedglyphdirect{CowsOnly}{contextlogo}
\getnamedglyphdirect{CowsOnly}{c_o_n_t_e_x_t}
```

There are two more ligatures:



and we leave it to you to figure out how to get them.

We end with the best of all: a colored logo.

```
\definefontsynonym
  [CowsColored]
  [koeielettersot*default,cowscolored]

\getnamedglyphdirect{CowsColored}{contextlogo}
```

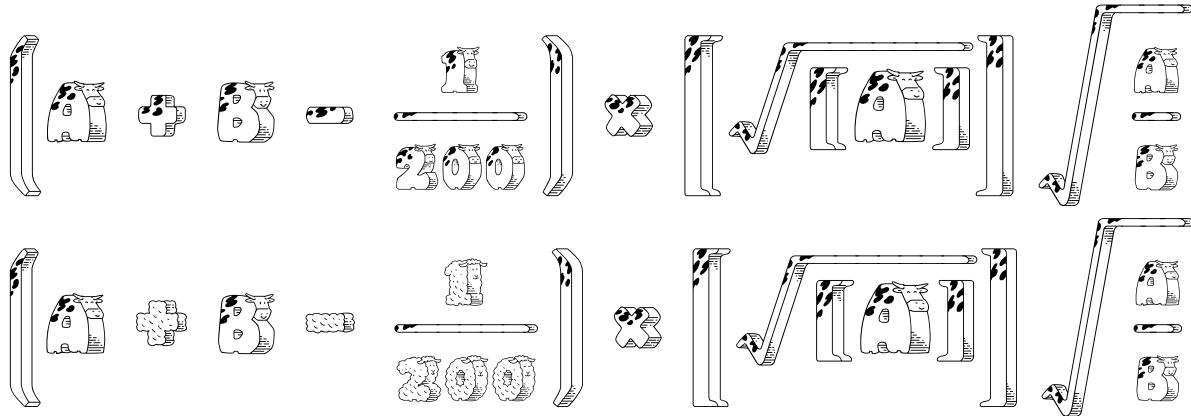


Figure 7: The same math formula as the previous figure, with matching rules created using `\mathrulesmode`.

input	<code>\CowsLogo</code>	<code>\CowsLigs</code>	<code>somelogo</code>	<code>s_o_m_e_l_o_g_o</code>
PragmaAde				
pragmaade				
context				
MP				
TeX				
metafun				
Example				
FoXeT				
TEX				
Wiki				
Cowtext				

Figure 8: Logos in ‘koeieletters’.



To make a quick start with these fonts, you can use one of:

```
\setupbodyfont[koeieletters]
\setupbodyfont[cows]
\setupbodyfont[coloredcows]
\setupbodyfont[sheep]
\setupbodyfont[coloredsheep]
```

where the `koeieletters` variant equals `sheep`. This is possible because we aliased the typescriptfiles to the predefined typeface setups in the typescript file.

◊ Taco Hoekwater, Hans Hagen
ConTeXt Group
<http://contextgarden.org>

Corrections for slanted stems in METAFONT and METAPOST

Linus Romer

Abstract

Slanting an outline font may change the width and angles of stems. The following article presents some formulae to correct these effects and provides corresponding METAFONT and METAPOST macros.

1 Slanting

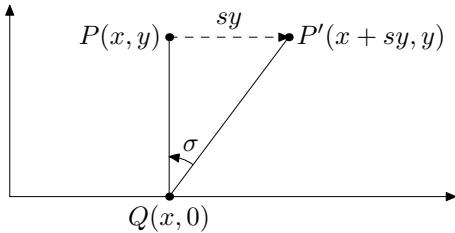
Slanted typefaces are quite common; they are usually called “oblique”. E.g., the *URW Gothic L Book* face is slanted forward by an angle of $\approx 10.5^\circ$ resulting in the *URW Gothic L Book Oblique* face:



Indeed, even the italic faces in *Computer Modern* are designed unslanted, to then be slanted forward by an angle of $\arctan(0.25) \approx 14^\circ$:



In this article, we will assume that slanting means horizontal *shearing*, which is the correct expression in mathematics. The following picture allows us to describe slanting mathematically:



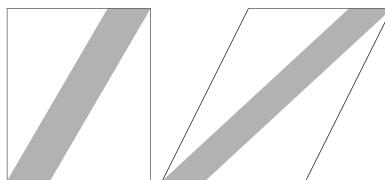
Every coordinate vector $\begin{pmatrix} x \\ y \end{pmatrix}$ is mapped to $\begin{pmatrix} x+sy \\ y \end{pmatrix}$, where s is the slanting amount:

$$\begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} 1 & s \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x+sy \\ y \end{pmatrix}$$

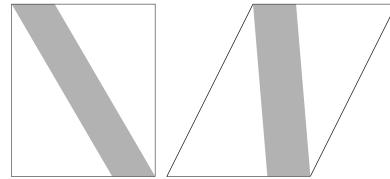
The directed angle $\sigma = \arctan(s)$ denotes the slanting angle. Note that s and σ are negative if and only if the slanting is backward.

2 Width correction for slanted stems

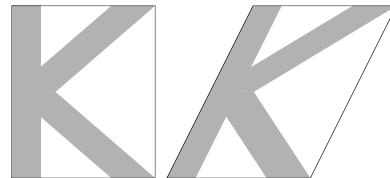
Slanting forward makes forward leaning outline stems slimmer:



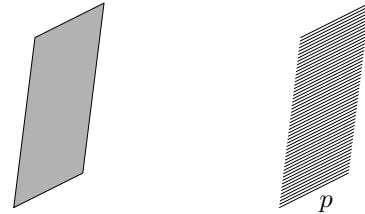
Conversely, slanting forward can make backward leaning stems fatter:



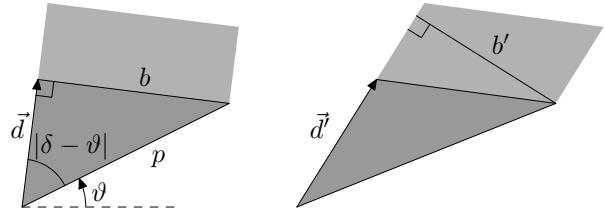
This affects slanted glyphs. E.g., a “K” with even width stems will have stems of different widths after slanting:



Outline stems can be imagined to be drawn by a razor pen of penwidth p .



We want to express the pen width p (before slanting) in terms of the future stem width b' (after slanting), the drawing direction \vec{d} and the pen angle ϑ .



In the figures above, \vec{d}' denotes the slanted vector \vec{d} . The angle $\delta = \text{angle}(\vec{d})$ is the directed angle between $\begin{pmatrix} 1 & s \\ 0 & 1 \end{pmatrix}$ and \vec{d} . The directed pen angle is denoted as ϑ . The distances b and b' are the heights of the non-slanted marked triangle and the slanted marked triangle, respectively. As we know, slanting is an area-preserving transformation. Hence, $|\vec{d}| \cdot b = |\vec{d}'| \cdot b'$ and thus

$$b = b' \cdot \frac{|\vec{d}'|}{|\vec{d}|} = b' \cdot \left| \frac{\begin{pmatrix} 1 & s \\ 0 & 1 \end{pmatrix} \cdot \vec{d}}{|\vec{d}|} \right| = b' \cdot \left| \begin{pmatrix} 1 & s \\ 0 & 1 \end{pmatrix} \cdot \frac{\vec{d}}{|\vec{d}|} \right|$$

By applying the definition of the sine function on the non-slanted triangle $\sin |\delta - \vartheta| = \frac{b}{p}$, we obtain the solution:

$$p = b' \cdot \left| \frac{\begin{pmatrix} 1 & s \\ 0 & 1 \end{pmatrix} \cdot \frac{\vec{d}}{|\vec{d}|}}{\sin (\text{angle}(\vec{d}) - \vartheta)} \right| \quad (1)$$

This formula has been published for the special case $|\text{angle}(\vec{d}) - \vartheta| = 90^\circ$ as “slant correction formula” by Jackowski, Nowacki, and Strzelczyk, 2000.

Here is equation (1) as a macro in METAFONT and METAPOST:

```
def penwidth(expr b,d,theta,s) =
  b*abs(length((d/length(d)) slanted s)
    / sind(angle(d)-theta))
enddef;
```

3 Fitting given boxes

In the following subsections, we will fit diagonal stems in some way into a given rectangular box, such that the diagonal will have the required width b *after* slanting. This is equivalent to the condition requiring that diagonal stems of a required width b fit a given slanted rectangular box (which is a parallelogram).

The following figures visualize the different meanings of “fitting” a stem of width = 10 pt into a slanted box of height = width = 50 pt.

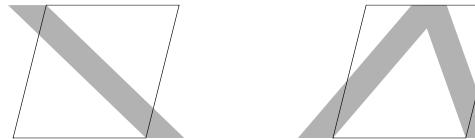
Inscribing (first) diagonal leans forward:



Inscribing (first) diagonal leans backward:



Overlapping (last) diagonal leans backward:



Overlapping (last) diagonal leans forward:



“Half inscribing” chained diagonals:

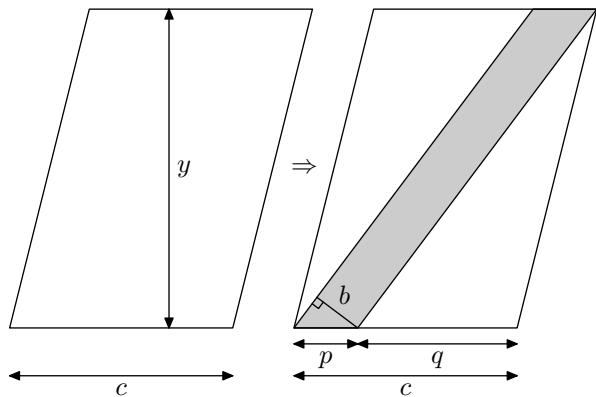


The source code for the preceding figures is given in subsection 3.4.

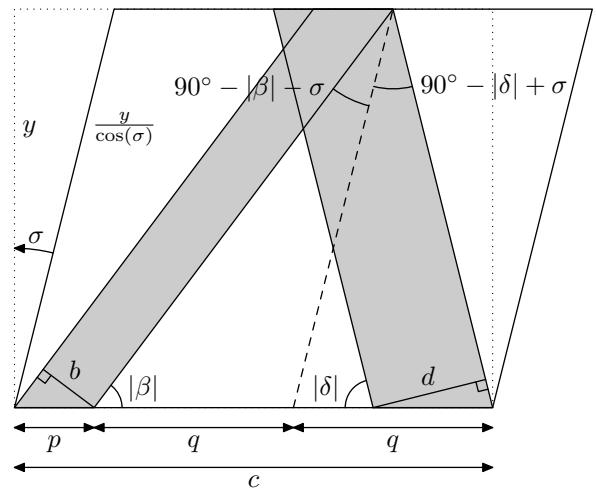
3.1 Inscribing diagonals

3.1.1 First diagonal forward

First, we are trying to *inscribe* a forward diagonal stem of width b into a rectangular box of width c and height y which was slanted by a slanting amount s . We will try to find the penwidth p as depicted below. Jackowski, Nowacki, and Strzelczyk (2000) have already algorithmically solved this problem by defining the binary operator $/\wedge$. Here, we will find an exact solution.



We can generalize the situation by introducing the variable a , which may be toggled between 1 and 2 and which stands for the number of inscribed (chained) diagonals. The situation for $a = 2$ is shown below:



The small rectangular triangle at the left foot leads to the relation

$$p = \frac{b}{\sin |\beta|}.$$

The sine theorem for the left large triangle yields (remember that $\cos(\sigma) > 0$)

$$\frac{q}{\sin(90^\circ - (|\beta| + \sigma))} = \frac{y/\cos(\sigma)}{\sin |\beta|}.$$

Hence, we obtain

$$\begin{aligned}
 q &= \frac{y \sin(90^\circ - (|\beta| + \sigma))}{\cos(\sigma) \sin |\beta|} \\
 &= \frac{y \cos(|\beta| + \sigma)}{\cos(\sigma) \sin |\beta|} \\
 &= y \cdot \frac{\cos |\beta| \cos(\sigma) - \sin |\beta| \sin(\sigma)}{\cos(\sigma) \sin |\beta|} \\
 &= y \cdot (\cot |\beta| - \tan(\sigma)) \\
 &= y \cdot \left(\sqrt{1/\sin^2 |\beta| - 1} - s \right) \\
 &= y \cdot \left(\sqrt{p^2/(b^2 - 1)} - s \right).
 \end{aligned} \tag{2}$$

Looking at the total width c , we get

$$c = p + aq = p + ay \left(\sqrt{p^2/(b^2 - 1)} - s \right).$$

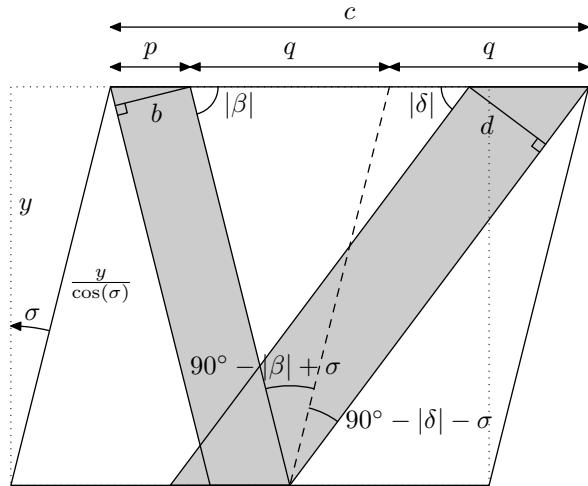
As $p^2/b^2 \geq 1$, this is an ordinary quadratic equation with the solutions

$$p = b \cdot \frac{-b(c + asy) \pm ay\sqrt{(c + asy)^2 + a^2y^2 - b^2}}{a^2y^2 - b^2}.$$

Because $-b(c + asy) \leq 0$, we are left with the only non-negative solution:

$$p = b \cdot \frac{-b(c + asy) + ay\sqrt{(c + asy)^2 + a^2y^2 - b^2}}{a^2y^2 - b^2} \tag{3}$$

3.1.2 First diagonal backward



Again

$$p = \frac{b}{\sin |\beta|}.$$

The sine theorem for the left large triangle yields

$$\frac{q}{\sin(90^\circ - (|\beta| - \sigma))} = \frac{y/\cos(\sigma)}{\sin |\beta|}.$$

Analogously as in subsection 3.1.1, we obtain

$$q = y \cdot \left(\sqrt{p^2/(b^2 - 1)} + s \right).$$

The equation

$$c = p + aq = p + ay \cdot \left(\sqrt{p^2/(b^2 - 1)} + s \right)$$

then leads to the solution

$$p = b \cdot \frac{-b(c - asy) + ay\sqrt{(c - asy)^2 + a^2y^2 - b^2}}{a^2y^2 - b^2}. \tag{4}$$

3.1.3 Generalization

After introducing the boolean variable e :

$$e = \begin{cases} -1 & \text{if first stem is leaning backward} \\ 1 & \text{otherwise} \end{cases}$$

we can generalize equations (3) and (4) to

$$p = b \cdot \frac{-b(c + aesy) + ay\sqrt{(c + aesy)^2 + a^2y^2 - b^2}}{a^2y^2 - b^2}$$

Replacing the b 's and c 's by $b_b = \frac{b}{y}$ and $c_c = \frac{c}{y}$ (avoiding arithmetic overflow) yields

$$p = b \cdot \frac{-b_b(c_c + aes) + a\sqrt{(c_c + aes)^2 + a^2 - b_b^2}}{a^2 - b_b^2} \tag{5}$$

If we calculate the diagonal ratio p/b with equation (5) for the slanting amount $s = 0$, we get the *Computer Modern* macro `diag_ratio` (Knuth, 2002):

$$\text{diag_ratio}(a, b, y, c) = b \cdot \frac{-b_b c_c + a\sqrt{c_c + a^2 - b_b^2}}{a^2 - b_b^2}$$

Thus, *Computer Modern* letters like A , V , Λ are computed without slanting corrections. Because of the unequal stem widths, this is not visible for the serif faces. However, the sans faces have uneven stem widths after slanting:

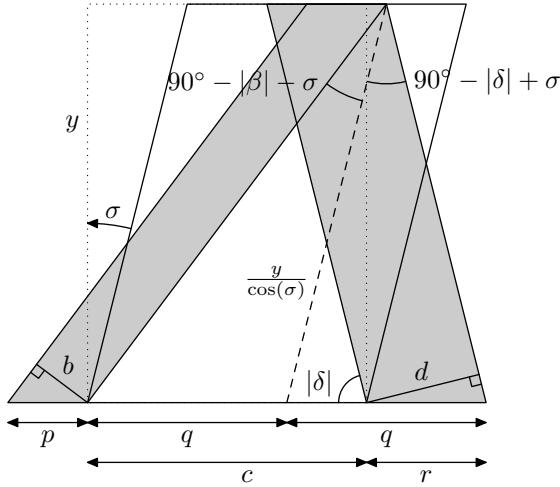


Equation (5) as a METAFONT/METAPOST macro:

```
vardef penwidththin(expr a,e,b,c,y,s) =
  numeric bb,cc; bb=b/y; cc=c/y;
  b*(-bb*(cc+a*e*s)+a*sqrt((cc+a*e*s)**2
    +a*a-bb*bb))/(a*a-bb*bb) enddef;
```

3.2 Overlapping diagonals

3.2.1 Last diagonal backward



The situation is almost the same as in subsection 3.1.1, the only difference being that the diagonals start outside the box and that we are looking for r :

$$r = \frac{d}{\sin |\delta|}$$

The sine theorem for the right large triangle yields

$$\frac{q}{\sin(90^\circ - (|\delta| - \sigma))} = \frac{y / \cos(\sigma)}{\sin |\delta|}.$$

Analogous to subsection 3.1.1, we obtain

$$\begin{aligned} q &= y \cdot (\cot |\delta| + \tan(\sigma)) \\ &= y \cdot \left(\sqrt{p^2/(b^2 - 1)} + s \right). \end{aligned} \quad (6)$$

Again, we write the total width as

$$c = a \cdot q - r = ay \cdot \left(\sqrt{r^2/(d^2 - 1)} + s \right) - r.$$

As long as $ay > d$, the only nonnegative solution is

$$r = d \cdot \frac{d(c - asy) + ay\sqrt{(c - asy)^2 + a^2y^2 - d^2}}{a^2y^2 - d^2}. \quad (7)$$

3.2.2 Last diagonal forward

Analogous to the difference between subsection 3.1.2 and 3.1.1, this subsection differs from subsection 3.2.2 in changing $(90^\circ - |\delta| + \sigma)$ to $(90^\circ - |\delta| - \sigma)$. Hence, it suffices to substitute s by $-s$ in equation (7):

$$r = d \cdot \frac{d(c + asy) + ay\sqrt{(c + asy)^2 + a^2y^2 - d^2}}{a^2y^2 - d^2} \quad (8)$$

3.2.3 Generalization

As in subsection 3.1.3, we are able to generalize the equations (7) and (8) after introducing

$$e = \begin{cases} -1 & \text{if last stem is leaning backward} \\ 1 & \text{otherwise} \end{cases}$$

to

$$r = d \cdot \frac{d(c + aesy) + ay\sqrt{(c + aesy)^2 + a^2y^2 - d^2}}{a^2y^2 - d^2}$$

Replacing the d 's and c 's by $d_d = \frac{d}{y}$ and $c_c = \frac{c}{y}$ (avoiding arithmetic overflow) yields

$$r = d_d \cdot \frac{d_d(c_c + aes) + a\sqrt{(c_c + aes)^2 + a^2 - d_d^2}}{a^2 - d_d^2}. \quad (9)$$

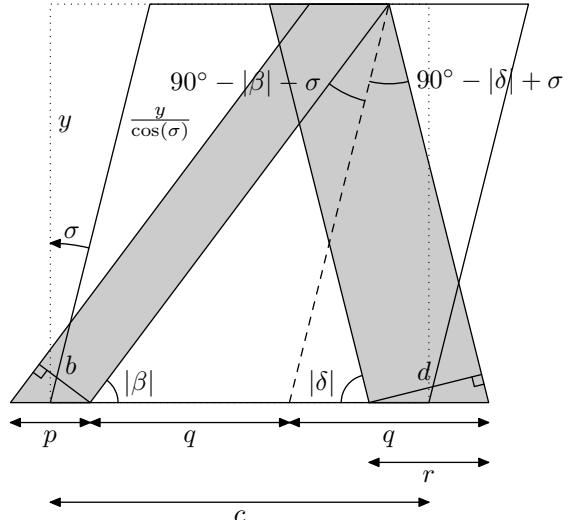
Equation (9) as a METAFONT/METAPOST macro:

```
vardef penwidthover(expr a,e,d,c,y,s) =
  numeric dd,cc; dd=d/y; cc=c/y;
  d*(dd*(cc+a*e*s)+a*sqrt((cc+a*e*s)**2
  +a*a-dd*dd))/(a*a-dd*dd) enddef;
```

Of course, equations (5) and (9) could again be easily united to one single equation.

3.3 Half inscribing diagonals

We are looking at two chained diagonals which fit the given box such that each penwidth is half inside the box. In contrast to the preceding problems, we do not have to consider the case with only one diagonal ($a = 1$) as this case is already covered with equation (1).



If we combine the equations (2) and (6) and include the boolean variable e from subsection 3.1.3, we obtain

$$\begin{aligned} y \cdot (\cot |\beta| - es) &= y \cdot (\cot |\delta| + es) \\ \implies \cot |\delta| &= \cot |\beta| - 2es. \end{aligned}$$

Looking at the total width c , we obtain

$$\begin{aligned} c &= 0.5p + 2q - 0.5r \\ \implies 2c &= p + 4q - r \\ \implies 2c &= b\sqrt{1 + \cot^2 |\beta|} + 4y(\cot |\beta| - es) \\ &\quad - d\sqrt{1 + (\cot |\beta| - 2es)^2}. \end{aligned}$$

This quartic equation can be solved exactly for $\cot |\beta|$, but the exact solution is long and tedious. For applications, this equation is best solved numerically, e.g., by the bisection method. In the end, we will find both widths p and r by

$$\begin{aligned} p &= b\sqrt{1 + \cot^2 |\beta|} \\ r &= d\sqrt{1 + (\cot |\beta| - 2es)^2}. \end{aligned}$$

The implementation in METAPOST with the bisection method returns the pair (p, r) :

```
vardef poswidthhalf(expr e,b,d,c,y,s) =
  numeric bb,cc,dd,ta,tb,t; % t=cot(beta)
  bb=b/y; dd=d/y; cc=c/y;
  ta=-100; tb=100; % boundaries
  forever:
    exitif abs(ta-tb)<=eps;
    t:=.5[ta,tb];
    if bb*(1+t)+4*(t-e*s)-dd*(1+(t-2*e*s))
      -2*cc>0: tb else: ta fi:=t;
  endfor
  (b*(1+t),d*(1+(t-2*e*s))) enddef;
```

3.4 Source codes for the box fitting figures

In the following, all METAPOST sources of the ten figures at the beginning of section 3 are given as building blocks in a compact form. The most important figures for font design are probably figures 0–3.

```
beginfig(0);
w:=50pt; h:=50pt; s:=.25; z1l=(0,0); z2r=(w,h);
z1r-z1l=z2r-z2l=(penwidthin(1,1,10pt,w,h,s),0);
penstroke z1e--z2e slanted s; endfig;

beginfig(1);
w:=50pt; h:=50pt; s:=.25; z1l=(0,0); z4r=(w,0);
z3r=z2r; y2r=h; x4r-x2r=x2r-x1r;
z1r-z1l=z2r-z2l=(penwidthin(2,1,10pt,w,h,s),0);
penstroke z1e--z2e slanted s;
z3r-z3l=z4r-z4l=(penwidth(10pt,z3r-z4r,0,s),0);
penstroke (z3e--z4e) slanted s; endfig;

beginfig(2);
w:=50pt; h:=50pt; s:=.25; z1l=(0,h); z2r=(w,0);
z1r-z1l=z2r-z2l=(penwidthin(1,-1,10pt,w,h,s),0);
penstroke (z1e--z2e) slanted s; endfig;

beginfig(3);
w:=50pt; h:=50pt; s:=.25; z1l=(0,h); z4r=(w,h);
z3r=z2r; y2r=0; x4r-x2r=x2r-x1r;
z1r-z1l=z2r-z2l=(penwidthin(2,-1,10pt,w,h,s),0);
penstroke (z1e--z2e) slanted s;
z3r-z3l=z4r-z4l=(penwidth(10pt,z3r-z4r,0,s),0);
penstroke (z3e--z4e) slanted s; endfig;
```

```
beginfig(4);
w:=50pt; h:=50pt; s:=.25; z1r=(0,h); z2l=(w,0);
z1r-z1l=z2r-z2l=(penwidthover(1,1,10pt,w,h,s),0);
penstroke (z1e--z2e) slanted s; endfig;

beginfig(5);
w:=50pt; h:=50pt; s:=.25; z1r=(0,0); z4l=(w,0);
z3r=z2r; y2r=h; x4r-x2r=x2r-x1r;
z3r-z3l=z4r-z4l=(penwidthover(2,1,10pt,w,h,s),0);
penstroke (z3e--z4e) slanted s;
z1r-z1l=z2r-z2l=(penwidth(10pt,z2r-z1r,0,s),0);
penstroke (z1e--z2e) slanted s; endfig;

beginfig(6);
w:=50pt; h:=50pt; s:=.25; z1r=(0,0); z2l=(w,h);
z1r-z1l=z2r-z2l
  =(penwidthover(1,-1,10pt,w,h,s),0);
penstroke (z1e--z2e) slanted s; endfig;

beginfig(7);
w:=50pt; h:=50pt; s:=.25; z1r=(0,h); z4l=(w,h);
z3r=z2r; y2r=0; x4r-x2r=x2r-x1r;
z3r-z3l=z4r-z4l=(penwidthover(2,-1,10pt,w,h,s),0);
penstroke (z3e--z4e) slanted s;
z1r-z1l=z2r-z2l=(penwidth(10pt,z2r-z1r,0,s),0);
penstroke (z1e--z2e) slanted s; endfig;

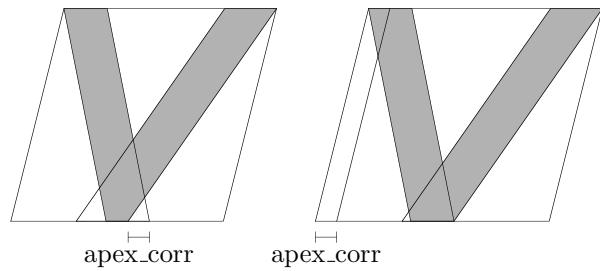
beginfig(8); w:=50pt; h:=50pt; s:=.25;
.5[z1l,z1r]=(0,h); .5[z4l,z4r]=(w,h); z3r=z2r;
y2l=y2r=y3l=h; y1l=y4l=h; x4r-x2r=x2r-x1r;
(x1r-x1l,x4r-x4l)=(x2r-x2l,x3r-x3l)
  =poswidthhalf(-1,10pt,10pt,w,h,s);
penstroke (z1e--z2e) slanted s;
penstroke (z3e--z4e) slanted s; endfig;

beginfig(9); w:=50pt; h:=50pt; s:=.25;
.5[z1l,z1r]=(0,0); .5[z4l,z4r]=(w,0); z3r=z2r;
y2l=y2r=y3l=h; y1l=y4l=0; x4r-x2r=x2r-x1r;
(x1r-x1l,x4r-x4l)=(x2r-x2l,x3r-x3l)
  =poswidthhalf(1,10pt,10pt,w,h,s);
penstroke (z1e--z2e) slanted s;
penstroke (z3e--z4e) slanted s; endfig;
```

Note: If you use `mfplain.mp` or METAFONT, you do not need to write `slanted s` every time, as this can easily be solved globally.

4 Apex correction

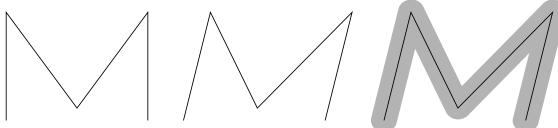
Of course, you generally do not want to join two chained diagonals directly, but you want them to overlap by an amount `apex_corr` as depicted below on the left:



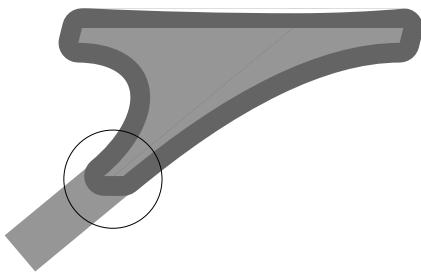
This does not require any new calculation formulae. For calculations, one just needs to add the `apex_corr` to the box width. In the METAFONT sources of *Computer Modern* (Knuth, 2014), the same trick is applied.

5 Drawn outline borders

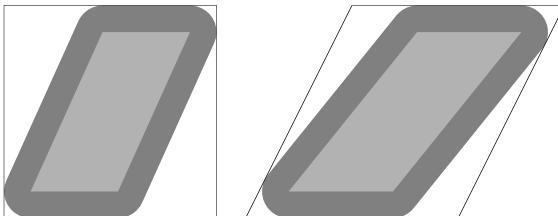
Drawn lines do normally not need slanting corrections, because the paths are slanted first and drawn in the end:



In the italic faces of *Computer Modern*, serifs are not only filled but also stroked. However, the joining stems are sometimes filled only. This creates bumps in letters like *A*, *K* and *X* (Jackowski, Ludwichowski, and Strzelczyk, 2009).



How can we avoid this effect? The formulae presented in section 3 are still valid for an additional circular border pen of width p_b if we use clever substitutions.

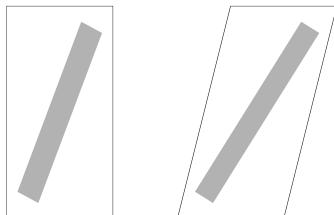


The METAPOST code for the upper right figure is indicated below:

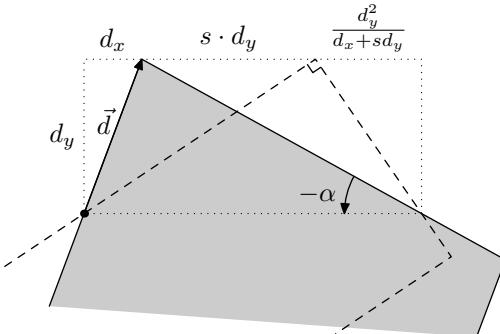
```
beginfig(10); w:=80pt; h:=80pt; s:=.5; pb:=20pt;
pickup pencircle scaled pb slanted s;
lft x1l=0; bot y1l=0; rt x2r=w; top y2r=h;
z1r-z1l=z2r-z2l
=(penwidthin(1,1,50pt-pb,w-pb,h-pb,s),0);
pickup pencircle scaled pb;
filldraw z1e--z2e slanted s; endfig;
```

6 Orthogonally cut slanted stems

For this section, the pen angle of the stem shall be exactly orthogonal to the stem direction \vec{d} after slanting.



We want to find the pen angle α before slanting.



In the preceding figure we find

$$\cot(-\alpha) = \left(sd_y + \frac{d_y^2}{d_x + sd_y} \right) : d_y = s + \frac{d_y}{d_x + sd_y},$$

as long as $d_x + sd_y \neq 0$. If $d_x + sd_y = 0$, the stem is vertical after slanting and we have the trivial solution $\alpha = 0$. Therefore,

$$\alpha = \begin{cases} \text{angle} \left(\begin{array}{c} s + \frac{d_y}{d_x + sd_y} \\ -1 \end{array} \right) & \text{if } d_x + sd_y \neq 0 \\ 0 & \text{otherwise.} \end{cases} \quad (10)$$

Equation (10) as a METAFONT/METAPOST macro:

```
def angleortho(expr d,s) =
  if xpart(d)+s*ypart(d)>0:
    angle(s+ypart(d)/(xpart(d)+s*ypart(d)), -1)
  else: 0 fi enddef;
```

References

- Jackowski, Bogusław, J. B. Ludwichowski, and P. Strzelczyk. “Math fonts: notes from the trenches”. ntg.nl/EuroTeX/2009/slides/jacko-slides.pdf, 2009.
- Jackowski, Bogusław, J. Nowacki, and P. Strzelczyk. “Antykwia Półtawskiego: a parameterized outline font”. *MAPS* **25**, 86–102, 2000. ntg.nl/maps/25/13.pdf.
- Knuth, Donald E. “The base file for Computer Modern”. ctan.org/tex-archive/fonts/cm/mf/cmbase.mf, 2002.
- Knuth, Donald E. “Computer Modern Roman upper case”. ctan.org/tex-archive/fonts/cm/mf/romanu.mf, 2014.

◊ Linus Romer
Oberseestrasse 7
Schmerikon, 8716
Switzerland
[linus.romer\(at\)gmx.ch](mailto:linus.romer(at)gmx.ch)

GUST e-foundry font projects

Bogusław Jackowski, Piotr Strzelczyk and
Piotr Pianowski

What is a document? It is a sequence of rectangles containing a collection of graphic elements.

What is a font? It is a sequence of rectangles containing a collection of graphic elements.

— Marek Ryćko

1 Introduction

The Polish TeX Users Group (GUST) has paid attention to the issue of the fonts since the beginning of its existence. In a way, it was a must, because the repertoire of the diacritical characters of the *Computer Modern* family of fonts (CM), “canonical” TeX family defined as the Metafont programs (see [5]), turned out to be insufficient for the Polish language. The efforts of the GUST font team (GUST e-foundry), led by Bogusław Jackowski, were kindly acknowledged by the professor Donald E. Knuth:

*I have been very pleased to see
so much excellent Polish work on TeX
for many years — and I humbly
beg to apologize for omitting the agonek
from my first Computer Modern fonts!*
Donald Knuth July 95

Obviously, our first fonts were PK bitmap fonts programmed using Metafont. Alas, the TeX/Metafont bitmap font format never became the world-wide standard. Therefore, the next step were fonts in the PostScript Type 1 format which fairly soon became obsolescent and was replaced by the OpenType format (OTF, a joint enterprise of Microsoft and Adobe, 1996, see [31]) which is actually a common container for the Adobe PostScript Type 1 and Microsoft TrueType (TTF) formats. In 2007, Microsoft extended the OTF standard with the capability of typesetting math formulas, largely based on ideas developed for TeX, and implemented it in MS Office. Soon, TeX engines were adapted to process such math OTF fonts. Therefore our recent fonts are released in the OpenType format, which also makes them easily usable outside of the TeX realm.

So far, no OTF successor is in sight, which is both good and bad (cf. Section 7.4).

We published our partial results successively as the work progressed. This paper provides an overall summary of our work: it describes the collections of fonts prepared by the GUST e-foundry, deals with some technical issues related to the generation of

fonts and their structure and puts forward a few proposals concerning future works.

This is not an overly strict report, but rather a story about our technical work on fonts, illustrated by representative examples which, we hope, show the essence of the matter. In order to keep our narration smooth, we decided not to use formal captions with explanations to figures and tables (only numbers of figures and tables are given). The relevant detailed descriptions always appear in the main text.

2 Historical background

PostScript and TeX are genetically related: their common ancestor is the ingenious idea of a programming language for the description of documents understood as a sequence of rectangular pages filled with letters and graphics. Both projects were devised nearly at the same time—at the turn of the 1970s to the 1980s.¹ And both are still alive and well, proving that the idea behind both projects was indeed brilliant.

From our perspective, the most important thing in common, and at the same time a key distinctive element, was the different handling of fonts and graphics; in other words, both systems clearly distinguished illustrations from fonts. That approach was justified by the computer technology at that time.

For both TeX and PostScript, fonts were external entities, both used metric files plus files defining glyph shapes, both defined contours as Bézier splines (planar polynomials of the 3rd degree), and for both fonts were to be prepared separately with dedicated font programs, prior to creating documents.

And this exhausts the list of similarities.

TeX worked with binary metric files, TFM; its output, a device independent file (DVI), was processed by so-called drivers which made use of the “proper” fonts, that is, the relevant bitmap collections, and produced output that could be sent to a printer or to a screen. The bitmaps were prepared independently with the Metafont program(s) which interpreted scripts written in the Metafont language and generated TFM and bitmap files.

In Metafont, the shapes of glyphs are defined as Bézier curves, stroked with a “pen” and/or filled.

Basic PostScript fonts (i.e., Type 1; see [28]) employ contours defined as non-intersecting closed Bézier outlines which can only be filled.² The “filled

¹ Formally, TeX was released a little earlier—TeX in 1982, PostScript in 1984, both with earlier work.

² The PostScript Type 1 documentation [28], p. 34, mentions the possibility of stroking: *a Type 1 font program can also be stroked along its outline when the user changes the PaintType entry in the font dictionary to 2. In this case,*

outline” paradigm relates also to the Microsoft TTF format and, thereby, to the OTF format.

PostScript Type1 fonts are usually (but not necessarily) accompanied by corresponding ASCII metric files (AFM), not used by the interpreters of the PostScript language. In the Microsoft Windows operating system, making an already complex situation even more complex, binary printer font metric files (PFM) were introduced for Windows drivers that used PostScript Type1 fonts.

For a long time, only commercial programs for generating PostScript Type 1 fonts were available. Only in 2001, George Williams released his remarkable FontForge program (initially dubbed PfaEdit; [25]). FontForge can generate outline fonts in many formats, including PostScript Type 1 and OTF.

PostScript was promptly (and rightly) hailed as the standard for printers and, more importantly, for phototypesetters, therefore a driver converting DVI files to PostScript became necessary. Fortunately for *TeXies*, PostScript is equipped also with Type 3 fonts; glyphs in Type 3 fonts can be represented by nearly arbitrary graphic objects, in particular, by bitmaps, therefore the making of a PostScript driver for converting DVI files to PostScript was possible already in 1986, when Dvips, the first and still most popular driver was released by Tomas Rokicki. (It’s a pity that the idea of Type 3 fonts was not supported and developed by Adobe.)

There were a few unsuccessful attempts to convert the basic *TeX* font collection, CM, to the PostScript Type1 format automatically, thus preserving the parameterization. The main hindrance was the excessive usage of stroked (both painted and erased) elements in the CM font programs, while, as was mentioned previously, the PostScript Type 1 and OTF formats accept only filled shapes.

The “filled outline” paradigm was a convenient optimization at the beginning of the computer typesetting era, when, for example, the generating of the complete collection of bitmaps for the CM fonts at resolution, say, 240 dots per inch (typical for dot matrix printers) took a few days. Nowadays, the paradigm still thrives by virtue of tradition: there is an abundance of such fonts and, and what is worse, all operating systems support only this kind of font.

3 First steps

Taking the above into account, we made up our minds to design our own programmable system for generating fonts in “world-compatible” formats.

overlapping subpaths will be visible in the output; this yields undesirable visual results in outlined characters. In practice, this possibility is not used.

3.1 Our tools

Our primary tool was MetaPost [4], a successor of Metafont, which promised well as a tool for making PostScript Type 1 fonts due to its native PostScript output. We called our MetaPost-based package MetaType 1 [13]. It was instantiated as a set of scripts using, besides MetaPost, T1utils, that is, Lee Hetherington’s (dis)assembler for PostScript Type 1 fonts (cf. [3, 4]). A few scripts written in Gawk and Perl were also employed.

On the one hand, such a simple approach turned out to be insufficient for generating OTF fonts, in particular OTF math fonts. On the other hand, it turned out to be flexible enough to include an extra external step for making OTF fonts. For text fonts, we employed the *Adobe Font Development Kit for OpenType* (AFDKO [26]); for math fonts, the FontForge library governed by Python scripts [15, 25]. In the future, we want replace AFDKO by a FontForge+Python utility (cf. Section 7).

A set of MetaPost macros in the MetaType 1 package defines two important procedures, essential for generating non-intersecting outlines and heavily used in our font programs: finding a common outline for overlapping figures, known also as removing overlaps, and finding the outline of a pen stroke, known as expanding strokes or finding the pen envelope.

Another important feature, hinting, is implemented, but, in the end, we decided to avoid manual hinting, since it is difficult to control and yields mediocre results. Metafont has no notion of hinting—the Metafont language simply offers rounding. Moreover, the language for describing outlines in PostScript Type 1 fonts cannot express even as trivial a mathematical operation as rounding.

For low-resolution devices, controlled rounding is crucial—hence the idea of “hinting”, that is, controlled rounding. Alas, hinting algorithms remain undisclosed, especially with regard to commercial typesetting devices such as phototypesetters. One can presume, however, that low-resolution devices are bound to disappear sooner rather than later: the resolution of display devices has reached almost 600 dpi and 1200 dpi (and more) for printers is nowadays nothing special. Therefore, running with the hare and hunting with the hounds, we decided to hint our recently released OTF fonts automatically with FontForge.

3.2 Trying our tools out

We tested our newborn MetaType 1 engine against a simple example, namely, Donald E. Knuth’s logo

font [17]: the sources, originally written in the Metafont language, were adapted to MetaType1's requirements. The distributed package contains both MetaType 1 sources and the resulting PostScript Type 1 files for the *logo* font [13].

The test proved the usefulness of the approach; hence, in 1999, we started a larger project: the programming of the long-established Polish typeface *Antykwa Półtawskiego* as a parameterized font. The preliminary family of fonts was released in 2000. Ten years later, prompted by the TeX community, we released the enhanced version of *Antykwa Półtawskiego* with the relevant OTF files [7]. In parallel, Janusz M. Nowacki used MetaType 1 to generate several replicas of Polish fonts, namely, *Antykwa Toruńska*, *Kurier*, *Iwona*, and *Cyklop* [22].

4 Latin Modern collection of fonts

Recall that the repertoire of diacritical characters in CM fonts was insufficient for most languages using diacritical characters. The TeX accenting mechanism (the \accent primitive), meant as a solution to this problem, was unsatisfactory—for example, accents and hyphenation conflicted. The problem was recognized relatively early and various approaches were used to remedy the situation.

For example, the Polish extension of CM in the PK format was prepared in Poland (PL fonts, [8]), but this worked only for Polish TeXies.

Also worthy of note is the *European Computer Modern Fonts* (EC) project led by Jörg Knappen and Norbert Schwarz, triggered during the TeX Users Group conference in Cork, 1990, and finished in 1997 [16]. The EC metric files, however, are slightly incompatible with CM metric files, by circa 0.025%.³

A rather general technique was applied by Lars Engebretsen, who attempted to eliminate the necessity of using the \accent primitive by making virtual fonts, dubbed *Almost European* (AE), containing quite a large set of the European diacritical characters [1].

Hyphenation worked with AE fonts, though still unsatisfactorily—for example, coinciding of such accents as cedilla or ogonek with a main glyph is inconvenient when a non-intersecting outline is required (for cutting plotters, for outlined titles, and so on). Moreover, the virtual fonts are obviously unusable outside the TeX realm.

³ The reason behind this discrepancy is a peculiarity of Metafont arithmetic: the formula $1/36 * i$ yields different results than the formulas $i/36$ and $1/36i$; for example, for $i = 3600$ the results are 99.97559 for the first formula and 100 for the latter formulas. The first formula was used in the EC fonts (in the `gdef` macro).

4.1 Latin Modern fonts in the PostScript Type1 format

In 2002, during the EuroBachotEX meeting, a proposal of converting Engebretsen's AE fonts into the PostScript Type 1 format and augmenting with them the set of necessary diacritical characters was put forward by representatives of European TeX user groups. We had no choice but to accept the proposal with delight.

Our initial plan was to use the AE fonts as our departure point; we even wanted to preserve the original name, *Almost European*, coined by Lars Engebretsen. It turned out, however, to be much more efficient to prepare the enhanced version of the CM fonts from scratch, and so sticking to Lars Engebretsen's name seemed inadequate, because the differences were too essential.

All in all, inspired by both EC and AE fonts, we came up with the *Latin Modern* (LM; see [11]) project which was accepted by the user groups.

Fortunately for us, freely available quality CM fonts in the PostScript Type 1 format already existed. In the 1980s and 90s, they were produced (from traced bitmaps improved by very solicitous manual tuning) for commercial purposes by Blue Sky Research and Y&Y. Nearly a decade later, they were released to the public thanks to the efforts of the American Mathematical Society.⁴

We converted the PostScript Type 1 files of the CM fonts to MetaType 1 and wrote the MetaPost software relevant for generating the characters we decided to add (mainly diacritical letters). The work had already been partially done by Janusz M. Nowacki, who prepared the PostScript Type 1 version of the PL fonts in 1997. The official version of the LM fonts, 1.000, was eventually released in 2006 (in the meantime, several unofficial versions were released for testing purposes). The LM collection of fonts consisted of 72 text fonts, each counting about 700 glyphs, plus 20 CM-like math fonts.

In 2009, an extensive revision of the LM fonts was carried out: the text fonts now contain more than 800 glyphs each (altogether more than 60,000 glyphs) and the glyphs conform to the changes introduced by Donald E. Knuth in 1992.

⁴ In 1997, a consortium of scientific publishers (American Mathematical Society, Elsevier Science, IBM Corporation, Society for Industrial and Applied Mathematics, and Springer-Verlag) in cooperation with Blue Sky Research and Y&Y decided to release these excellent fonts non-commercially; in order to assure the authenticity of the fonts, copyright was assigned to the American Mathematical Society. (<http://www.ams.org/publications/type1-fonts>).

Almost all of the CM text fonts have counterparts in the LM family; the exceptions are one monospaced font, `cmtex10`, emulating Donald E. Knuth's keyboard layout, and the rarely used `cmff10`, `cmfi10`, `cmfib8`, and `cminch`. So far, nobody has complained about this inconsistency. Instead, encouraged by Hans Hagen, we decided to create 10 variants of typewriter LM fonts not having counterparts in the CM family: `lmtlc10`, `lmtk10`, `lmtl10`, `lmvtk10`, and `lmvtl10` (monospaced light condensed and monospaced and variable-width dark and light, respectively) plus their oblique variants `lmtko10`, `lmtlo10`, `lmtlc10`, `lmvtko10`, and `lmvtlo10`.

4.2 Latin Modern fonts in the OTF format

It was relatively easy to prepare the LM family of fonts in the OTF format using the AFDKO package: it mainly necessitated preparing a few extra data files in the *OpenType Feature File Specification* language [32]. Needless to say, the experience gathered at this stage came in handy during the work on the TG fonts (see Section 5).

There was trouble, however, with the 20 math fonts. We provided the respective LM equivalents in PostScript Type 1 format. For compatibility with the (obsolete) PL fonts, the symbol fonts, `1msy*` and `1mbsy*`, contain two extra glyphs: slanted greater-or-equal and less-or-equal signs, used traditionally in Polish math typography. As the math extension for OpenType did not exist yet, we decided not to convert these fonts to OTF. We knew that the companies that had invented and maintained the OTF standard, in cooperation with the American Mathematical Society and the Unicode Consortium, were working on extending the standard with math typesetting capabilities. We expected that by using the enhanced OTF specification we would be able to create a T_EX-compatible math OTF collection. Alas, the Unicode Consortium report on Unicode support for mathematics [37], followed by the initially confidential Microsoft specification [29], snuffed out our hopes. It turned out that OTF math and T_EX math cannot be reconciled. More information on the interrelationships between OTF and T_EX math can be found in Ulrik Vieth's thorough analysis [24].

4.3 Repertoire issues

Our primary aim was to provide a repertoire of diacritical letters rich enough to cover all European languages. We thoroughly exploited Michael Everson's comprehensive study of European alphabets [2], as well as other sources. Several other languages using Latin-based alphabets, such as Vietnamese, Navajo and Pali, are covered.

Initially, contrary to the *Latin Modern* name, we considered including Cyrillic alphabets also. Having thought the matter over, we decided, with regret, to abandon this idea and concentrated our efforts on OTF math fonts.

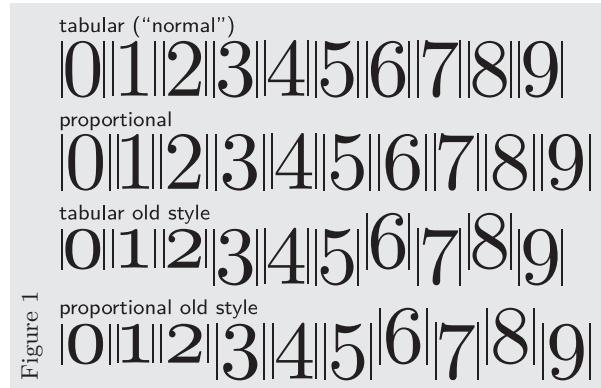
Besides diacritical characters, the Latin Modern fonts contain also a number of glyphs traditionally present in T_EX fonts, such as Greek symbols, currency symbols, technical symbols, etc. Detailed description of the contents of the fonts can be found in the document entitled *The Latin Modern Family of Fonts. Technical Documentation*, included in the LM distribution package [11].

Two groups of glyphs are widely used in typography but neglected to a certain extent in the CM fonts, namely, *caps and small caps* and *old style numerals*, also known as *text figures* or *nautical digits*; the latter name originates from their widespread use in tables in nautical almanacs at one time. For reasons hard to explain, the caps and small caps were implemented in the CM family as a separate font, while the old style numerals (upright!) are in the math italic font (`cmmi*`).

The LM fonts incorporated caps and small caps from the CM family, together with its width idiosyncrasy: the `1mcsc10` font, like `cmcsc10`, has capital letters wider than the `1mr10` font by circa 8%. There are two caps and small caps fonts in the LM collection, namely, the regular and typewriter specimen, `1mcsc10` and `1mtcsc10`, as in CM, plus their oblique variants, `1mcsc10` and `1mtcsc10`, absent from CM. In principle, small caps glyphs could be transferred to other fonts, but we decided to not alter the framing of the original CM family, more so as CM has no sans-serif caps and small caps; however, the problem of extending the LM family with bold counterparts of `1mcsc10` and `1mtcsc10` (and their oblique variants), raised repeatedly by CM/LM users, needs serious consideration.

Concerning old style numerals, we could not accept the CM oddity and included them in all text fonts of the LM family. Further, all numerals come in 2 'flavors': normal (fixed-width a.k.a. tabular) and proportional (variable-width, having balanced sidebearings) which altogether yields 4 variants—see Figure 1.

The TFM format contains only 256 slots for glyphs, thus, the whole repertoire of glyphs cannot be accessed at once if T_EX is used in a "traditional" way, that is, with TFM files. In particular, accessing the different kinds of numerals when using PostScript Type 1 fonts plus TFM metrics turns out to be clumsy; as a result, only tabular old style numerals are available in our package, in the TS1 encoding



(see below). On the one hand, OTF fonts seem more convenient as they do not impose such a restriction; for example, all numerals can be accessed by using the OTF feature mechanism, more precisely, by the features `onum`, `lnum`, `pnum`, and `tnum` [33]. On the other hand, OTF metric data cannot, in general, be fully compatible with TFM metric data because the glyph widths in OTF fonts must be represented by integer quantities. This should be considered a drawback by TeXies—see Section 4.4.

Following the L^AT_EX tradition, we provided several encodings for the LM fonts, namely:

- ◊ CS (CS TUG) encoding (`cs-* .tfm`),
- ◊ EC (Cork) encoding (`ec-* .tfm`),
- ◊ L7X (Lithuanian) encoding (`l7x-* .tfm`),
- ◊ QX (GUST) encoding (`qx-* .tfm`),
- ◊ RM (“regular math”, used in OT1 and OT4) encodings (`rm-* .tfm`),
- ◊ Y&Y’s T_EX’n’ANSI a.k.a. LY1 encoding (`texnansi-* .tfm`),
- ◊ T5 (Vietnamese) encoding (`t5-* .tfm`),
- ◊ Text Companion for EC fonts a.k.a. TS1 (`ts1-* .tfm`).

The L^AT_EX support for all these encodings, due to Marcin Woliński, is also part of the LM distribution.

TFM files nominally representing the same encoding do not always define the same set of characters; for example, the character sets of `cmr10.tfm` and `cmtt10.tfm` differ. The original CM fonts comprise 7 different character sets, with an idiosyncratic difference between the `cmr10` and `cmr5` layouts. As a remnant of the CM design, there are 5 different character sets of the LM text fonts:

1. 821 glyphs (basic set): `lmb10 lmb010 lmbx10 lmbx12 lmbx5 lmbx6 lmbx7 lmbx8 lmbx9 lmbxi10 lmbxo10 lmdunh10 lmduno10 lmr10 lmr12 lmr17 lmr5 lmr6 lmr7 lmr8 lmr9 lmri10 lmri12 lmri7 lmri8 lmri9 lmro10`

- `lmro12 lmro17 lmro8 lmro9 lmss10 lmss12 lmss17 lmss8 lmss9 lmssbo10 lmssbx10 lmssdc10 lmssdo10 lmss10 lmss12 lmss17 lmss8 lmss9 lmu10 lmvtk10 lmvtko10 lmvtl10 lmvtlo10 lmvtt10 lmvtto10`
- 2. 824 glyphs: `lmssq8 lmssqbo8 lmssqbx8 lmssqo8`
extra characters: `varI varIJ varLogonek`
- 3. 814 glyphs: `lmcsc10 lmcsc010`
missing characters: `f_k ff ffi ffl fi fl longs`
- 4. 785 glyphs: `lmtk10 lmtko10 lmtl10 lmtlc10 lmtlco10 lmtlo10 lmtt10 lmtt12 lmtt8 lmtt9 lmtti10 lmtt10`
missing characters: `f_k ff ffi ffl fi fl Germandbls hyphen.prop IJ ij permyriad servicemark suppress trademark varcopyright varregistered zero.oldstyle zero.prop one.oldstyle one.prop two.oldstyle two.prop three.oldstyle three.prop four.oldstyle four.prop five.oldstyle five.prop six.oldstyle six.prop seven.oldstyle seven.prop eight.oldstyle eight.prop nine.oldstyle nine.prop`
- 5. 784 glyphs: `lmtcsc10 lmtcs010`
missing characters: as in 4, also `longs`

4.4 Compatibility issues

We did our best to provide outline fonts that can be used as a replacement for CM fonts. To a certain extent, we managed to achieve this goal, namely, the PostScript drivers which process TeX documents typeset with CM metric files, can use either CM or LM PostScript Type 1 fonts—special map files for PostScript Type 1 fonts are available for this purpose. The metric files, however, cannot be used replaceably, because the typesetting algorithms are intrinsically unstable—even tiny (rounding) errors may yield glaringly different results.

Therefore, LM users also cannot expect PostScript Type 1 and OTF fonts to be used replaceably. Recall that the OTF format requires integer number representation for glyph widths. The “reference” quantity is the *em* unit: 1 *em* = 2048 units for fonts using splines of the 2nd degree, 1 *em* = 1000 units for fonts using splines of the 3rd degree (e.g., our LM and TG fonts). Therefore, in our case, the difference in width is on average 1/2000 *em* (twice as large as the variation in the EC widths), that is, circa 0.005 *pt* for 10-point fonts.

Because the MetaType1 sources of the LM fonts are the result of conversion from PostScript Type 1,

the widths stored in the LM TFM files are not identical to the respective original CM widths. They are closer, however, to the original quantities by an order of magnitude compared to the EC and OTF widths. At the cost of great effort (by referring to the Metafont sources), we might have eliminated rounding errors in LM widths. But it would not cure the problem of (non-)replaceability, as widths are not the only source of trouble. Differences in heights and depths of glyphs may also yield unexpected behavior of the TeX typesetting algorithm.

The problem of heights and depths in TeX turns out to be unavoidable, and quite serious: the TFM format permits by design only 16 different heights and depths, including the obligatory entries containing the value zero. If there are in fact more heights and depths in a given font, their number is cleverly reduced to 16 by Metafont (as well as by the MetaPost and TFMtoPL programs). One of the certainly unwanted results is that the same glyph in different encodings may have different heights and/or depths! For example, the height of the letter ‘A’ is 6.88875 pt in the `rm-lmr10.tfm` file (this layout is an extension to 256 slots of the `cmr10` layout), it is 6.99648 pt in the `t5-lmr10.tfm` file (Vietnamese layout), while in the canonical `cmr10.tfm` file it is 6.83331 pt.

This is not the end of the list of possible sources of incompatibility between CM and LM fonts. Positioning of the accents is also a long story. These and related aspects are explained minutely in [12].

Finally, let us consider a somewhat atypical example of incompatibility between the LM and CM fonts related to Donald E. Knuth’s mistake in a CM `ligtable` program, uncorrectable for obvious reasons but basically harmless; namely, `roman.mf` contains the following:

```
% three degrees of kerning
k#:=-.5u#; kk#:=-1.5u#; kkk#:=-2u#;
ligtable "k":
  if serifs: "v": "a" kern -u#, fi
  "w": "e" kern k#, "a" kern k#,
  "o" kern k#, "c" kern k#;
```

The culprit is the `if serifs` clause: the kern pair ‘ka’ appears twice in the TFM files of serif fonts with the values $-u\#$ and $-0.5u\#$, respectively, as is easily seen in the following fragment of the `cmr10.pl` file (the respective lines are marked with arrows):

```
(CHARACTER C k
  (CHARWD R 0.527781)
  (CHARHT R 0.694445)
  (COMMENT
    (KRN C a R -0.055555)←
    (KRN C e R -0.027779)←
    (KRN C a R -.027779)←
    (KRN C o R -0.027779)
```

(KRN C c R -0.027779)

)

Moreover, there are no ‘va’, ‘vc’, ‘ve’, and ‘vo’ kern pairs in sans-serif fonts, although there are ‘kc’, ‘ka’, ‘ke’, ‘ko’, ‘wa’, ‘wc’, ‘we’, and ‘wo’ kern pairs in these fonts. We could not see the reason for ignoring ‘v’ in this context, thus we decided to add the relevant kern pairs in the LM fonts; we also added quite a few other kern pairs missing, in our opinion, from the CM fonts, for example, ‘eV’ and ‘kV’.

Summing up, we believed that we had good reasons for giving up the struggle for a “100-percent compatibility” between LM and CM metrics, whatever that would mean, and to confine ourselves to providing the mentioned replaceability of outlines.

5 The TeX Gyre collection of fonts

Heartened by the results of the LM enterprise, we accepted without hesitation the next proposal: the “LMization” of the family of fonts provided by Ghostscript as a replacement for the renowned *Adobe base 35* fonts, generously released by the URW++ company under free software licenses.

- ◊ ITC Avant Garde Gothic (book, book oblique, demi, demi oblique)
- ◊ ITC Bookman (light, light italic, demi, demi italic)
- ◊ Courier (regular, regular oblique, bold, bold oblique)
- ◊ Helvetica (medium, medium oblique, bold, bold oblique)
- ◊ Helvetica Condensed (medium, medium oblique, bold, bold oblique)
- ◊ New Century Schoolbook (roman, roman italic, bold, bold italic)
- ◊ Palatino (regular, regular italic, bold, bold italic)
- ◊ Symbol
- ◊ Times (regular, regular italic, bold, bold italic)
- ◊ ITC Zapf Chancery (medium italic)
- ◊ ITC Zapf Dingbats

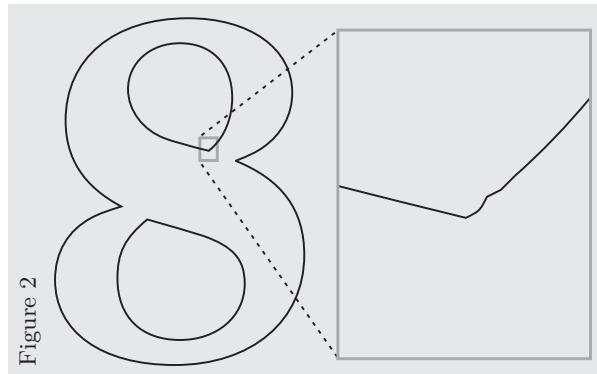
Since our aim was “LMization”, we excluded the Symbol and ITC Zapf Dingbats non-text fonts from the scope of our interest.

After a brief (but heated) debate, the name of the project and of its constituent fonts were coined. The project was dubbed TeX Gyre (TG) and the following names were accepted (the respective file name kernels, original Adobe names and Ghostscript, that is, URW, names are given in parentheses):

- ◊ TG Adventor (`qag` / ITC Avant Garde Gothic / URW Gothic L)
- ◊ TG Gyre Bonum (`qbk` / ITC Bookman / URW Bookman L)
- ◊ TG Cursor (`qcr` / Courier / Nimbus Mono L)
- ◊ TG Heros (`qhv` / Helvetica / Nimbus Sans L)
- ◊ TG Heros Condensed (`qhvc` / Helvetica Condensed / Nimbus Sans L Condensed)
- ◊ TG Schola (`qcs` / New Century Schoolbook / Century Schoolbook L)
- ◊ TG Pagella (`qp1` / Palatino / URW Palladio L)
- ◊ TG Termes (`qtm` / Times / Nimbus Roman No9 L)
- ◊ TG Chorus (`qzc` / ITC Zapf Chancery / URW Chancery L)

We initially considered including Cyrillic alphabets, but, as in the case of the LM fonts, we eventually abandoned this idea, also with regret, the more so as the Ghostscript fonts at that time contained an (apparently unfinished) set of Cyrillic glyphs.

We expected that the main effort would be the making of extra glyphs plus maybe correcting outlines here and there. To our surprise, quite a few glyphs required tuning because of evident errors in outlines. One of the most striking examples is the glyph ‘eight’ from the URW Schoolbook bold font⁵ — see Figure 2.



Mostly, we removed redundant or wrong nodes (points) from the outline definitions, deleting more than 5% of them in all. In the case of TG Pagella, however, the insertion of extra nodes turned out necessary. The chart in Figure 3 shows some statistics for the upright TG fonts. The diagram concerns the version of the Ghostscript fonts which we used as our starting point. We used circa 350 glyphs from

⁵ Recently, the font has been renamed to ‘C059 bold’; the bug was removed from the Ghostscript distribution only in 2015, although the TEX Collection 2016 distribution still contains (due to the legacy reasons) the faulty glyph.

each font. The total number of nodes in these glyphs varied from circa 10,000 to 25,000 per font. In the current release, the TG text fonts count almost 1100 glyphs each with the number of nodes varying from circa 30,000 to 65,000 (for sans-serif and serif italic variants, respectively; see [14]).

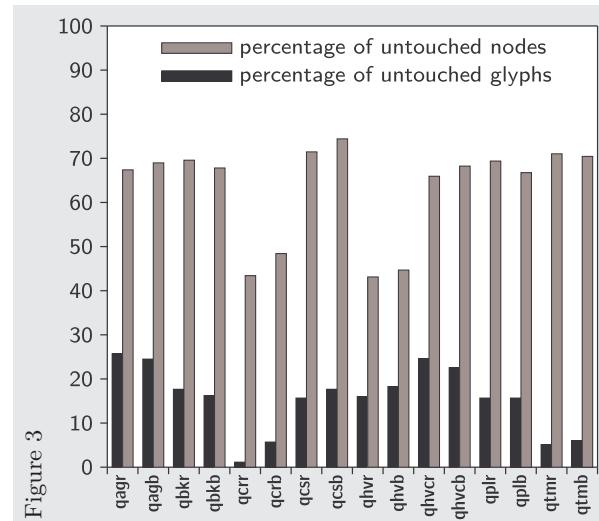


Figure 3

5.1 Repertoire issues

The difference in the number of glyphs between the TG and LM text fonts (the former having circa 250 glyphs more per font) is due mainly to the presence of small caps in the TG fonts (225 glyphs per font). Also unlike the LM fonts, each TG font contains the complete Greek alphabet and a few technical glyphs, such as ‘lozenge’ and ‘lscript’.⁶ Except for these, the LM and TG fonts share the same repertoire of glyphs and the same set of TFM encodings (see Section 4.3).

The LATEX support for these encodings was also provided by Marcin Woliński.

5.2 Compatibility issues

The consistency of the widths of the original Adobe and the respective TG glyphs was one of our main concerns, as the TG fonts were meant as potential replacements for the Adobe fonts. It turned out, however, that the original font metric files [27], contained apparent metric flaws which we decided, with some hesitation, not to retain.

A typical example (concerning Helvetica, a.k.a. Nimbus Sans L, a.k.a. TG Heros) is depicted in Figure 4. Both Spanish ‘j’ and Scandinavian ‘ø’ glyphs belong to the Adobe Standard Encoding set, hence

⁶ For historical reasons, the LM fonts contain a few additional variants of the base, left, and right double quotes, absent from the TG fonts.

one can expect that they should be considered important and thus unchangeable. Nevertheless, we could not see a reason for using widths different from the ‘!’ and ‘o’ widths, respectively, and certainly there is no substantiation for the asymmetry of sidebearings, especially in the case of ‘ø’; therefore, we decided to alter the metrics.

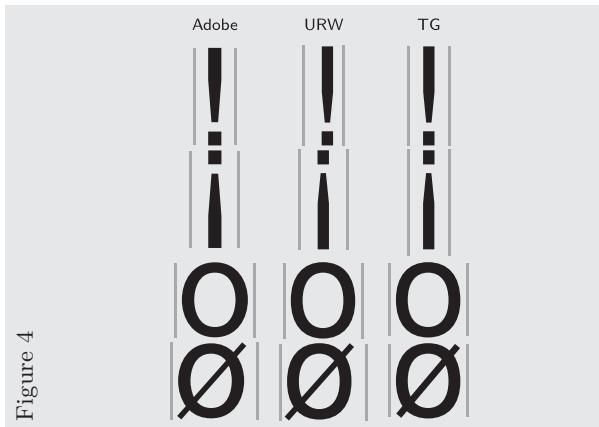


Figure 4

Fortunately, there are few such cases in the TG collection; each is mentioned in the documentation of the TG fonts [14].

Because the original widths for the TG fonts, unlike in the LM collection, were integer numbers, there is no metric discrepancy between the PostScript Type1 and OTF font formats, as far as widths are concerned. Heights and depths, however, are subject to the same restrictions as discussed in Section 4.4.

6 OTF math fonts

The chronic problem of lack of math support for the TG collection became the impetus for our third venture: math fonts for the LM and TG collections in the OTF format. The recent update of the LM and TG fonts took place at the end of 2009. The math extension for the OTF format ([29]) had been released and there existed the FontForge font editor ([25]) capable of generating such fonts. So we embarked upon an expedition into unknown regions—since then we have focused our attention on the work on OTF math fonts, again with the benevolent encouragement and support from the T_EX users groups.

As we should have expected, the task turned out interesting and absorbing, and, according to Hofstadter’s Law,⁷ we spent more time on it than we expected. From the very beginning, we aimed at making a collection of mutually consistent math OTF

⁷ Hofstadter’s Law: *It always takes longer than you expect, even when you take into account Hofstadter’s Law*—Douglas R. Hofstadter.

fonts and we underestimated the heterogeneity of the sources of additional alphabets and the problem of interrelationships—works on subsequent fonts entailed moving backwards to the fonts which we had prematurely considered ready. Nevertheless, in 2011, we happily announced the release of our first math OTF font, namely, Latin Modern Math. Altogether, six math fonts have been released by the GUST e-foundry so far [9, 10]:

- ◊ TG Latin Modern Math
- ◊ TG Bonum Math
- ◊ TG Schola Math
- ◊ TG Pagella Math
- ◊ TG Termes Math
- ◊ TG DejaVu Math

This amounts to nearly half of all OTF math fonts released in the world. Besides these, the following OTF math fonts have been released: Asana by Apostolos Syropoulos, Neo-Euler and XITS by Khaled Hosny, STIX by the STI Pub companies,⁸ Cambria Math by Microsoft,⁹ Lucida Math by Bigelow & Holmes, and Minion Math by Johannes Küster; the latter three fonts are distributed commercially.

6.1 OTF math font contents

Math OTF fonts, as we expounded in [10], are truly nasty beasts. In accordance with [35] and [37], they are expected to contain a plethora of glyphs: letters, arrows, math operators and delimiters, geometrical shapes, technical symbols, etc. The presence of some of them, particularly the (over)abundance of peculiar geometrical shapes and arrows, is hard to substantiate in our opinion.

Initially, we planned also releasing the math companion to the T_EX Gyre sans-serif fonts, TG Advenir and TG Heros, but the Unicode specification for the contents of math fonts, [37], turned out definitely “serif-oriented”. Let us take, for example, the arrangement of the LM Math font shown in Table 1: following the cited specification, we combined several LM source fonts into a single complex font. As the table clearly shows, the basic subsets, that is, plain, bold, italic and bold italic are assumed to consist of serif glyphs by default. It is not obvious how

⁸ The STIX project began through the joint efforts of American Mathematical Society (AMS), American Institute of Physics Publishing (AIP), American Physical Society (APS), American Chemical Society (ACS), Institute of Electrical and Electronic Engineers (IEEE), and Elsevier Science; these companies are collectively known as the STI Pub companies.

⁹ Cambria Math was the first math font published, conforming to the specification *MATH — The mathematical typesetting table* [29]. It was released by Microsoft in 2007, along with a MS Office version equipped with the capability of handling the math font and editing math formulas.

the table should be adjusted to suit sans-serif math fonts. Work on this issue is in progress.

category	charset	source fonts
plain (upright, serif)	L*, G, D	lmr, lmmi (upright)
bold	L, G, D	lmbx, lmmib (upright)
italic	L, G	lmmi
bold italic	L, G	lmmib
sans-serif	L, D	lmss
sans-serif bold	L, G, D	lmssbx
sans-serif italic	L	lmssso
sans-serif bold italic	L, G	lmssbo
calligraphic	L	eusm (slanted)
bold calligraphic	L	eusb (slanted)
Fraktur	L	eufm
bold Fraktur	L	eufb
double-struck	L, D	bbold (by Alan Jeffrey)
monospace	L, D	lmtt

L, G, D — Latin, Greek and digits, respectively
L* — contains also diacritical letters and punctuation

Table 1
All the alphanumeric glyphs specified in the table, except for the “plain” ones (first row), are given special mathematical Unicode slots — see [37]

The original CM fonts, and thus the LM fonts, do not contain the complete sans-serif Greek. We generated the missing glyphs using modified Metafont sources in order to generate outlines instead of bitmaps and tuning the result manually, if required.

Moreover, a math font is bound to contain many other characters, most notably glyphs used for subscripts of the 1st and 2nd order, used also for superscripts; we’ll refer to them shortly *pars pro toto* as subscripts. They are accessed by the OTF feature mechanism, more precisely by the math extension feature *ssty* [33].

Extensible symbols, like large brackets or radicals, are another important group of math-oriented glyphs. An extensible symbol consists of a collection of a few components (the left part of Figure 5) assembled by the typesetting engine into a seemingly single character (the right part of Figure 5).

First, a glyph of adequate size is searched for in the so-called chain of glyph variants (here: the three leftmost curly braces). If a proper glyph is not found, the typesetting engine assembles a respectively large symbol from the relevant pieces using a fairly complex algorithm — everybody who has attempted unsuccessfully to fit braces around a formula according to one’s desire probably knows it.

In the case of the radical symbol, the situation is still more complex, because the OTF and T_EX geometric structure, as well as the relevant metric data of the components differ, as is shown in Figure 6 which visualises “stages” of the assembling of an extensible radical symbol.

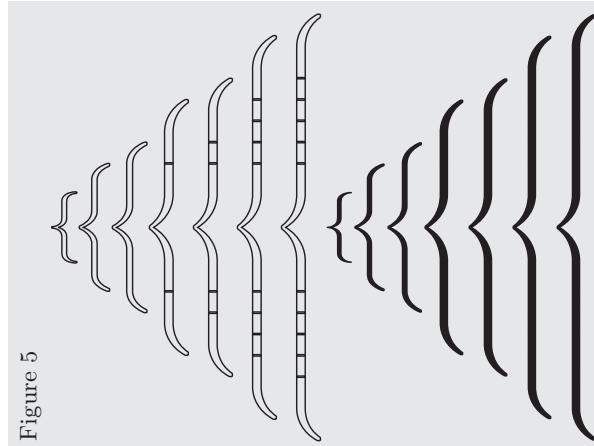


Figure 5

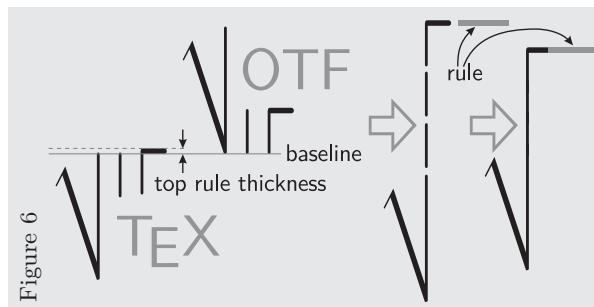


Figure 6

In both cases, the radical symbol is assembled from glyphs taken from a relevant font and a line (rule) drawn by the typesetting program at the top of the symbol (marked with a gray color). The thickness of the rule, however, is given explicitly as a parameter in math OTF files, while it is inferred from the height of the top element of the radical symbol in T_EX (recall the problem of the limited number of heights in a T_EX metric file).

There is an important difference between the T_EX and OTF math font specification concerning extensible glyphs: T_EX is equipped with the ability to assemble compound glyphs from pieces only vertically, while the OTF format offers both horizontal and vertical assembling. In T_EX (i.e., in the plain T_EX format), such glyphs as horizontal braces are defined by special macros using rules and a few glyphs from a relevant font:

```
\def\downbracefill
  {${\@m@th \setbox\z@\hbox{\$}\braceld\$}%
   \braceld\leaders\vrule height\ht\z@
   depth\z@\hfill\braceru
   \braceru\leaders\vrule height\ht\z@
   depth\z@\hfill\bracerd\$}
\def\upbracefill [...]
```

Incidentally, it seems that diagonal extensible glyphs have not yet been invented. Could it be that the conundrum is too difficult to solve?

More on the differences between the T_EX and OTF specifications of the structure of math fonts can be found in Ulrik Vieth's survey [24].

In the case of LM Math, we were fortunate to have well-known clean sources for a base, already containing 7-point and 5-point variants, suitable for typesetting subscripts, and components for assembling extensible glyphs.

We have to confess, however, that we were not especially delighted with the Computer Modern calligraphic script. More pleasingly designed, to our eyes, are the calligraphic letters of the renowned Euler family. Therefore, we decided to transfer the glyphs from the Euler fonts (slanting them slightly) to the LM Math font:

$\mathcal{A}\mathcal{B}\mathcal{C}\mathcal{P}\mathcal{Q}\mathcal{T}\mathcal{A}\mathcal{B}\mathcal{C}\mathcal{P}\mathcal{Q}\mathcal{T}$	$\mathcal{A}\mathcal{B}\mathcal{C}\mathcal{P}\mathcal{Q}\mathcal{T}\mathcal{A}\mathcal{B}\mathcal{C}\mathcal{P}\mathcal{Q}\mathcal{T}$	$\mathcal{A}\mathcal{B}\mathcal{C}\mathcal{P}\mathcal{Q}\mathcal{T}\mathcal{A}\mathcal{B}\mathcal{C}\mathcal{P}\mathcal{Q}\mathcal{T}$
Computer Modern	Euler	Latin Modern Math

In the case of the TG math fonts, the situation was slightly worse. The basic sources, that is, the text fonts, were already improved by us, but the sources of the relevant additional character sets were highly heterogeneous. We used freely available fonts of the best possible quality as our base; nevertheless, much manual tuning was necessary (recall the tuning of the sources of the TG text fonts).

Moreover, not all suitable fonts had acceptable free licenses. In a few cases, we had to contact the authors personally. It should be emphasized that in all cases the authors, if we managed to reach them, courteously agreed to make their fonts available for our purposes. The following external fonts were used in the TG math fonts project (in alphabetical order):

- ◊ Lato by Łukasz Dziedzic (TG Schola Math)
- ◊ Kerkis by Apostolos Syropoulos and Antonis Tsolomitis (TG Bonum Math)
- ◊ Leipziger Fraktur replica by Peter Wiegel (TG Bonum Math and TG Termes Math)
- ◊ Math Pazo by Diego Puga (TG Pagella Math)
- ◊ Odstemplik by Grzegorz Luk (gluk) (TG Pagella Math)
- ◊ Theano Modern by Alexey Kryukov (TG Schola Math)

We are most grateful to all the authors for their prominent aid.

6.2 Visual issues

Observe that the same monospaced alphanumeric characters (excerpted from TG Cursor) are shared by TG Bonum Math, TG Schola Math, and TG Termes Math. In such cases one must be carefully check whether the size of the glyphs being included fits the

size of the basic set of glyphs. Nominally, all fonts (both in the PostScript Type 1 and OTF formats) have the same design size, 10 typographic points (recall that 1 point = 1/72 inch; cf. Section 4.4). Working on the TG math fonts, we had to adjust the size of the subsets a few times. For example, we enlarged the borrowed monospaced glyphs to circa 112.5% in TG Schola Math; otherwise the monospaced alphabet looked too small in combination with Schola's brawny glyphs.

The visual harmonizing of the supplementary alphabets with the main font face concerns not only alphanumeric glyphs. Even essentially geometrical shapes should also reflect the characteristic features of the main font, for example, the thickness of stems, the ending of arms, etc. Seemingly trivial glyphs, such as arrows, serve as a convenient example: they have slightly different shapes in each of our math fonts—see Figure 7.

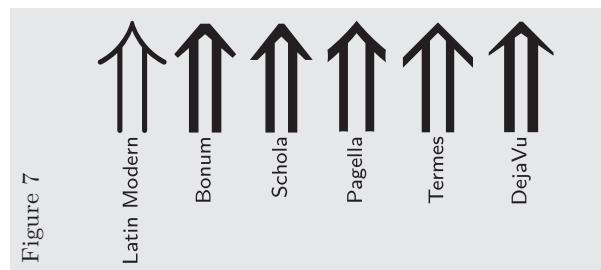


Figure 7

Another example is the shape of integrals. Historically, the integral symbol originates from the letter 'long s'¹⁰ which is nowadays identical with a barless 'f'. Therefore, we did our best to preserve some characteristic features of the letter 'f' in the design of the integral shape—see Figure 8.

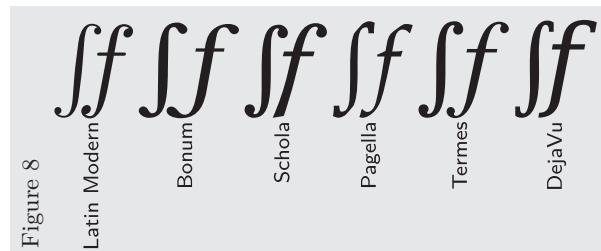


Figure 8

We are not going to dwell on the visual aspects of the math font design, as the number of details relevant for a font with over 4000 glyphs (and

¹⁰ The cursive long 's' for denoting an integral operation, i.e., infinite summation, was introduced by Gottfried Wilhelm Leibniz in 1675 in an unpublished paper *Analyseos tetragonistica pars secunda* (*Second part of analytical quadrature*).

counting) would perhaps become rather overwhelming. Notwithstanding, one aspect needs emphasizing, namely, the problem of sidebearings and kerning for alphanumeric symbols.

It is generally accepted by typographers that math symbols should have larger sidebearings than the respective text glyphs. In particular, *T_EX* math italic glyphs are a little bit broader and have larger sidebearings than the text italic glyphs. It is not obvious, however, how large such sidebearings should be. We have already experimented with a few sizes but further tuning will probably be necessary. Of course, the broadening of sidebearings should not be applied to the basic font, that is, regular upright, because of multiletter names of functions and operators, such as ‘sin’ or ‘max’, which are traditionally typeset with regular upright letters.

As regards the problem of kerning, we decided not to include kerns in math fonts, although providing kerns for the upright regular alphabet might be reasonable. Actually, we consider introducing special math kerning (so-called “staircase” kerns a.k.a. “cut-ins”; see Section 7). Thankfully, nobody has complained yet because of the lack of kerning in our math OTF fonts.

6.3 Repertoire issues

At present, a common standard for the repertoire of characters that should be present in an OTF math font does not exist. After several debates (mostly during *T_EX* conferences) and many experiments, we came up with the tentative repertoire scheme presented in Table 2, which can be considered a detailed specification of Table 1.

B – basic letters, A – accented letters, G – Greek letters,
D – digits, O – other symbols, P – punctuation

	B	A	G	D	O	P
plain (upright, serif)	+ _s	+ _s	+ ^d _s	+ _s	+	+ _s
italic	+ _s		+ _s			
bold	+ _s		+ ^d _s	+ _s		
bold italic	+ _s		+ _s			
sans-serif	+					
sans-serif italic	+			+		
sans-serif bold	+			+		
sans-serif bold italic	+			+		
calligraphic	+					
bold calligraphic	+					
Fraktur	+					
bold Fraktur	+					
double-struck	+				+	
monospace	+				+	

d — digamma excluded from relevant Unicode blocks

s — subscripts are to be added

Table 2

We would like all TG math fonts (Bonum, DejaVu, Pagella, Schola, and Termes) to share this pattern. For LM Math, however, we adopted another scheme because of legacy concerns. Currently, LM Math contains circa 4800 glyphs while the remaining fonts contain circa 4250 glyphs each. LM Math contains more subscripts, as the original sources already provided them. On the other hand, the TG math fonts contain more size variants of integral symbols. This yields circa 550 glyphs more (net) in LM Math. At the moment, *T_EX* Gyre DejaVu Math contains (in accordance with Table 2) subscript variants for bold upright glyphs, while the remaining TG math fonts need to be complemented with these glyphs. It is a typical instance of the frequently occurring “backtracking” procedures: the final decision was undertaken only after a few fonts had been released.

Concerning subscripts, it should be emphasized that subscript glyphs for the TG math fonts are obtained using the approach applied in the Metafont sources of the Euler family of fonts, that is, by non-uniform rescaling of the respective normal-size glyphs. In a way, such “optical scaling” can be considered undesirable; nevertheless, it proved acceptable for the Euler fonts, thus we decided to apply it also to the TG math fonts.

Some rather quirky characters which received Unicode slots are spaces. Unicode [35] defines quite a few space-related glyphs:

0008	BACKSPACE (<control>)
*0020	SPACE
*00A0	NO-BREAK SPACE
*2002	EN SPACE
*2003	EM SPACE
1361	ETHIOPIC WORDSPACE
1680	OGHAM SPACE MARK
*2004	THREE-PER-EM SPACE
*2005	FOUR-PER-EM SPACE
*2006	SIX-PER-EM SPACE
*2007	FIGURE SPACE
*2008	PUNCTUATION SPACE
*2009	THIN SPACE
*200A	HAIR SPACE
*200B	ZERO WIDTH SPACE
*202F	NARROW NO-BREAK SPACE
*205F	MEDIUM MATHEMATICAL SPACE
2408	SYMBOL FOR BACKSPACE
2420	SYMBOL FOR SPACE
3000	IDEOGRAPHIC SPACE
303F	IDEOGRAPHIC HALF FILL SPACE
*FEFF	ZERO WIDTH NO-BREAK SPACE
1DA7F	SIGNWRITING LOCATION-WALLPLANE SPACE
1DA80	SIGNWRITING LOCATION-FLOORPLANE SPACE
E0020	TAG SPACE

We decided to include a subset of the Unicode-defined spaces (marked with asterisks in the above list), even though their meaning and usage seems vague. The problem of spaces touches a general

problem of the relation between the Unicode standard and typography. We discuss this topic in more detail in the next section.

The next two pages show the representative subset (for LM Math and TG Termes Math) of the repertoire we adopted as the GUST e-foundry “private standard”; gray squares denote zero-width characters. As one can see, the repertoires are very similar. One can assume that the difference in the repertoire will be imperceptible in practical applications.

6.4 Unicode: the typographer’s friend or enemy?

An important subject, closely related to the contents and repertoire issues, is briefly mentioned in Sections 4.2, 6.1, and 6.3: the problem of the character set defined by the Unicode standard [35], and specified for math fonts in *Unicode Technical Report #25* [37].

The Unicode Consortium claims that “the Unicode standard follows a set of fundamental principles” and gives, among others, the following “principle”: *characters, not glyphs and semantics*. We are not able to reconcile these principles with the cases discussed in this section.

It should be emphasized that not all glyphs used extensively in typography, in particular, in or out of math formulas, are assigned Unicode numbers. Examples of such glyphs are small caps and old style numerals. Nothing in these two classes of glyphs has received Unicode numbers, although there are codes for much more narrowly used double struck characters or monospaced and sans-serif digits.

Another example of “unicodeless” glyphs are the pieces used for assembling extensible characters (cf. Section 6.1, Figures 5 and 6).

It is not so bad if whole blocks of characters are included or excluded. The worse situation is an inconsistency of including only some glyphs. The abovementioned double struck glyphs are a good example of such a situation. Here is the group of double struck glyphs assigned Unicode slots, without apparent rhyme or reason:

```
2102 DOUBLE-STRUCK CAPITAL C
210D DOUBLE-STRUCK CAPITAL H
2115 DOUBLE-STRUCK CAPITAL N
2119 DOUBLE-STRUCK CAPITAL P
211A DOUBLE-STRUCK CAPITAL Q
211D DOUBLE-STRUCK CAPITAL R
2124 DOUBLE-STRUCK CAPITAL Z
213C DOUBLE-STRUCK SMALL PI
213D DOUBLE-STRUCK SMALL GAMMA
213E DOUBLE-STRUCK CAPITAL GAMMA
213F DOUBLE-STRUCK CAPITAL PI
2140 DOUBLE-STRUCK N-ARY SUMMATION
2145 DOUBLE-STRUCK ITALIC CAPITAL D
2146 DOUBLE-STRUCK ITALIC SMALL D
```

```
2147 DOUBLE-STRUCK ITALIC SMALL E
2148 DOUBLE-STRUCK ITALIC SMALL I
2149 DOUBLE-STRUCK ITALIC SMALL J
```

The remaining letters of the alphabet (upper and lower case) and digits received mathematical codes (1D538–1D56B), with gaps at the glyphs above.

Another notable example are sub- and superscripts, theoretically not needed in math fonts, because the sub- and superscript characters (“unicodeless”) are accessed there by the OTF feature mechanism (the `ssty` feature, cf. Section 6.1). In practice, for legacy reasons, sub- and superscript glyphs are expected to be present in a text font, and, consequently, in the basic (plain) charset of a math font too—see Table 1. The Unicode standard [35] enumerates the following Latin letters in this context:

```
1D62 LATIN SUBSCRIPT SMALL LETTER I
1D63 LATIN SUBSCRIPT SMALL LETTER R
1D64 LATIN SUBSCRIPT SMALL LETTER U
1D65 LATIN SUBSCRIPT SMALL LETTER V
2090 LATIN SUBSCRIPT SMALL LETTER A
2091 LATIN SUBSCRIPT SMALL LETTER E
2092 LATIN SUBSCRIPT SMALL LETTER O
2093 LATIN SUBSCRIPT SMALL LETTER X
2094 LATIN SUBSCRIPT SMALL LETTER SCHWA
2095 LATIN SUBSCRIPT SMALL LETTER H
2096 LATIN SUBSCRIPT SMALL LETTER K
2097 LATIN SUBSCRIPT SMALL LETTER L
2098 LATIN SUBSCRIPT SMALL LETTER M
2099 LATIN SUBSCRIPT SMALL LETTER N
209A LATIN SUBSCRIPT SMALL LETTER P
209B LATIN SUBSCRIPT SMALL LETTER S
209C LATIN SUBSCRIPT SMALL LETTER T
2C7C LATIN SUBSCRIPT SMALL LETTER J
2071 SUPERSCRIPT LATIN SMALL LETTER I
207F SUPERSCRIPT LATIN SMALL LETTER N
```

Besides the somewhat surprising presence of the ‘schwa’ character and the striking asymmetry between the number of sub- and superscripts, most questionable here is the incompleteness of the Latin alphabet. So far, we have not included these glyphs in neither text nor math fonts.

Another example concerns math italic symbols. The Unicode standard reads:

```
1D44E MATHEMATICAL ITALIC SMALL A
1D44F MATHEMATICAL ITALIC SMALL B
1D450 MATHEMATICAL ITALIC SMALL C
1D451 MATHEMATICAL ITALIC SMALL D
1D452 MATHEMATICAL ITALIC SMALL E
1D453 MATHEMATICAL ITALIC SMALL F
1D454 MATHEMATICAL ITALIC SMALL G ⇝?
1D456 MATHEMATICAL ITALIC SMALL I
1D457 MATHEMATICAL ITALIC SMALL J
1D458 MATHEMATICAL ITALIC SMALL K
1D459 MATHEMATICAL ITALIC SMALL L
1D45A MATHEMATICAL ITALIC SMALL M
1D45B MATHEMATICAL ITALIC SMALL N
1D45C MATHEMATICAL ITALIC SMALL O
1D45D MATHEMATICAL ITALIC SMALL P
1D45E MATHEMATICAL ITALIC SMALL Q
1D45F MATHEMATICAL ITALIC SMALL R
```


Glyph repertoire of T_EX Gyre Termes Math

ASCII: ! "#\$%&'()*+, -./0123456789; <=> ? @ ABCDEFGHIJKLMNOPQRSTUVWXYZ
VWXZY[\]^_` abcdefghijklmnopqrstuvwxyz{|}~

Latin-1: ¡¢£¤¥¦§¨©ª«¬®¬°±’µ¶·,º»¿ÀÁÃÄÅÆÇÈÉÊËÌÍÏÐÑÒÓÔÖÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëìíïðñòóôöö÷øùúûüýþÿ

Mathematical Operators: $\forall \exists \nexists \emptyset \Delta \nabla \notin \in \ni \exists \nexists \prod \coprod \sum - \mp + \backslash * \circ \cdot \sqrt{\infty} \llcorner \lrcorner \triangleleft \triangleleft \triangleleft \triangleleft \wedge \vee \cup \cap \int$

Box Drawing Block Elements: - | L + — █

Supplemental Mathematical Operators: $\odot \oplus \otimes \cup \uplus \sqcap \sqcup \times \int \int \int \oint \times \amalg \leq \geq \lessapprox \gtrapprox \triangleleft \triangleright \veeleftarrow \veerightarrow \wedgeleftarrow \wedgerightarrow \triangleleft \triangleright$

Mathematical Alphanumeric Symbols: **A B C D E F G H I J K L M N O P Q R S T U V W X Y Z a b c d e f g h i j k l m n o p q r s t u v w x y**

*z ABCDEFGHIJKLMNOPQRSTUVWXYZabcdeghijklmnopqrstuvwxyzACDGJ
KNOOPQSTUVWXYZabcdfhi jklmnopqrstuvwxyzABCDEFG*

H E T J K L M N O P Q R S T U V W X Y Z a b c d e f g h i j k l m n o p q r s t u v w x y z A B

DEFGIJKLMOSTUVWXYZ abcdefghijklmnopqrstuvwxyz ABCDEFGH

IJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ

q r s t u v w x y z i j A B Γ Δ E Z H Θ I K A M N Ξ O P R Θ S T T Y F X Ψ Ω V a b γ δ ε ζ η θ i k λ μ ν ξ o p o c σ t u ψ γ ψ ω δ e ψ z φ ρ σ A B Γ Δ E Z H Θ I K A M N Ξ O P R Θ S T T Y F X Ψ Ω V

$\alpha\beta\gamma\delta\varepsilon\zeta\eta\theta\iota\kappa\lambda\mu\nu\xi\sigma\pi\rho\varsigma\sigma\tau\upsilon\varphi\chi\psi\omega\partial\epsilon\psi\chi\varphi\sigma\text{ΑΒΓΔΕΖΗΘΙΚΑΜΝΞΟΠΡ}$
 $\Theta\sum\Gamma\Phi\chi\varPsi\Omega\forall\alpha\beta\gamma\delta\varepsilon\zeta\eta\theta\iota\kappa\lambda\mu\nu\xi\sigma\pi\rho\varsigma\sigma\tau\upsilon\varphi\chi\psi\omega\partial\epsilon\psi\chi\varphi\sigma\text{ΑΒΓΔΕΖΗ}$

ψωδεθηφρω01234567890123456789012345678901234567890123456789

```
1D460 MATHEMATICAL ITALIC SMALL S
1D461 MATHEMATICAL ITALIC SMALL T
1D462 MATHEMATICAL ITALIC SMALL U
1D463 MATHEMATICAL ITALIC SMALL V
1D464 MATHEMATICAL ITALIC SMALL W
1D465 MATHEMATICAL ITALIC SMALL X
1D466 MATHEMATICAL ITALIC SMALL Y
1D467 MATHEMATICAL ITALIC SMALL Z
```

Math italic ‘h’ is apparently missing. In fact, the Unicode Consortium decided to leave the slot 1D455 unused forever and “inflate” the meaning of the slot formerly assigned to the Planck constant:

```
210E PLANCK CONSTANT
= height, specific enthalpy, ...
* simply a mathematical italic h;
  this character's name results
  from legacy usage
```

More on Unicode’s ambiguities, inconsistencies, riddles, curiosities, etc. concerning typography applications, especially math typesetting, can be found in Piotr Strzelczyk’s ruminations [23].

From the typographer’s point of view, the enumeration of all signs used in the world does not seem to be a good idea. Therefore, it should be no surprise that it turned out to be impossible to create a math OTF font that has an internally logical and coherent structure and, at the same time, is practically useful. Conforming rigorously to the mentioned specifications does not help too much—it would lead, in our opinion, to fonts containing mostly seldom used glyphs. Needless to say, research examining which characters from the Unicode repertoire are actually used in practice, would be welcome. Lacking such empirical data, we adopted Cambria Math as our “reference point”. Cambria Math contains circa 6500 glyphs; we decided to reduce this number to at most 4500 for the TeX Gyre series of math fonts. We can reluctantly consider proposals for suitable extensions, if truly needed.

6.5 Compatibility issues

We already explained why 100-percent compatibility of the LM text fonts with the parent Computer Modern fonts is unfeasible; thereby, the same applies, even to a greater extent, to the LM Math font (cf. the case of horizontal extensible braces in Section 6.1).

Recall that we used an alternative calligraphic alphabet in LM Math (Section 6.1). This incompatibility can be relatively easily patched, by including two calligraphic scripts in the font and using OTF ‘stylistic sets’ features, implemented as the OTF features **ss01–ss20** (see [33]).

There is, however, a flaw shared by both Computer Modern and Euler calligraphic alphabets and thus inherited by the Latin Modern Math: the lack

of a corresponding lower case alphabet. The Unicode specification assigns slots to lower case mathematical calligraphic letters, implying that they are expected to be present in math fonts.

We could not find a calligraphic font with a proper license optically matching Euler or Computer Modern upper case calligraphic letters, and we gave up in advance the idea of designing the respective matching lower case letters ourselves. Instead, we began to consider the inclusion of yet another stylistic set, a “home-made” calligraphic font inspired by (but not based on) the original Computer Modern calligraphic script. Although we would never dare to make a text calligraphic font without close co-operation by a professional type designer, we took a chance and attempted to prepare a symbol calligraphic font for TG DejaVu Math.

There was an additional reason for doing our own calligraphic script. Anticipating future work (see Section 7 below), we looked for a calligraphic script matching a sans-serif font. Having not found any, we decided to prepare our own. Because our calligraphic alphabet for TG DejaVu Math was, of course, defined by a parametric MetaType 1 program, it was possible to prepare a sans-serif (linear) variant of the calligraphic script with reasonable effort. A tentative, experimental linear calligraphic alphabet is shown in Figure 9: the calligraphic script used in TG DejaVu Math (top) and its linear variant to be used in a sans-serif math font (bottom).



Figure 9

Fortunately, there is no issue of compatibility regarding the TG math fonts, as there are no predecessors. The only question is the similarity of repertoire between the LM and TG math fonts. As explained in Section 6.3, the repertoires are bound to differ slightly, for the usual legacy reasons.

For the same reasons, some technical details of the font structure also cannot be implemented similarly. Worthy of mentioning in this context is a peculiar difference between the LM and TG math fonts related to integrals; namely, the TG math fonts contain extensible integrals, which are definable within the OTF math format—but we are not aware of any typesetting engine that can take advantage of this possibility.

7 Plans for the future

7.1 Testing and maintenance

Tasks that are important today and will be forever important in the future are maintenance and testing. There is, of course, neither a single tool for testing nor a unique maintenance procedure. Each case demands a specific approach.

It is Piotr Pianowski who is responsible for testing fonts and preparing adequate tools. Tests refer to both the appearance of the fonts and their internal structure. In particular, the intermediate PostScript Type 1 code needs checking. The following example shows a case where the inspection of the code revealed an error in the `parenright.ex` procedure (describing an extender of the extensible right parenthesis, which should be just a rectangle), probably due to the wrong rounding procedure: the number ‘1’ means that the line drawn by the command `rlineto` is not strictly vertical. The left column contains the correct code for the corresponding component of the extensible left parenthesis:

<code>/parenleft.ex {</code>	<code>/parenright.ex {</code>
143 609 <code>hsbw</code>	338 609 <code>hsbw</code>
127 418 <code>rmoveto</code>	128 418 <code>rmoveto</code>
-127 <code>hlineto</code>	-128 <code>hlineto</code>
-418 <code>vlineto</code>	1 -418 <code>rlineto</code>
127 <code>hlineto</code>	126 <code>hlineto</code>
<code>closepath</code>	<code>closepath</code>
<code>endchar</code>	<code>endchar</code>
<code>} ND</code>	<code>} ND</code>

It is next to impossible to perceive such a tiny deformity on a printout of glyph shapes, which does not mean that the bug should not be fixed.

The next example, Figure 10, shows one of the tests we use for checking a vexing problem regarding Greek letter names. Some Greek letters have a shape variant, and there is an unfortunate discrepancy between `TEXies` and the rest of the world (the rightmost two columns marked with asterisks) as to which glyphs are considered “normal” and which “variant”.

Almost every time we deal with the Greek alphabet, a mistake in variant names tries to creep in. Without tests of this kind we would be lost.

	texgyredejavu-math.otf			
	normal	var	normal*	var*
\mstd	$\epsilon \phi \rho \theta$	$\epsilon \varphi \rho \vartheta$	$\epsilon \varphi \rho \theta$	$\epsilon \phi \varrho \vartheta$
\mrm	$\epsilon \varphi \rho \theta$	$\epsilon \phi \varrho \vartheta$	$\epsilon \phi \rho \theta$	$\epsilon \varphi \varrho \vartheta$
\mit	$\epsilon \phi \rho \theta$	$\epsilon \varphi \varrho \vartheta$	$\epsilon \varphi \rho \theta$	$\epsilon \phi \varrho \vartheta$
\mbf	$\epsilon \varphi \rho \theta$	$\epsilon \phi \varrho \vartheta$	$\epsilon \phi \rho \theta$	$\epsilon \varphi \varrho \vartheta$
\mbi	$\epsilon \phi \rho \theta$	$\epsilon \varphi \varrho \vartheta$	$\epsilon \varphi \rho \theta$	$\epsilon \phi \varrho \vartheta$
\mbfss	$\epsilon \phi \rho \theta$	$\epsilon \varphi \varrho \vartheta$	$\epsilon \varphi \rho \theta$	$\epsilon \phi \varrho \vartheta$
\mitss	$\epsilon \phi \rho \theta$	$\epsilon \varphi \varrho \vartheta$	$\epsilon \varphi \rho \theta$	$\epsilon \phi \varrho \vartheta$
\mbiss	$\epsilon \phi \rho \theta$	$\epsilon \varphi \varrho \vartheta$	$\epsilon \varphi \rho \theta$	$\epsilon \phi \varrho \vartheta$

Figure 10

The last example concerning maintenance and testing issues, Figure 11, shows a typical test of the structure of extensible characters, “embellished” horizontal arrows in this case: ‘vh 2’ means that there are 2 size variants, ‘ch 3’ and ‘ch 5’—that there are 3 and 5 horizontal components of a given glyph, respectively. Tests of this kind are essential for maintaining uniformity across a font collection as well as inside a single font.

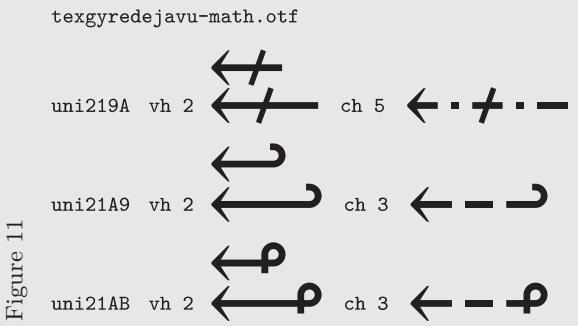


Figure 11

As a result of remarks to date from the users of our fonts, we gathered a list of recommended fixes and improvements—some trivial, like reports on malformed glyphs or wrongly assigned Unicode slots, some fairly difficult, like suggestions to implement anchors or math staircase kerns.

The latter two potential improvements are still pending, mainly because of vague specification and the question of practical application. Being unsure whether all relevant typesetting programs would be able to handle such improvements, we preferred to linger till the engines would become stable. It seems that the time is ripe to attempt these extensions, especially as the staircase kerns were ultimately implemented in Xe^TE_X in the middle of 2014.

We are also not sure which OTF features should be present in math fonts. At the moment, only math-oriented OTF features are implemented. Perhaps we should consider the inclusion of text-oriented OTF features too, like the mentioned `onum`, `lnum`, `pnum`,

and `tnum` for switching between numeral variants, or stylistic sets for switching between calligraphic alphabets.

Also as suggested by users, we consider excerpting a number of glyphs, mostly geometrical shapes and arrows but also selected math operators and relational symbols, etc. (but excluding extensible and subscript characters), from math fonts and transferring them into the respective text fonts. Such glyphs, though prepared for math-oriented applications, can also be fruitfully used in conventional fonts, that is, those not equipped with the MATH table, for the typesetting of technical documents. This, obviously, requires a proper specification and careful selection of the glyphs in question. This is, in fact, one of the planned stages of work on new fonts.

Of course, MetaType1 itself also requires maintenance: enhancing, modifying, fixing, etc. For example, we had to extend the otherwise stable set of MetaPost macros used in MetaType1 in order to handle the math extension of the OTF specification.

Commencing our works for the GUST e-foundry, we underestimated the inexorable Hofstadter's Law (see footnote 7 on page 324) which apparently applies not only to time resources. We believed that MetaType1 could be kept simple: just MetaPost, a few trivial Gawk scripts and a stand-alone, stable converter to PostScript Type 1 fonts, T1utils, and that's all. In accordance with Hofstadter's Law, the task has unavoidably turned out to be much more complex than we expected and the implementation—too heterogeneous.

In order to remedy this, we intend to unify MetaType1 by eliminating Gawk, Perl, T1utils, and, as we mentioned in Section 3.1, AFDKO. We aim at employing two basic “subengines”, namely, MetaPost (for generating outlines) and Python (for arranging the MetaPost output) plus one external, possibly replaceable, “subengine”, for now, the FontForge Python library (for generating OTF fonts and the theoretically obsolescent but still widely used PostScript Type 1 fonts).

7.2 More GUST e-foundry fonts

In the nearest future, we would like to elaborate and release three experimental math fonts, namely: bold math, sans-serif math, and monospaced math. These fonts fit neither the Microsoft nor Unicode specifications ([29] and [37], respectively). Therefore, we have to start by working out an altered specification, adjusted to our purposes, for these non-standard cases. For example, it is not at all clear what to do with the sans-serif alphabet in a

sans-serif math font: should it be omitted, left intact, or modified somehow (how?).

Bold math fonts can be used in titles containing math formulas, as shown in Figure 12 (a tentative version of TG Termes Bold Math is used).

1. Equivalence of the integral ($\iint_{\partial S} \mathbf{E} \cdot d\mathbf{S} = 0$) and differential ($\nabla \cdot \mathbf{E} = 0$) formulas

In this section, we shall prove that the integral and differential forms of Maxwell's equation known as “Gauss's law for magnetism”, i.e.:

$$\iint_{\partial S} \mathbf{E} \cdot d\mathbf{S} = 0 \quad (1)$$

Figure 12 and

$$\nabla \cdot \mathbf{E} = 0 \quad (2)$$

are equivalent.

Nowadays, many documents are being typeset in sans-serif, even schoolbooks. Computer presentations may serve as another example. For such applications sans-serif math seems plausible. An example of such an application is shown in Figure 13 (a tentative version of TG DejaVu Sans Math is used).

Various notations for continued fractions

The integers a_0, a_1 etc., are called *coefficients* or *terms* of the continued fraction. One can abbreviate the continued fraction

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3}}}$$

in the notation of Carl Friedrich Gauss as

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3}}}$$

or in the notation of Alfred Pringsheim as

$$x = a_0 + \cfrac{1}{|a_1|} + \cfrac{1}{|a_2|} + \cfrac{1}{|a_3|}.$$

source: http://en.wikipedia.org/wiki/Continued_fraction

Figure 13

TEX users are accustomed to the use of control sequences for math-oriented glyphs such as `\infty`, `\sum` or `\pm`. Some of these glyphs can be accessed (typed in and displayed) directly in text editors, provided the font used by the editor contains the relevant glyphs. The lion's share of nominally math-oriented glyphs can also be prepared as monospaced glyphs, usable in text editors, provided they are assigned Unicode numbers. One can expect that it will

improve the legibility of sources and, thereby, the efficiency of preparation of such documents—see Figure 14 (a tentative version of TG DejaVu Mono Math is used).

Figure 14

```
\section{Алгоритм де Кастельє}

Заданий многочлен Бернштейна $B$ ступеня $n$ з опорними точками $\beta_0, \dots, \beta_n$  

$$ B(t) = \sum_{i=0}^n \beta_i b_{i,n}(t),  

\text{де } B - \text{базис многочлена Бернштейна, многочлен в точці } t \text{ може бути визначений за допомогою рекурентного спiвiдношення:  

\$\$\\eqalign{& \beta_i^{(0)} = \beta_i \\ & \beta_i^{(j)} = \beta_i^{(j-1)}(1-t) + \beta_{i+1}^{(j-1)}t, \\ & \quad \backslash mbox{, } i=0, \dots, n-j \backslash mbox{, } j=1, \dots, n \backslash cr } }\\\$\$  

\text{Тодi вiзначення } B \text{ в точцi } t \text{ може бути вiзначено в } n \text{ крокiв алгоритmu.  

Результат } B(t) \text{ дано за:  

\$\$  

& B(t) = \beta_0^{(n)}.  

\text{Tакож, крива Безье } B \text{ може бути роздiлена в точцi } t \text{ на двi кривi з вiдповiдними опорними точками:  

\$\$  

& \beta_0^{(0)}, \beta_0^{(1)}, \dots, \beta_0^{(n)}  

& \beta_0^{(n)}, \beta_1^{(n-1)}, \dots, \beta_n^{(0)}  

\$\$  

\text{source: } \url{https://uk.wikipedia.org/wiki/Алгоритм_де_Кастельє}
```

Note that treating subscripts incoherently by the Unicode Standard (Section 6.4) may prove to be a significant impediment in the latter case, that is, for fonts without full subscript support. Perhaps the defining of the complete set of subscripts and superscripts (partially “unicodeless”) and accessing them via stylistic set features (ss01–ss20) can be a solution, provided a given text editor handles the relevant OTF features.

7.3 Legal issues

The problem of copyrights and licenses has clung to us like a leech from the very beginning and still persists. We are not copyright experts and we do not want to be. Being sick of trouble with releasing our work due to discussions about legal aspects of distributing free fonts, we coined a pun *lice-sense*. Just one example: the release of TG DejaVu Math was delayed by about a year because of doubts raised concerning legal matters.

Having said this, we should emphasize that it does not mean that there has been no activity from the side of the GUST e-foundry regarding legal issues. On the contrary, our magnificent liaison officer, Jerzy Ludwichowski, has made Herculean efforts in order to provide appropriate licenses for GUST fonts. In particular, he prepared a proposal of a license for the URW fonts that would suit GUST e-foundry needs, managed to contact in person the URW++

managing director, Dr. Peter Rosenfeld, and, after long negotiations, received the gracious approval for the additional license.

All the fonts released so far are licensed under the GUST Font License (GFL; see [30]), except for TG DejaVu Math which has a somewhat complex license (see [9], the Manifest file for TG DejaVu Math). Recently, many fonts are being released under the SIL Open Font License (OFL; see [34]); therefore, we consider dual-licensing GUST fonts (GFL+OFL).

More details concerning legal matters relevant to GUST e-foundry fonts can be found in a series of publications by Jerzy Ludwichowski—see, for example [18, 19, 20, 21].

7.4 Constraints of tradition vs. our dreams

We are not particularly enthusiastic about font technology nowadays, but we are not going to spit into the wind, and try to make the best of what is available. This does not mean, however, that we are going to relinquish dreams of a successor to the currently prevalent “Knuth–Gutenberg” model of typesetting, that is, the model of stiff rectangles (types) arranged within a larger rectangle (page; a series of pages being called a document), deeply ingrained in Johannes Gutenberg’s technology of movable type, and transferred by Donald E. Knuth, among others, to the realm of computers.

Computers facilitate some operations, such as kerning and ligatures, thereby accelerating work on documents. The ease of use of affine transformations is also considered an advantage of computers by many graphic designers. We are inclined to consider both these aspects as being of rather equivocal benefit. Leaving aside these philosophical questions and a far-reaching yet obvious idea, in fact also philosophical, that a font could be defined as a structured collection of general purpose object programs, we confine ourselves to pointing out two examples of aspects that might be improved within the framework of the contemporary edifice of typesetting.

Shrinkable and stretchable spaces, as in TeX, would supposedly be convenient also in OTF fonts and seem not too hard to implement. This simple problem touches on a fairly general and not in the least trivial problem which could be called “the conflict of competence”: which “knowledge” should be implemented in a font and which—in a typesetting program.

Another improvement, this time hard to implement, is related to a naïve question: why must additional alphabets be embedded into a single math font? The answer is simple: operating systems are poorly designed with regard to font management. In

particular, a user is not allowed to define a “private” family of fonts—the commonplace pattern “regular, regular italic, bold, bold italic” is very difficult to dislodge. One can imagine other variants of this idea, namely, borrowing components for extensible characters or kerns from several fonts. As far as we know, such an approach has not been implemented in any typesetting program or any operating system. *T_EX* is no exception (unless virtual fonts are used); note, for example, that changing fonts within a word switches off the *T_EX* hyphenation mechanism.

It should be emphasized that several such problems were tackled in *LuaT_EX* by Hans Hagen and team; it does not seem, however, as if amendments of this kind are to be introduced worldwide in any near future. This is understandable, as the constraints of tradition and compatibility usually slow down the innovative processes. For example, recently announced advancements in the OTF format specification (see, e.g., [6]), can actually be considered a step backward towards a previously abandoned idea, temporarily as it turns out, of *multiple master fonts*. Anyway, this announcement augurs both ill and well for us: it means, on the one hand, that we will probably have to reimplement MetaType 1 in order to keep pace with the surrounding world and, on the other hand, that we still have something to do and something to think about.

8 Acknowledgements

We are indebted to all the people and *T_EX* groups that have supported our font enterprises. Almost all the GUST e-foundry projects have been kindly supported by the Czechoslovak *T_EX* user group *C_STUG*, the German-speaking *T_EX* user group DANTE e.V., the Polish *T_EX* Users Group GUST, the Dutch-speaking *T_EX* user group NTG, TUG India, UK-TUG, and, last but not least, TUG. In a few cases, GUTenberg, the French-speaking *T_EX* Users Group, supported us too.

We are very grateful to Karel Píška and Ulrik Vieth, who performed extensive tests of our fonts and inspired us with insightful comments, and to Marcin Woliński, who provided the indispensable *I^AT_EX* support for our fonts.

The exceptional, personal thanks we owe to our friends who kept our spirits up for many years and tirelessly encouraged us to work on fonts: Hans Hagen, Johannes Küster, Jurek Ludwichowski, Volker RW Schaa, Jola Szelatyńska—hearty thanks!

All trademarks belong to their respective owners and have been used here for informational purposes only.

References

Presentations, publications and packages

- [1] Lars Engebretsen, *Almost European Fonts*
<https://ctan.org/pkg/ae>
- [2] Michael Everson, *The Alphabets of Europe*
<http://www.evertype.com/alphabets/>
- [3] Lee Hetherington, Eddie Kohler, *T1utils*
<http://www.lcdf.org/type/t1asm.1.html>
<http://www.lcdf.org/type/t1disasm.1.html>
- [4] John D. Hobby, *MetaPost*
<https://ctan.org/pkg/metapost>
- [5] Donald E. Knuth, *The T_EXbook, T_EX: The Program, The Metafontbook, Metafont: The Program, Computer Modern Typefaces*, Computers & Typesetting, vol. A–E, Addison-Wesley, Reading, Massachusetts, 1986
- [6] John Hudson, *Introducing OpenType Variable Fonts*, 2016
<https://medium.com/@tiro/introducing-opentype-variable-fonts-12ba6cd2369>
- [7] Bogusław Jackowski, Janusz M. Nowacki, Piotr Strzelczyk, *Antyqua Póltawskiego: a parameterized outline font*, Proceedings of the EuroT_EX Conference, Heidelberg, Germany, 1999
article: <http://www.staff.uni-giessen.de/partosch/eurotex99/jackowski>
download: <http://www.gust.org.pl/projects/e-foundry/politawski>
- [8] Bogusław Jackowski, Marek Ryćko, *Polish extension of Computer Modern fonts*
<https://ctan.org/pkg/pl-mf>
- [9] Bogusław Jackowski, Piotr Strzelczyk, Piotr Pianowski, GUST e-foundry math fonts
The Latin Modern Math (LM Math) font:
<http://www.gust.org.pl/projects/e-foundry/lm-math>
The T_EX Gyre (TG) Math Fonts:
<http://www.gust.org.pl/projects/e-foundry/tg-math>
<https://ctan.org/pkg/tex-gyre-math>
- [10] Bogusław Jackowski, Piotr Strzelczyk, *How to make more than one math OpenType font or the Beasts of Fonts*, DANTE 2011 Meeting, Bremen, Germany, 2011
<http://www.gust.org.pl/projects/e-foundry/math/beasts05.pdf>
- [11] Bogusław Jackowski, Janusz M. Nowacki, Piotr Strzelczyk, *The Latin Modern (LM) Family of Fonts*
<http://www.gust.org.pl/projects/e-foundry/latin-modern>
- [12] Bogusław Jackowski, Janusz M. Nowacki, *Latin Modern fonts: how less means more*, Proceedings of the EuroT_EX conference, Pont-à-Mousson, France, 2005
<http://tug.org/TUGboat/tb27-0/jackowski.pdf>

- [13] Bogusław Jackowski, Janusz M. Nowacki, Piotr Strzelczyk, *MetaType 1: a MetaPost-based engine for generating Type 1 fonts*, 2001
article: <http://www.ntg.nl/maps/26/15.pdf>
presentation: <http://ntg.nl/eurotex/JackowskiMT.pdf>
download: <https://ctan.org/pkg/metatype1>
- [14] Bogusław Jackowski, Janusz M. Nowacki, Piotr Strzelczyk, *The TeX Gyre (TG) Collection of Fonts* <http://www.gust.org.pl/projects/e-foundry/tex-gyre>
- [15] Bogusław Jackowski, Janusz M. Nowacki, Piotr Strzelczyk, *TeX Gyre Pagella Math or Misfortunes of Math Typographer*, BachoTeX XX, Bachotek, Poland, 2012
<http://www.gust.org.pl/projects/e-foundry/math/misfortunes02.pdf>
- [16] Jörg Knappen, Norbert Schwarz, *European Computer Modern Fonts* <https://ctan.org/pkg/ec>
- [17] Donald E. Knuth, *Metafont and MetaPost logo fonts*. <https://ctan.org/pkg/mflogo-font>
- [18] Jerzy B. Ludwichowski, *GUST font licenses*, BachoTeX XIV, Bachotek, Poland, 2006
<http://tug.org/fonts/licenses/gfl.pdf>
- [19] Jerzy B. Ludwichowski, Karl Berry, *GUST Font License: An application of the L^AT_EX Project Public License*, XVII European TeX Conference and BachoTeX XV, Bachotek, Poland, 2007; TUGboat, Volume 29 (2008), No. 1
http://www.gust.org.pl/projects/e-foundry/licenses/tb91Berry_Ludwichowski.pdf
- [20] Jerzy B. Ludwichowski, *Licensing of the TeX Gyre family of fonts*, XIX European TeX Conference and 3rd International ConTeXt User Meeting, The Hague, The Netherlands, 2009
<https://www.ntg.nl/EuroTeX/2009/slides/jerzy-slides.pdf>
- [21] Jerzy B. Ludwichowski, *Is there life besides licensing?*, DANTE e.V. General Meeting, Bremen, Germany, 2011
<https://www.dante.de/events/Archiv/dante2011/programm/vortraege/folien-jl.pdf>
- [22] Janusz M. Nowacki, *Polish fonts* <http://jmnn.pl/en/>
- [23] Piotr Strzelczyk, *Standard Unicode w typografii*, Acta Poligraphica nr 1/2013 (in Polish; to be published also in English under the title *Standard Unicode in typography*)
http://www.cobrpp.com.pl/actapoligraphica/uploads/pdf/AP2013_01_Strzelczyk.pdf
see also: Piotr Strzelczyk, *(uni)coding of math fonts*, BachoTeX XIX, Bachotek, Poland, 2011
http://www.gust.org.pl/bachotex/2011-en/presentations/Strzelczyk_1_2011
- [24] Ulrik Vieth, *OpenType math illuminated*, TUGboat, Volume 30 (2009), No. 1
<http://tug.org/TUGboat/tb30-1/tb94vieth.pdf>
- [25] George Williams and FontForge project contributors, *FontForge* <http://fontforge.github.io/en-US/>
- General purpose documentation:
- [26] *Adobe Font Development Kit for OpenType* <http://adobe.com/devnet/opentype/afdko.html>
- [27] *Adobe base 35 fonts*
Adobe original AFM files: <ftp://ftp.adobe.com/pub/adobe/type/win/all/afmfiles/base35/>
URW replacement: <https://ctan.org/pkg/urw-base35>
- [28] *Adobe Type 1 Font Format* https://partners.adobe.com/public/developer/en/font/T1_SPEC.PDF
- [29] *MATH — The mathematical typesetting table* <https://www.microsoft.com/typography/OTSPEC/math.htm>
- [30] *GUST Font License* <http://www.gust.org.pl/fonts/licenses/GUST-FONT-LICENSE.txt>
<http://tug.org/fonts/licenses/GUST-FONT-LICENSE.txt>
- [31] *OpenType specification (full)* <http://www.microsoft.com/en-ph/download/details.aspx?id=1144>
- [32] *OpenType Feature File Specification* http://www.adobe.com/devnet/opentype/afdko/topic_feature_file_syntax.html
- [33] *Registered features — definitions and implementations (p-t)* <https://www.microsoft.com/typography/otspec/featuretags.htm>
- [34] *SIL Open Font License* <https://scripts.sil.org/OFL>
- [35] *The Unicode Standard 9.0.0, 2016* <http://unicode.org/versions/Unicode9.0.0>
- [36] *The Unicode Standard: A Technical Introduction* <http://unicode.org/standard/principles.html>
- [37] Barbara Beeton, Asmus Freytag, Murray Sargent III, *Unicode Technical Report #25. Unicode Support for Mathematics* <http://unicode.org/reports/tr25/>
- All links above were tentatively accessed 05.07.2015.
- ◊ Bogusław Jackowski
Gdańsk, Poland
`b_jackowski (at) gust dot org dot pl`
 - ◊ Piotr Strzelczyk
Sopot, Poland
`p.strzelczyk (at) gust dot org dot pl`
 - ◊ Piotr Pianowski
Trąbki Wielkie, Poland
`p.pianowski (at) gust dot org dot pl`

Localisation of TeX documents: tracklang

Nicola L. C. Talbot

Abstract

TeX is an excellent typesetting system, but its ancient (in computing terms) origin means that it lags behind modern competition in terms of localisation.

Word processors and spreadsheet applications can query the operating system's localisation-related environment variables to determine how to format information, such as dates, times or currency. If the user is writing a single-language document in their own native language, there's no need to keep stating their language and region every time they create a new spreadsheet or word processor document. Whereas with TeX (in its various formats), users may find themselves having to provide this information repeatedly within a single document.

This article describes the development of the `tracklang` [8] package, which can be used in L^AT_EX or input as a generic TeX file. It attempts to keep track of the localisation setting so that the user doesn't have to redundantly supply information.

1 Introduction

Let's consider two hypothetical people, Alice and Bob. Alice lives in the United Kingdom (UK) and speaks English. Bob lives in Canada and speaks French as his primary language, but is also fluent in English. Alice has her computer set up so that the operating system environment variables include:

`LANG=en_GB.utf8`

`LC_ALL=POSIX`

Bob has something similar, but for some reason he likes to have his messages in English:

`LANG=fr_CA.utf8`

`LC_MESSAGES=en_CA.utf8`

Both Alice and Bob have to send out invoices from time to time. They could just use a spreadsheet which will conveniently look up the localisation variables and format the date using their own regional format (British for Alice and French Canadian for Bob) and will format the currency column according to their region (GBP £ for Alice and CAD \$ for Bob). However, Alice and Bob both want to use L^AT_EX, and they've discovered a package called, say, `easyinvoice` that looks promising.

Alice wants to invoice someone for a DVD costing £5. Rather bizarrely, and with no regard for exchange rates, Bob is coincidentally invoicing someone for a DVD costing C\$5. Both start with the same document:

```
\documentclass{article}
\usepackage[T1]{fontenc}
\usepackage{easyinvoice}
```

```
\begin{document}
\begin{invoice}
\itemrow{DVD}{5}
\end{invoice}
\end{document}
```

In both cases this produces the same result:

Item	Invoice Date: June 14, 2016.
DVD	Price (EUR)
	5

Please pay within 28 days of invoice date.

This isn't suitable for either Alice or Bob. It's closer to Alice's requirements as it's in English, but the currency is incorrect and the date uses the American style. Alice and Bob both remember about the `babel` package [1] and decide to load it before `easyinvoice`. In Alice's case, she does:

```
\usepackage[british]{babel}
```

and Bob does:

```
\usepackage[canadien]{babel}
```

Unfortunately for both of them, this only has a minor change. For Alice, the result is now:

Item	Invoice Date: 14th June 2016.
DVD	Price (EUR)
	5

Please pay within 28 days of invoice date.

For Bob, the result is now:

Item	Invoice Date: 14 juin 2016.
DVD	Price (EUR)
	5

Please pay within 28 days of invoice date.

So in both cases, the only thing that has changed is the date format. The code for `easyinvoice.sty` is as follows:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{easyinvoice}

\providecommand*\@date{\today}
\newcommand{\invoicedatename}[1]{Invoice Date}
\newcommand{\invoiceitemname}[1]{Item}
\newcommand{\invoicepricename}[1]{Price}
\newcommand{\invoicemoneyname}[1]{EUR}
\newcommand{\invoicenote}[1]{Please
pay within 28 days of invoice date.}
```

```
\newcommand*{\itemrow}[2]{\#\#1\#\#2}

\newenvironment{invoice}%
{%
 \par\hfill\invoicedatename: \@date.\par
 \begin{center}%
 \begin{tabular}{lr}
 \invoicetitlename &
 \invoicename\ \\
 (\invoicemoneyname)%
 \end{tabular}%
 \end{center}%
 \end{tabular}%
 \end{center}%
 \invoicemoneyname\par
 \medskip\par
}
\endinput
```

The package author has provided a way of altering the fixed names (by providing commands like `\invoicedatename`) but babel can't alter them (since it's unaware of them) and the `easyinvoice` package author hasn't provided translations. It's therefore necessary for both Alice and Bob to make the necessary changes by redefining the relevant commands. In Alice's case this is just the currency unit:

```
\renewcommand{\invoicemoneyname}{GBP}
```

However Bob needs to redefine all region-sensitive commands.

This is a nuisance for Alice and Bob (especially Bob) and while they can create a template `.tex` file to copy every time they want to create an invoice, there may be other packages they might want to use that likewise need modifications. It's not the best example for Alice and Bob to present to their spreadsheet-using colleagues in a bid to encourage them to switch to L^AT_EX.

2 Adding multi-lingual support to packages

Let's suppose now that the author of the `easyinvoice` package decides to provide some regional support in response to feedback from Alice and Bob. The next version now has some additional lines of code:

```
\newcommand{\invoicebritish}{%
 \renewcommand{\invoicedatename}{Invoice Date}%
 \renewcommand{\invoicetitlename}{Item}%
 \renewcommand{\invoicename}{Price}%
 \renewcommand{\invoicemoneyname}{GBP}%
 \renewcommand{\invoicemoneyblurb}{Please
 pay within 28 days of invoice date.}%
}

\newcommand{\invoicecanadian}{%
 \renewcommand{\invoicedatename}{Date
 de la Facture}%
}
```

```
\renewcommand{\invoicetitlename}{Article}%
\renewcommand{\invoicename}{Prix}%
\renewcommand{\invoicemoneyname}{CAD}%
\renewcommand{\invoicemoneyblurb}{S'il
vous plaît payer dans les 28 jours
suivant la date de facturation.}%
}
```

Now Bob can simply do `\invoicecanadian`, which saves him a few lines of code, but there's not a great deal of difference to Alice who now simply replaces:

```
\renewcommand{\invoicemoneyname}{GBP}
```

with

```
\invoicebritish
```

Alice may be wondering at this point why the package author has set the defaults to English text with European currency. Many of the packages on CTAN are written by a single author, and the original package was simply to help the author perform some task. The author then decided that the package might be useful to others and made it publicly available. It's therefore not too surprising to find that the package defaults match the requirements of the package author. In this case, it might be that the package author is, say, an English speaker living in the Republic of Ireland (RoI).

How can the `easyinvoice` package author be more helpful to Alice and Bob? The package could define options that select the appropriate `\invoice<lang>` command. For example:

```
\DeclareOption{british}{\invoicebritish}
\DeclareOption{canadian}{\invoicecanadian}
\ProcessOptions
```

Now Bob can do:

```
\documentclass{article}
\usepackage[T1]{fontenc}
\usepackage[canadian]{babel}
\usepackage[canadian]{easyinvoice}
```

```
\begin{document}
\begin{invoice}
\itemrow{DVD}{5}
\end{invoice}
\end{document}
```

This is still a bit of a nuisance as Bob has to tell both babel and `easyinvoice` to use French Canadian. In this example, the document has a single language and is for a single region. The localisation is essentially a document-wide setting here, and therefore this seems a valid instance of making it a document class option:

```
\documentclass[canadian]{article}
\usepackage[T1]{fontenc}
\usepackage{babel}
```

```
\usepackage{easyinvoice}

\begin{document}
\begin{invoice}
\itemrow{DVD}{5}
\end{invoice}
\end{document}
```

Now Bob only needs to set this information once per document. Of course, his spreadsheet-using colleagues might point out that they don't need to do it at all, but Bob decides to put up with that.

Bob now remembers that the recipient is in an English-speaking part of Canada, and he decides that perhaps he'd better produce a dual-language invoice, so he tries:

```
\documentclass[canadien,canadian]{article}
\usepackage[T1]{fontenc}
\usepackage{babel}
\usepackage{easyinvoice}

\begin{document}
\selectlanguage{canadien}
\begin{invoice}
\itemrow{DVD}{5}
\end{invoice}

\selectlanguage{canadian}
\begin{invoice}
\itemrow{DVD}{5}
\end{invoice}
\end{document}
```

This produces:

Article	Prix (CAD)
DVD	5

S'il vous plaît payer dans les 28 jours suivant la date de facturation.

Article	Prix (CAD)
DVD	5

S'il vous plaît payer dans les 28 jours suivant la date de facturation.

This hasn't worked for two reasons. The first one being that `easyinvoice` doesn't provide a `canadian` option. This can be added to the package:

```
\newcommand{\invoicecanadian}{%
\renewcommand{\invoicedatename}{Invoice Date}%
\renewcommand{\invoiceitemname}{Item}%
\renewcommand{\invoicename}{Price}%
\renewcommand{\invoicecurrencyname}{CAD}%
\renewcommand{\invocepaymentblurb}{Please
```

```
pay within 28 days of invoice date.}%
}

\DeclareOption{canadian}{\invoicecanadian}
```

Bob's document now produces:

Item	Price (CAD)
DVD	5

Please pay within 28 days of invoice date.

Item	Price (CAD)
DVD	5

Please pay within 28 days of invoice date.

This is because the `easyinvoice` package isn't aware of the language changes. Only the date has changed because that's controlled by `babel`. The language in effect is the last one in the `easyinvoice` options list, as that was the one most recently set.

The `babel` package has hooks that are used when the language is set, such as `\captions<lang>` which redefines all the kernel fixed-text commands. To be more generally helpful, the `easyinvoice` package could test for the existence of `\captionsbritish`, `\captionscanadien` and `\captionscanadian` and add to them. For example, the following code could be added to the `easyinvoice` package:

```
\@ifundefined{captionsbritish}{}
  {\addto\captionsbritish{\invoicebritish}}
\@ifundefined{captionscanadien}{}
  {\addto\captionscanadien{\invoicecanadien}}
\@ifundefined{captionscanadian}{}
  {\addto\captionscanadian{\invoicecanadian}}
```

With this modification, Bob's document now produces:

Article	Prix (CAD)
DVD	5

S'il vous plaît payer dans les 28 jours suivant la date de facturation.

Item	Price (CAD)
DVD	5

Please pay within 28 days of invoice date.

Alice and Bob are now both happy, but the package author might be feeling somewhat less so. What started out as a simple, short package has bloated. Each supported language and region combination requires a block of code in the form:

```
\newcommand{\invoicebritish}{%
\renewcommand{\invoicedatename}{Invoice Date}%
\renewcommand{\invoicename}{Price}%
\renewcommand{\invoicecurrencyname}{CAD}%
\renewcommand{\invocepaymentblurb}{Please
```

```
\renewcommand{\invoiceitemname}{Item}%
\renewcommand{\invoicepricename}{Price}%
\renewcommand{\invoicedcurrencyname}{GBP}%
\renewcommand{\invoicedpaymentblurb}{Please
pay within 28 days of invoice date.}%
}
\DeclareOption{british}{\invoicebritish}
\@ifundefined{captionsbritish}{}%
{\addto\captionsbritish{\invoicebritish}}
```

So far the *easyinvoice* package only supports three language and region combinations. The more options that are added, the more bloated the package becomes and the harder it is to manage it. Another method is needed to trim down this code. The *babel* package stores the names of all loaded languages in `\bbl@loaded`. It's a bit risky using an internal command defined by another package, especially if it's not documented in the user guide. Internal commands are the closest packages can get to declaring private variables. There's no guarantee that they won't change or disappear in future versions, but let's suppose the *easyinvoice* package author decides to take a gamble on it. The three `\@ifundefined` blocks can now be changed from

```
\@ifundefined{captionsbritish}{}%
{\addto\captionsbritish{\invoicebritish}}%
\@ifundefined{captionscanadien}{}%
{\addto\captionscanadien{\invoicedcanadien}}%
\@ifundefined{captionscanadian}{}%
{\addto\captionscanadian{\invoicedcanadian}}%
```

to:

```
\ifdef\bbl@loaded
{%
\@for\@this@lang:=\bbl@loaded\do{%
\ifcsdef{invoice}\@this@lang}{%
\cseappto{captions}\@this@lang}{%
\expandonce
{\csname invoice\@this@lang
\endcsname}}%
}%
\PackageWarning{easyinvoice}{Sorry,
no support for language `@\this@lang'}%
}%
}%
}{}%
```

(The *easyinvoice* author has wisely decided to use the *etoolbox* package [3] to help here.) This method has the added advantage of warning the user if their chosen language isn't supported.

The package options are still declared

```
\DeclareOption{british}{\invoicebritish}
\DeclareOption{canadien}{\invoicedcanadien}
\DeclareOption{canadian}{\invoicedcanadian}
```

in case the user has decided not to use *babel*. For example, Alice might decide that she ought to explicitly set the date (using `\date`) so she has a record of it if she needs to recheck it. (She might have removed the PDF after posting it to tidy up her file system.)

```
\documentclass{article}
\usepackage[british]{easyinvoice}
\date{14th June 2016}
```

```
\begin{document}
\begin{invoice}
\itemrow{DVD}{5}
\end{invoice}
\end{document}
```

This correctly displays the GBP currency.

Betty, from elsewhere in the UK, has discovered the *easyinvoice* package and decides to use it. Unlike Alice, Betty is in the habit of using *babel* with the `UKenglish` option, so she tries it out:

```
\documentclass{article}
\usepackage[UKenglish]{babel}
\usepackage{easyinvoice}

\begin{document}
\begin{invoice}
\itemrow{DVD}{5}
\end{invoice}
\end{document}
```

This produces the error message:

```
Package easyinvoice Warning: Sorry, no
support for language 'UKenglish'
```

and displays the following in the output:

Invoice Date:	14th June 2016.
Item	Price (EUR)
	5

Please pay within 28 days of invoice date.

This is because the *easyinvoice* package doesn't recognise `UKenglish` as a synonym for `british`. A simple fix is to add

```
\newcommand{\invoiceUKenglish}{\invoicebritish}
```

to the package code.

Betty now decides that actually she's going to switch to X_ET_AX and start using the *polyglossia* package [2] instead. Her document is now:

```
\documentclass{article}
\usepackage{polyglossia}
\setmainlanguage[variant=uk]{english}
\usepackage{easyinvoice}
```

```
\begin{document}
\begin{invoice}
\itemrow{DVD}{5}
\end{invoice}
\end{document}
```

The invoice goes back to looking like:

Item	Price (EUR)
DVD	5

Please pay within 28 days of invoice date.

This time `easyinvoice` gives no warning message. Since `babel` hasn't been loaded, `\bbl@loaded` is no longer defined, so can't be iterated over.

The earlier method of testing for the existence of commands like `\captionsbritish` no longer works here, as `polyglossia` only uses the root language name. Thus, although the document has requested the UK variant of English, only `\captionsenglish` is defined.

What should the `easyinvoice` package do in this situation? Testing if `\captionsenglish` is defined doesn't identify the region. The best that can be done is to modify the package option declarations:

```
\DeclareOption{british}{\invoicebritish
  \ifdef{\captionsenglish}
    {\appto{\captionsenglish}{\invoicebritish}}%
  {}%
}
\DeclareOption{canadien}{\invoicecanadien
  \ifdef{\captionsfrench}
    {\appto{\captionsfrench}{\invoicecanadien}}%
  {}%
}
\DeclareOption{canadian}{\invoicecanadian
  \ifdef{\captionsenglish}
    {\appto{\captionsenglish}{\invoicecanadian}}%
  {}%
}
```

This means that Betty now has to do:

```
\documentclass{article}
\usepackage{polyglossia}
\setmainlanguage[variant=uk]{english}
\usepackage[british]{easyinvoice}
```

```
\begin{document}
\begin{invoice}
\itemrow{DVD}{5}
\end{invoice}
\end{document}
```

That is, Betty has to specify her language and region twice in the document. This shouldn't be necessary.

I mentioned earlier the possibility that the fictional author of such an `easyinvoice` package might

be an English speaker in the RoI. What localisation setting is available for users there who need to write a document in English? The `babel` package provides the following English options: `english`, `USenglish` (or `american`), `UKenglish` (or `british`), `canadian`, `australian` and `newzealand`; while `polyglossia` provides: `us` (or `american`), `usmax`, `uk` (or `british`), `australian` and `newzealand`.

Thus, neither `babel` nor `polyglossia` provides a way of identifying the English language used in the RoI. The sensible solution would appear to be to use the closest matching alternative. In this case it's `british` (or the UK synonym) to match the date. Aside from political sensitivities, this doesn't help the `easyinvoice` package because it will assume that the currency should be GBP. There may be other packages the user requires as well that are sensitive to the territory. For example, UTC+1 is generally denoted BST (British Summer Time) in the UK but IST (Irish Summer Time) in the RoI, CET in Jersey or Guernsey, etc.

3 The `tracklang` package

I have a number of packages for which I want to provide regional support, but they can become so bogged down with the code to determine the document language and region settings that they can end up being too high-maintenance to support. Any development around the basic task of the package becomes sidelined in an attempt to support all the various ways in which a user might want to identify their preferences. Are they using `babel` or `polyglossia` or `translator` (provided with `beamer` [4]) or `ngerman` [5] or some other language package that I don't know about?

The aim of the `tracklang` package is to simplify this. It tries to determine what language and regional settings the user has requested, so that it can provide the information to interested packages in a more accessible manner. It doesn't provide translations. It's not an alternative to `babel` or `polyglossia`. It doesn't switch any document settings on. It just attempts to keep track of the user's settings.

3.1 Informing `tracklang` of the document languages

The L^AT_EX file `tracklang.sty` inputs the generic T_EX code file `tracklang.tex`. L^AT_EX users can load the package in the usual way:

```
\usepackage{tracklang}
```

However, there's little need to load it directly in the document preamble as it's intended as a resource for package writers, so it's more likely to be loaded in a package:

```
\RequirePackage{tracklang}
```

The only options it has are language or dialect names or regional identifiers to allow them to be picked up from the document class options.

Non-L^AT_EX users may load the `tracklang.tex` file in the usual way. Pre-version 1.3 required a category code change for the @ character. Version 1.3 added code to automatically set and restore the cat-code for the benefit of non-L^AT_EX users. Version 1.3 also introduced some new commands to make it easier to query and parse the system environment variables `LC_ALL` and `LANG`.

Since generic code has no concept of document class or package options, generic use requires that the document dialects be identified using

```
\TrackPredefinedDialect{(name)}
```

where `(name)` is a dialect label which is recognised by `tracklang`.

For example, here's the start of a L^AT_EX document:

```
\documentclass[british]{article}
\usepackage{tracklang}
```

The analogous plain T_EX is:

```
\input tracklang
\TrackPredefinedDialect{british}
```

There are some synonyms available so, for example, instead of `british` I can use `UKenglish` or `en-GB`. The advantage of `british` and `UKenglish` in the document class options list is that they're also recognised by packages such as `babel`. However, if those packages aren't in use, the ISO form fits in better with global standards.

With version 1.3, you can instead look up your system's language environment variable using

```
\TrackLangFromEnv
```

This first queries `LC_ALL`. If that's unavailable, it then queries `LANG`. Unfortunately Windows stores the locale information in the registry rather than in environment variables. In this case, if `texosquery` [7] has also been loaded (either through `\usepackage` for L^AT_EX users or `\input` for generic use) then `tracklang` will use `texosquery` as a fallback if it fails to get a result with the environment variables. (This will also be used as a fallback for LuaT_EX if the locale is simply identified as the C or POSIX locale.)

Alice has `LANG` set to `en_GB.utf8`, so instead of

```
\TrackPredefinedDialect{british}
```

she can just do

```
\TrackLangFromEnv
```

(Provided either `\directlua` is defined or the shell escape is available.)

The first environment variable to be queried is `LC_ALL`, which Alice has set to `POSIX`. This is not

useful for `tracklang`, (similarly if it had been set to C), so `\TrackLangFromEnv` tries again with `LANG`.

As a by-product, the component parts of the localisation identifier are available in the following commands:

```
\TrackLangEnvLang
```

This contains the language code. For Alice: `en`.

```
\TrackLangEnvTerritory
```

This contains the territory code. For Alice: `GB`.

```
\TrackLangEnvCodeSet
```

This contains the code set. For Alice: `utf8`.

```
\TrackLangEnvModifier
```

This contains the modifier. In Alice's case, this is empty as the modifier isn't present.

The entire value is stored in

```
\TrackLangEnv
```

If this command has already been defined, then `\TrackLangFromEnv` will skip the environment variable query step. For example, no shell escape (or `\directlua`) is performed with:

```
\def\TrackLangEnv{en_GB}
\TrackLangFromEnv
```

The underscore character here has its usual subscript category code. This is the first choice by `tracklang`'s internal parser when trying to split the language code from the region code. It will also allow a hyphen (with category code 12) as the separator:

```
\def\TrackLangEnv{en-GB}
\TrackLangFromEnv
```

and will finally try the underscore character with category code 12.

```
\edef\TrackLangEnv{en\_string_GB}
\TrackLangFromEnv
```

Also, case matters: the language code must be in lower case and the territory code in capitals.

Here's an example plain T_EX document:

```
\input tracklang
\TrackLangFromEnv
Language: \TrackLangEnvLang.
Territory: \TrackLangEnvTerritory.
Codeset: \TrackLangEnvCodeSet.
Modifier: \TrackLangEnvModifier.
\bye
```

If this file is called, say, `myDoc.tex`, then if Alice does:

```
pdftex myDoc
```

Then the resulting PDF contains:

Language en. Territory: GB. Codeset: utf8.
Modifier: .

Similar results are obtained with ε - \TeX , $\text{Xe}\text{\TeX}$ and $\text{Lua}\text{\TeX}$. Unfortunately it doesn't work with the non-extended \TeX :

```
tex myDoc
```

This produces the following warnings:

```
tracklang Warning: \TrackLangQueryEnv
non-operational as can't determine if the
shell escape has been enabled. (Consider
using eTeX or pdfTeX.)
tracklang Warning: \TrackLangFromEnv
non-operational as \TrackLangEnv is empty
Neither \shellescape nor \pdfshellescape are
defined, so tracklang can't determine if the shell escape
is available, and therefore it won't make the attempt.
This avoids the possibility of triggering the error:
! I can't find file `"\kpsewhich --var-value=LC_ALL".
1.2 \input |"\kpsewhich --var-value=LC_ALL"
(Press Enter to retry, or Control-D to exit)
Please type another input file name:
```

If \shellescape / \pdfshellescape is defined but is zero (disabled), the first warning changes to:

```
tracklang Warning: \TrackLangQueryEnv
non-operational as shell escape has been
disabled
```

If the shell escape is disabled, Alice can instead define \TrackLangEnv from the command line:

```
tex "\def\TrackLangEnv{\$LANG}\input myDoc"
```

Alternatively, she can use $\text{Lua}\text{\TeX}$:

```
luatex --no-shell-escape myDoc
```

This now uses \directlua to obtain the environment variable value.

Bob doesn't have LC_ALL set, but he does have LC_MESSAGES . If he wants to query this, he can use:

```
\TrackLangQueryOtherEnv{LC_MESSAGES}
\TrackLangFromEnv
```

This first tries LC_ALL , but if that doesn't yield a result, it then tries the variable name provided in the argument (LC_MESSAGES in this example). If that also doesn't provide a value, it falls back on LANG . The result is again stored in \TrackLangEnv so \TrackLangFromEnv doesn't repeat the environment variable query. The code can be slightly modified to only perform $\text{\TrackLangQueryOtherEnv}$ if \TrackLangEnv hasn't already been defined:

```
\ifx\TrackLangEnv\undefined
  \TrackLangQueryOtherEnv{LC_MESSAGES}
\fi
\TrackLangFromEnv
```

There's a significant difference between directly setting a dialect using $\text{\TrackPredefinedDialect}$ (including implicitly through the document class options) and using \TrackLangFromEnv .

With $\text{\TrackPredefinedDialect}$, an error will occur if an explicit label isn't recognised (or in the case of a document class option, it will be ignored). Whereas with \TrackLangFromEnv , if the language and territory combination is unrecognised, \TrackLang will define a new dialect to represent it.

For example, Jacques from Brussels can use:

```
\TrackPredefinedDialect{fr-BE}
```

since fr-BE is recognised by \TrackLang , but he can't replace fr-BE with en-BE , since that's not a predefined dialect.

However, Jacques can do:

```
\input tracklang
\def\TrackLangEnv{en-BE}
\TrackLangFromEnv
Language: \TrackLangEnvLang.
Territory: \TrackLangEnvTerritory.
Codeset: \TrackLangEnvCodeSet.
Modifier: \TrackLangEnvModifier.
\bye
```

The resulting PDF now shows:

```
Language: en. Territory: BE. Codeset: .
Modifier: .
```

The emphasis here is on reading the locale environment variables such as LANG because it's easy to call kpsewhich from \TeX and capture the output. However, version 1.3 of \TrackLang also introduces a command for parsing a regular language tag. For example:

```
\TrackLanguageTag{hy-Latn-IT-arevela}
```

The next version of texosquery (1.2) will include a new option which can be used to access the locale information in this format:

```
\input texosquery
\input tracklang
\TeXOSQueryLangTag{\langtag}
\TrackLanguageTag{\langtag}
```

3.2 Support for known language packages

The IAT\TeX file tracklang.sty has some awareness of babel , translator , polyglossia and ngerman . After it has input tracklang.tex and processed any options, it then tests if any of the declared options have actually been used. For example:

```
\documentclass{article}
\usepackage[british]{babel}
\usepackage{tracklang}
```

Here british has been passed to babel , not the document class. This means that it's not detected when \TrackLang 's options are processed.

When this occurs, `tracklang` has to go through the pesky process of trying to work out if any of the language packages that it knows about have been loaded. If any have, then `tracklang` needs to work out the language settings. The simplest of these is `ngerman`. If it's been loaded, that just means doing `\TrackPredefinedDialect{ngerman}`

(Similarly for `german`.)

The hardest of these is `polyglossia`, as it currently doesn't keep a list of all the user's selected languages. Instead, `tracklang` needs to iterate through all known languages and check each one to determine if it has been loaded by testing the existence of `\langle lang \rangle @loaded`. (For versions of `tracklang` before 1.3, the iteration was over a hard-coded list of known `polyglossia` languages, but this could miss any new languages that might later be supported, so as of v1.3 the iteration is over all `tracklang`'s declared options, which is a longer list and therefore slower.) There's also no way of determining if a language was loaded with a particular variant, so the regional information can't be determined. These limitations may be addressed in the future, which would make integration with `polyglossia` much easier.

If `babel` has been loaded, then `\bbbl@loaded` should be defined, in which case `tracklang` can iterate through that list and add each loaded language to the list of tracked dialects. In the event that `\bbbl@loaded` isn't defined but `babel` is loaded, `tracklang` will iterate through a list of its own predefined dialects that are available as package options and test if the captions hook exists for that option. As with the above case of `polyglossia`, this is a much longer list.

If `translator` has been loaded, `tracklang` iterates over the internal language list `\trans@languages`.

This is a bit clumsy, but it tidies the mess away from other packages so they don't have to do it.

3.3 Querying `tracklang` for the document languages

All the dialects tracked using the commands in the previous sections are stored by `tracklang` in an internal list. The *root* languages are stored in another list, and any provided ISO codes are also stored.

This section looks at how a package can query this information to determine which localisation settings need to be applied. You can test if any languages are being tracked using:

```
\AnyTrackedLanguages{\langle true \rangle}{\langle false \rangle}
```

For example:

```
\input tracklang
\TrackPredefinedDialect{en-GB}
```

```
\AnyTrackedLanguages{\Yes}{\No}.
```

produces: Yes.

You can iterate over all known dialects using

```
\ForEachTrackedDialect{\langle cs \rangle}{\langle body \rangle}
```

This sets the control sequence given by `\langle cs \rangle` at the start of each iteration and does `\langle body \rangle`. For example:

```
\input tracklang
\TrackPredefinedDialect{en-CA}
\TrackPredefinedDialect{fr-CA}
Dialects:
\ForEachTrackedDialect
  {\thisdialect}{\thisdialect. }
\bye
```

This produces:

Dialects: canadian. canadien.

If only the root language name is given, that will appear in the dialect list. For example:

```
\input tracklang
\TrackPredefinedDialect{english}
Dialects:
\ForEachTrackedDialect
  {\thisdialect}{\thisdialect. }
\bye
```

This produces:

Dialects: english.

In the case of Jacques' unknown combination:

```
\input tracklang
\def\TrackLangEnv{en-BE}
\TrackLangFromEnv
Dialects:
\ForEachTrackedDialect
  {\thisdialect}{\thisdialect. }
\bye
```

The result is now:

Dialects: enBE.

A useful command that can be used within the body of `\ForEachTrackedDialect` is

```
\IfTrackedLanguageFileExists
  {\langle dialect \rangle}{\langle prefix \rangle}{\langle suffix \rangle}
  {\langle found code \rangle}{\langle not found code \rangle}
```

This tests the existence of a file whose name is in the form `\langle prefix \rangle \langle tag \rangle \langle suffix \rangle` where the `\langle tag \rangle` part is determined by the `\langle dialect \rangle`. If a match is found, `\langle found code \rangle` is performed, otherwise `\langle not found code \rangle`. The `\langle not found code \rangle` part is also done if `\langle dialect \rangle` hasn't been added to the list of tracked dialects, or if `\langle dialect \rangle` is empty, but this situation won't occur when used within the `\langle body \rangle` argument of `\ForEachTrackedDialect`.

In the `\CurrentTrackedTag` part, you can obtain the value of `\tag` from

`\CurrentTrackedTag`

This means that within `\CurrentTrackedTag` you can do

`\input{\prefix}\CurrentTrackedTag \suffix}`

Other convenient commands available for use within `\CurrentTrackedTag` are as follows:

`\CurrentTrackedLanguage`

This is set to the root language label (for example, `english` if the dialect is `british`).

`\CurrentTrackedDialect`

This is set to the dialect label (for example, `british`).

This is the same value as `\dialect`.

`\CurrentTrackedRegion`

This is set to the region code (ISO 3166-1), if known for this dialect (empty otherwise).

`\CurrentTrackedIsoCode`

This is set to the ISO code (either 639-1 or 639-2) for the root language, if known (empty otherwise).

`\IfTrackedLanguageFileExists` guesses what the `\tag` should be based on whether or not the dialect has an ISO 3166-1 country code, and if the root language has an ISO 639-1 or 639-2 language code. The first guess that matches a file name on TEX's path will provide the value of `\tag`.

For example, for the `british` dialect, the tries will be in the order: `british` (dialect label), `en-GB` (ISO 639-1 and ISO 3166-1), `eng-GB` (ISO 639-2 and ISO 3166-1), `en` (ISO 639-1), `eng` (ISO 639-2), `GB` (ISO 3166-1), `english` (language label). Whereas, for the `UKenglish` dialect, the tries will be in the order: `UKenglish` (dialect label), `en-GB`, `eng-GB`, `en`, `eng`, `GB`, `english` (language label). It's therefore best not to use a file naming scheme that has dialect labels as the `\tag` part unless there's a particular reason to treat synonymous dialect labels differently.

Synonyms for the root language are treated as regionless dialects; so, for example, with `francais` the order is just: `francais` (dialect label), `fr`, `fra`, `french` (language label). Compare this with the regionless `french` language where the order is: `french` (dialect label), `fr`, `fra`, `french` (language label). Here the dialect label is identical to the language label so the fourth guess either won't be tried (because a match has already been found) or will fail (because if it did match, it would've been picked up on the first guess).

3.4 Example package using `tracklang`

Returning to the example `easyinvoice` package, it no longer needs to define all the language options, as

they'll be picked up by `tracklang`. The code that changes the commands that produce the fixed text (such as `\invoicedatename`) will go in separate files, which will use the naming scheme

`easyinvoice-\tag.1df`

This fits in with `\IfTrackedLanguageFileExists`, where `\prefix` is `easyinvoice-` and `\suffix` is `.1df`.

These files can simply be input using `\input`, but it's useful to provide an equivalent to commands like `\RequirePackage` and `\ProvidesPackage`. The new improved version of `easyinvoice` is now:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{easyinvoice}
\RequirePackage{etoolbox}
\RequirePackage{tracklang}

% If user hasn't requested a language
% try LC_ALL or LANG environment variable
\AnyTrackedLanguages{}{\TrackLangFromEnv}

% Set defaults
\newcommand{\invoicedatename}{Invoice Date}
\newcommand{\invoiceitemname}{Item}
\newcommand{\invoicepricename}{Price}
\newcommand{\invoicecurrencyname}{EUR}
\newcommand{\invoicepaymentblurb}{Please
pay within 28 days of invoice date.}

\providecommand*{\@date}{\today}
\newcommand*{\ProvidesInvoiceResource}[1]{%
  \ProvidesFile{easyinvoice-\#1.1df}%
}
\newcommand*{\RequireInvoiceResource}[1]{%
  \ifcsundef{ver@easyinvoice-\#1.1df}%
  {%
    \input{easyinvoice-\#1.1df}%
  }%
  {}%
}
\newcommand*{\RequireInvoiceDialect}[1]{%
  \IfTrackedLanguageFileExists{\#1}%
  {easyinvoice-\#1.1df}%
  {\#1}%
  \%
  \PackageWarning{easyinvoice}%
  {No support for dialect `#\#1'}%
}
\ForEachTrackedDialect{\this@dialect}{%
  \RequireInvoiceDialect{\this@dialect}%
}

% Main package code:
\newcommand*{\itemrow}[2]{\#\#1\#\#2}
```

```
\newenvironment{invoice}%
{%
  \par\hfill\invoicedatename: \@date.\par
  \begin{center}%
  \begin{tabular}{lr}
  \invoicetemname &
  \invoicepricename\
  (\invoicecurrencyname)%
  \end{tabular}%
  \end{center}%
  \invoicedatepaymentblurb\par
  \medskip\par
}%
\endinput
```

At the start, this loads `tracklang`. If it hasn't picked up any localisation, an attempt is made to query the environment variables `LC_ALL` or `LANG`.

Now for the LDF files. The language settings are provided in a file that uses the root language label in the `<tag>` part. The territory settings are provided in a file that uses the ISO country code in the `<tag>` part.

For example, `easyinvoice-english.1df`:

```
\ProvidesInvoiceResource{english}
\providecommand*\{englishinvoice}\{%
  \renewcommand{\invoicedatename}{Invoice Date}%
  \renewcommand{\invoicetemname}{Item}%
  \renewcommand{\invoicepricename}{Price}%
  \renewcommand{\invoicedatepaymentblurb}{Please
  pay within 28 days of invoice date.}%
}%
\englishinvoice

% polyglossia check: \captions<root language>
\ifundef\captionsenglish
{%
  babel check: \captions<dialect>
  \ifcsundef\captions\CurrentTrackedDialect\{}%
    \csgappto\captions\CurrentTrackedDialect\{}%
      {\englishinvoice}%
  \}%
}%
\gappto\captionsenglish{\englishinvoice}%
\endinput
```

The territory file `easyinvoice-GB.1df`:

```
\ProvidesInvoiceResource{GB}
\providecommand*\{GBinvoice}\{%
  \renewcommand{\invoicecurrencyname}{GBP}%
}%
\GBinvoice
\endinput
```

The dialect settings are stored in a file where the `<tag>` part is formed from the ISO language code

and country code. This file needs to load the root language LDF file and the territory LDF file.

For example, `easyinvoice-en-GB.1df` can look like this:

```
\ProvidesInvoiceResource{en-GB}
\RequireInvoiceResource{english}
\RequireInvoiceResource{GB}

\ifundef\captionsenglish
{%
  \ifcsundef\captions\CurrentTrackedDialect\{}%
    \csgappto\captions\CurrentTrackedDialect\{}%
      {\GBinvoice}%
  \}%
}%
\gappto\captionsenglish{\GBinvoice}
```

If, for example, `babel` has been loaded with the `british` option, this means the `\captionsbritish` hook now includes

```
\englishinvoice
\GBinvoice
```

With `polyglossia`, these are in `\captionsenglish` (but `tracklang` must be informed that the `en-GB` dialect is required).

The LDF files rely on `\CurrentTrackedDialect` being set, which it will be when the file is loaded within `\IfTrackedLanguageFileExists`. If an attempt is made to use `\RequireInvoiceResource` when this command hasn't been set, there'll be a problem with the caption hooks.

Although `\RequireInvoiceResource` could include a check for this, `\RequireInvoiceDialect` is a more general purpose command, so it's better to restrict `\RequireInvoiceResource` to use within the resource files:

```
% Default behaviour outside of resource files:
% generate an error and ignore the argument.
\newcommand*\{noop@RequireInvoiceResource\}[1]\{%
  \PackageError{easyinvoice}
  {\string\RequireInvoiceResource\space only
  permitted within invoice resource files.}%
}%
\let\RequireInvoiceResource
\noop@RequireInvoiceResource

% Actual behaviour of \RequireInvoiceResource
% used within resource files.
\newcommand*\{@RequireInvoiceResource\}[1]\{%
  \ifcsundef\ver@easyinvoice-\#1.1df\}%
{%
  \input{easyinvoice-\#1.1df}%
}%

```

```

{}%
}

% General use command.
\newcommand*{\RequireInvoiceDialect}[1]{%
  \IfTrackedLanguageFileExists{#1}{%
    {easyinvoice-}%
    {.ldf}%
    {%
      % Enable \RequireInvoiceResource so that it can
      % be used in resource files.
      \let\RequireInvoiceResource
        \ORequireInvoiceResource
      % Load resource file.
      \RequireInvoiceResource\CurrentTrackedTag
      % Disable \RequireInvoiceResource.
      \let\RequireInvoiceResource
        \noop@RequireInvoiceResource
    }%
    {%
      \PackageWarning{easyinvoice}%
      {No support for dialect `#1'}%
    }%
}

```

(This could be extended to add code prohibiting \RequireInvoiceDialect within resource files.)

3.5 Using a `tracklang`-enabled package

With this new arrangement, Alice can do:

```

\documentclass{article}
\usepackage{easyinvoice}

\begin{document}
\begin{invoice}
\itemrow{DVD}{5}
\end{invoice}
\end{document}

```

As long as she has the shell escape enabled or she's using Lua^TE_X, the result is:

Invoice Date: June 14, 2016.	
Item	Price (GBP)
DVD	5

Please pay within 28 days of invoice date.

This still uses the default US date style because easyinvoice doesn't make any changes to \today. Alice could load datetime2 [6] as well, but it might be helpful for easyinvoice to do this automatically.

The datetime2 package also uses tracklang, so it seems the best solution would be to just load it with

\RequirePackage{datetime2}

However, datetime2 defaults to numeric ISO date style. The `useregional` option is required to switch on the regional support. However, it's best not to

use the optional argument of \RequirePackage as it can result in a package option clash error if it has already been loaded. It's possible that the user has already loaded datetime2 with their own preferred style, and easyinvoice shouldn't interfere with this.

Thus, a better approach is to use:

```

\PassOptionsToPackage
  {useregional=text}{datetime2}
\RequirePackage{datetime2}

```

Maybe easyinvoice should also allow the user to pass options to datetime2 within easyinvoice's option list:

```

\PassOptionsToPackage
  {useregional=text}{datetime2}
\DeclareOption*{%
  \PassOptionsToPackage
    {\CurrentOption}{datetime2}}
}
\ProcessOptions
\RequirePackage{datetime2}

```

So the new improved easyinvoice package now works just fine for Alice; however, Betty, who's using polyglossia, still needs to explicitly indicate her region:

```

\documentclass[en-GB]{article}
\usepackage{polyglossia}
\setmainlanguage[variant=uk]{english}
\usepackage{easyinvoice}

```

```

\begin{document}
\begin{invoice}
\itemrow{DVD}{5}
\end{invoice}
\end{document}

```

Since tracklang has detected polyglossia's english setting, \TrackLangFromEnv isn't used. To help here, the easyinvoice package could provide an option to insist on querying the environment variable even if there are other languages present. For example:

```
\DeclareOption{env}{\TrackLangFromEnv}
```

This means that Betty can now do:

```

\documentclass{article}
\usepackage{polyglossia}
\setmainlanguage[variant=uk]{english}
\usepackage[env]{easyinvoice}

```

```

\begin{document}
\begin{invoice}
\itemrow{DVD}{5}
\end{invoice}
\end{document}

```

which saves her the redundant document option.

Another possibility is to add a test for the existence of \TrackLangEnv, regardless of whether or not any languages have been detected:

```
\ifdef\TrackLangEnv
  {\TrackLangFromEnv}
  {\AnyTrackedLanguages{}{\TrackLangFromEnv}}
```

This will add to the document's dialect list if it's not already present. In the case of `en-GB`, the dialect is considered a synonym for `british` but not a synonym of `UKenglish`, even though both dialects have the same language and country codes.

If neither `babel` nor `polyglossia` are loaded, the last dialect in the list will be the one in effect. For example, the following document adds `fr-CA` to the list of tracked dialects:

```
\documentclass[fr-CA]{article}
\usepackage{easyinvoice}

\begin{document}
\begin{invoice}
\itemrow{DVD}{5}
\end{invoice}
\end{document}
```

However, if the document is compiled using
`pdflatex '\def\TrackLangEnv{en-GB}\input{myDoc}'`

Then the `en-GB` setting will override `fr-CA`.

Seán from the RoI also decides to use `easyinvoice` but he prefers to have the date include the time and zone information:

```
\documentclass{article}
\usepackage{easyinvoice}
\date{\DTMnow}

\begin{document}
\begin{invoice}
\itemrow{DVD}{5}
\end{invoice}
\end{document}
```

He has `LC_ALL` set to `en_IE` and as he has shell escape enabled (or is using `LuaLaTeX`) this is added to the list of tracked dialects. In this case, only the `english` LDF file is loaded, not the `GB` or `en-GB` files. This means that the currency is unchanged, which is fine for Seán.

Since `datetime2` has been loaded with the regional option on, its `en-IE` style is automatically set, so `UTC+1` is displayed as `IST`, as shown below:

Invoice Date: 14th June 2016 1:10pm IST.	
Item	Price (EUR)
DVD	5

Please pay within 28 days of invoice date.

This is a simple solution for all the countries that use the Euro currency; however, multilingual documents that switch from one territory to another

need help to return to the default. This can be done by defining a command for setting the country defaults (in `easyinvoice.sty`). For example:

```
\newcommand*\countrydefaultinvoice{%
  \renewcommand{\invoicecurrencyname}{EUR}%
}
```

Now the root language LDF file needs to add this to the captions hooks:

```
\ifundef\captionsenglish
{%
  \ifcsundef{captions\CurrentTrackedDialect}{%
    \csgappto{captions\CurrentTrackedDialect}{%
      \englishinvoice
      \countrydefaultinvoice
    }%
  }%
}%
{%
  \gappto\captionsenglish{%
    \englishinvoice
    \countrydefaultinvoice
  }%
}
```

Remember that `easyinvoice-en-GB.1df` adds to the hook after this, so `\GBinvoice` will override this default setting if the dialect is `en-GB`.

Let's not forget about Bob in Canada. He needs `easyinvoice-CA.1df`:

```
\ProvidesInvoiceResource{CA}
\providecommand*\CAinvoice{%
  \renewcommand{\invoicecurrencyname}{CAD}%
}
\CAinvoice
\endinput
```

The Canadian English file `easyinvoice-en-CA.1df`:

```
\ProvidesInvoiceResource{en-CA}
\RequireInvoiceResource{english}
\RequireInvoiceResource{CA}
```

```
\ifundef\captionsenglish
{%
  \ifcsundef{captions\CurrentTrackedDialect}{%
    \csgappto{captions\CurrentTrackedDialect}{%
      \CAinvoice
    }%
  }%
}%
{%
  \gappto\captionsenglish{\CAinvoice}
\endinput
```

The French Canadian file `easyinvoice-fr-CA.1df` is similar:

```
\ProvidesInvoiceResource{fr-CA}
```

```
\RequireInvoiceResource{french}
\RequireInvoiceResource{CA}

\ifundef\captionsfrench
{%
  \ifcsundef{captions\CurrentTrackedDialect}{%
    {}%
  }%
  \csgappto{captions\CurrentTrackedDialect}{%
    \CAinvoice
  }%
}%
}%
{\gappto\captionsfrench{\CAinvoice}}
\endinput
```

This needs `easyinvoice-french.1df`:

```
\ProvidesInvoiceResource{french}
\providecommand*\frenchinvoice{%
  \renewcommand{\invoicedatename}{Date
de la Facture}%
\renewcommand{\invoiceitemname}{Article}%
\renewcommand{\invoicepricename}{Prix}%
\renewcommand{\invocepaymentblurb}{S'il
vous pla\^{\i}t payer dans les 28 jours
suivant la date de facturation.}%
}
\frenchinvoice

\ifundef\captionsfrench
{%
  \ifcsundef{captions\CurrentTrackedDialect}{}{%
    \csgappto{captions\CurrentTrackedDialect}{%
      {}%
      \frenchinvoice
      \countrydefaultinvoice
    }%
  }%
}%
{\gappto\captionsfrench{%
  \frenchinvoice
  \countrydefaultinvoice
}}%
}%
}%
\endinput
```

This suits Jacques just fine as, like Seán, he only needs the root language file since he wants the country default.

Meanwhile Hank, over in the USA, only needs `easyinvoice-US.1df`:

```
\ProvidesInvoiceResource{US}
\providecommand*\USinvoice{%
  \renewcommand{\invoicecurrencyname}{USD}%
}
\USinvoice
\endinput
```

and `easyinvoice-en-US.1df`:

```
\ProvidesInvoiceResource{en-US}
\RequireInvoiceResource{english}
\RequireInvoiceResource{US}

\ifundef\captionsenglish
{%
  \ifcsundef{captions\CurrentTrackedDialect}{%
    {}%
  }%
  \csgappto{captions\CurrentTrackedDialect}{%
    \USinvoice
  }%
}%
}%
{\gappto\captionsenglish{\USinvoice}}
\endinput
```

Now Seán decides to provide an Irish Gaelic version `easyinvoice-irish.1df`:¹

```
\ProvidesInvoiceResource{irish}
\providecommand*\irishinvoice{%
  \renewcommand{\invoicedatename}{D\^{\i}ata
Sonraisc}%
\renewcommand{\invoiceitemname}{M\^{\i}r\^{\i}e}%
\renewcommand{\invoicepricename}{Praghais}%
\renewcommand{\invocepaymentblurb}{Tabhair
\^{\i}oc laistigh de 28 l\^{\i}a \^{\i}o dh\^{\i}ata
an tsonraisc.}%
}
\irishinvoice

\ifundef\captionsirish
{%
  \ifcsundef{captions\CurrentTrackedDialect}{}{%
    \csgappto{captions\CurrentTrackedDialect}{%
      {}%
      \irishinvoice
      \countrydefaultinvoice
    }%
  }%
}%
{\gappto\captionsirish{%
  \irishinvoice
  \countrydefaultinvoice
}}%
}%
}%
\endinput
```

Again, he doesn't need to worry about providing a ga-IE LDF file since he wants the default currency.

Now Ciaran in Northern Ireland discovers this and tries to produce an invoice in Irish Gaelic:

¹ If the Irish and French text here are a bit iffy, it just goes to show how unwise it is to expect someone to provide translations for languages they don't know or aren't fluent in. They tend to cheat and use a popular translation website.

```
\documentclass[ga-GB]{article}
\usepackage{easyinvoice}

\begin{document}
\begin{invoice}
\itemrow{DVD}{5}
\end{invoice}
\end{document}
```

To his surprise, although the date is in Irish and the currency is GBP, the text is in English:

```
Invoice Date: 14 Meitheamh 2016.
Item    Price (GBP)
DVD      5
```

Please pay within 28 days of invoice date.

An inspection of the transcript shows that only the GB LDF file has been loaded. The problem here is that there's no ga-GB file, so the first LDF file to match *<tag>* is the GB file.

The solution is to add `easyinvoice-ga-GB.ldf`:

```
\ProvidesInvoiceResource{ga-GB}
\RequireInvoiceResource{irish}
\RequireInvoiceResource{GB}

\ifundef\captionsirish
{%
  \ifcsundef{captions\CurrentTrackedDialect}%
  {}%
  {%
    \csgappto{captions\CurrentTrackedDialect}{%
      \GBinvoice
    }%
  }%
}%
{\gappto\captionsirish{\GBinvoice}}
\endinput
```

For any new LDF file, no change is required to the code in `easyinvoice.sty`. As long as the files are placed on `TEX`'s path, `easyinvoice` will detect them.

4 Language packages

A *language package* is one that actually sets the document language (hyphenation patterns, redefining fixed name commands such as `\contentsname`, possibly set fonts and so on; e.g., `babel`). The `easyinvoice` package is an example of a package that needs to know the document language. How can language package authors help packages like `easyinvoice`?

Let's suppose I want to write a language package that sets up a document for Ancient Greek. If this is for single language documents (just Ancient Greek and nothing else), all I need to do is add the following lines to my package:

```
\input{tracklang}%
\TrackPredefinedDialect{greek}
\SetTrackedDialectModifier{greek}{ancient}
```

I've used `\input` rather than `\RequirePackage` here to skip the tests for `babel`, `polyglossia` etc. There's no need to test for the possible language packages because this is the language package. (There's a test in `tracklang.tex` to prevent multiple loading.)

In this case the label `greek` is recognised by `tracklang`, but if it weren't, I could replace the above with:

```
\input{tracklang}
\TrackLocale{el@ancient}
```

This has the ISO 639-1 code (`el`) with a modifier (`ancient`). `\TrackLocale` works in the same way as `\TrackLangFromEnv` but doesn't use any of the `\TrackLangEnv...` commands. If I prefer to use an IETF language tag I can use `\TrackLanguageTag` instead.

As of version 1.3, `tracklang` recognises nearly 200 languages with ISO 639-1 or 639-2 codes. However, if my root language isn't included in that list, I can add it using:

```
\AddTrackedLanguage{greek}
\AddTrackedIsoLanguage{639-1}{el}{greek}
\AddTrackedIsoLanguage{639-2}{ell}{greek}
```

or for a regional dialect:

```
\AddTrackedDialect{greekCY}{greek}
\AddTrackedIsoLanguage{639-1}{el}{greek}
\AddTrackedIsoLanguage{639-2}{ell}{greek}
\AddTrackedIsoLanguage{3166-1}{CY}{greekCY}
```

If my package is providing support for multiple languages or dialects with caption hooks in the form `\captions<lang>`, then I also need to use `\AddTrackedDialect` if `<lang>` isn't recognised by `tracklang`.

```
% user has requested "ancientgreek":
\AddTrackedDialect{ancientgreek}{greek}
\AddTrackedIsoLanguage{639-1}{el}{greek}
\AddTrackedIsoLanguage{639-2}{ell}{greek}
% define caption hook:
\def\captionsancientgreek{%
  ...
}
```

In this case, `tracklang` doesn't recognise 'ancientgreek', but since it does recognise 'greek' and knows the ISO codes for it, I can actually just do:

```
% user has requested "ancientgreek":
\AddTrackedDialect{ancientgreek}{greek}
\AddTrackedLanguageIsoCodes{greek}
% define caption hook:
\def\captionsancientgreek{%
  ...
}
```

Now if a user wants to use this language package and `easyinvoice`, then `easyinvoice` can find out the

document language without having to know anything about my Ancient Greek package.

Note that the above code is all generic with the exception of

```
\input{tracklang}
```

which needs to be replaced with:

```
\input tracklang
```

for plain \TeX . (This syntax also works with \LaTeX .)

5 Summary

5.1 Document authors

Load the language package before any packages that use `tracklang`. For example:

```
\documentclass{article}
\usepackage[british]{babel}
\usepackage{easyinvoice}
```

If the region is needed but isn't provided by the language package (or no language package required), use the ISO format. For example:

```
\documentclass[en-IE]{article}
\usepackage[english]{babel}
\usepackage{easyinvoice}
```

Generic use (query operating system):

```
\input tracklang
\TrackLangQueryEnv
\input genericinvoice
```

5.2 Package writers

\LaTeX packages need to use

```
\RequirePackage{tracklang}
```

to pick up `babel`, etc., options. Generic use:

```
\input tracklang
```

In either case, if no languages found, query OS:

```
\AnyTrackedLanguages{}{\TrackLangFromEnv}
```

For package `foo`, put the language or regional commands in separate `foo-<tag>.ldf` files, which are loaded using

```
\def\RequireFooResource#1{\input foo-#1.ldf}
\def\RequireFooDialect#1{%
  \IfTrackedLanguageFileExists{#1}{foo-}{.ldf}{%
    \RequireFooResource\CurrentTrackedTag}%
  {}% no support warning
}
\ForEachTrackedDialect{\thisdialect}{%
  \RequireFooDialect\thisdialect
}%
```

6 Conclusion

The `tracklang` package provides a way for package authors to conveniently query the document language settings to make it easier to provide multilingual support. The generic code allows it to be used with multiple \TeX formats, and the \LaTeX code additionally detects and supports common language packages.

`\IfTrackedLanguageFileExists` allows a modular approach so that localisation support can be added and maintained independently of the main package code. This shifts the expectation that a single person (the package author) should not only be able to write \TeX code but also be fluent in all known languages and dialects, to a community-based approach with the package author maintaining the base package code and any interested volunteers providing the benefit of their own local knowledge.

References

- [1] Javier Bezos and Johannes L. Braams. The `babel` package, 2016. ctan.org/pkg/babel.
- [2] François Charette and Arthur Reutenauer. `polyglossia`: an alternative to the `babel` package, 2016. ctan.org/pkg/polyglossia.
- [3] Philipp Lehman. The `etoolbox` package, 2011. ctan.org/pkg/etoolbox.
- [4] Vedran Milić, Joseph Wright, and Till Tantau. The `beamer` class, 2015. ctan.org/pkg/beamer.
- [5] Bernd Raichle. Kurzbeschreibung `german.sty` und `n german.sty`, 1998. ctan.org/pkg/german, ctan.org/pkg/n german.
- [6] Nicola Talbot. The `datetime2` package, 2016. ctan.org/pkg/datetime2.
- [7] Nicola Talbot. `texosquery`: Query OS information from \TeX , 2016. ctan.org/pkg/texosquery.
- [8] Nicola Talbot. The `tracklang` package, 2016. ctan.org/pkg/tracklang.

◊ Nicola L. C. Talbot
 School of Computing Sciences
 University of East Anglia
 Norwich Research Park
 Norwich NR4 7TJ
 United Kingdom
 N.Talbot (at) uea dot ac dot uk
<http://www.dickimaw-books.com>

Glisternings: Index headers; Numerations; Real number comparison

Peter Wilson

Gaul as a whole is divided into three parts.

De Bello Gallico, JULIUS CAESAR

The aim of this column is to provide odd hints or small pieces of code that might help in solving a problem or two while hopefully not making things worse through any errors of mine.

La dernière chose qu'on trouve faisant un ouvrage, est de savoir celle qu'il faut mettre la première.

The last thing one knows in constructing a work is what to put first.

Pensées, BLAISE PASCAL

1 Index headers

In the manual for the `memoir` class the headers for the index included the first and last main index entries on the page in question. The technique I used was fairly simple and can be applied to any document whose index is generated via the `MakeIndex` program. It consists of defining a macro in the document's preamble (or a package) and a simple `MakeIndex` style file [4, 8, 16].

Below is an example of a document file, where the new macro is `\idxmark` and the `fancyhdr` package [15] is used for specifying the page headers and footers.

`\idxmark{<entry>}` prints `<entry>` and also sets both the marks to `<entry>`, where `\leftmark` resolves to the first mark on the page and `\rightmark` to the last mark on the page.

```
\documentclass[...twoside]{...}
\usepackage{makeidx,fancyhdr}
\newcommand*\idxmark[1]{\#1\markboth{\#1}{\#1}}
\pagestyle{fancy}
%% set up general fancy headers/footers
\makeindex
\begin{document}
... % many \index{...} ...
\clearpage
%% fancy headers/footers for index pages
\fancyhead{}
\fancyhead[LE,RO]{%
    \rightmark\space---\space\leftmark}
\fancyfoot{}
\fancyfoot[C]{\thepage}
\printindex
\end{document}
```

Peter Wilson

The index is typically set in two columns and L^AT_EX's marking system did not always function well in this case; in the past, including the `fixltx2e` package was needed to fix that, but nowadays it is included by default.

In the example the general header/footer styles are set in the preamble using the `fancyhdr` package facilities. The (special) headers and footers for the index are set just before the index commences. When looking for something in a book I flick it half closed only seeing the outer portion of each page. Consequently, in this case I have put the first and last index entries on the page at the outer of the headers where they are easy to see and the page number, which is not particularly important when looking through the index, centered at the foot of the page.

Here are the essentials of the `MakeIndex` style file for the application. For each main entry, `<entry>` in the input `.idx` file it outputs `\idxmark{<entry>}` in the `.ind` file which is used by `pdflatex` to print the index.

```
% MakeIndex style file flindex.ist
% output main entry as <entry> as:
%   \item \idxmark{<entry>}
item_0 "\n\\item \\idxmark{"
delim_0 ", "
% not forgetting subitem
\item_X1 "} \n \\subitem "
```

The `memoir` manual, `memman.pdf`, has an index that is over 40 pages long with each alphabetic section headed by the letter (A, B, C, etc.). The `hyperref` package is used to enable PDF bookmarks and Lars Madsen developed code so that the index letter subheads would appear in the bookmark listing. Here is the skeleton of how this was done, where the `\doidxbookmark{<head>}` macro does the work. The `<head>` argument is an index subhead and the macro adds it to the list of bookmarks at one level below the Index entry. The default `MakeIndex` generated subhead for entries that do not commence with an alphanumeric is 'Symbols', but I felt that 'Alphabetics' was a better subhead. `\doidxbookmark` prints the appropriate subhead centered in a bold font and adds it to the bookmark list.

```
...
\usepackage{ifpdf}
\ifpdf
    \usepackage[pdftex,
        plainpages=false,
        pdfpagelabels,
        bookmarksnumbered,
        colorlinks,
        ocgcolorlinks,
    ]{hyperref}
\else
```

```
\usepackage[plainpages=false,
            pdfpagelabels,
            bookmarksnumbered,
            colorlinks,
            ]{hyperref}

\fi
\makeatletter
\newcommand*\doidxbookmark[1]{%
  \def\@tempa{\Symbols}\def\@tempb{\#1}%
  \centering\bfseries \ifx\@tempa\@tempb
    Alphabetics
    \phantomsection%
    \belowpdfbookmark{Alphabetics}{Alphabetics-idx}%
  \else
    #1%
    \phantomsection%
    \belowpdfbookmark[#1]{#1-idx}%
  \fi
  \vskip\baselineskip\par}
\makeatother
...
\begin{document}
...
\clearpage
\pdfbookmark[Index]{Index}
\phantomsection
\printindex
\end{document}
```

The MakeIndex style file must be written so that it wraps the `\doidxbookmark` around the subheads, like this:

```
% MakeIndex style file flindex.ist
% output main entry ...
% Wrap and uppercase head letters
headings_flag 1
heading_prefix "\doidxbookmark{"
heading_suffix "}"
```

And here is the sequence of commands to generate the indexed `file.tex` document

```
> pdflatex file
> makeindex -s flindex.ist file
> pdflatex file
```

As yet a child, nor yet a fool to fame,
I lisped in numbers, for the numbers came.

An Epistle to Dr Arbuthnot,
ALEXANDER POPE

2 Numerations

2.1 Sorted lists

Reza wrote to ctt [12] saying that he had developed code that would sort the items in a `description` en-

vironment and asked if there was a way to enumerate the entries.

I had no idea that you could sort items within L^AT_EX but it appears that Nicola Talbot's incredible `datatool` package [14] enables you to do that, and much more.

Here is Reza's original code, to which I have added some comments to try and indicate what is happening and changed the environment name to distinguish it from other later code:

```
%%% Reza's code
\usepackage{datatool}
\newcommand{\sortitem}[2]{%
  % start a new row in the db
  \DTLnewrow{list}%
  % add key/value to row
  \DTLnewdbentry{list}{label}{#1}%
  % add another key/value to the row
  \DTLnewdbentry{list}{description}{#2}%
}
\newenvironment{sorteddesc}{%
  % use or create a db called 'list'
  \DTLifdbexists{list}
    {\DTLcleardb{list}}{\DTLnewdb{list}}%
}{%
  % at the end of the environment sort the
  % db in ascending order of the label
  \DTLsort{label}{list}%
  % start a description environment
  \begin{description}%
    % iterate through the db,
    % picking out the keys/values
    \DTLforeach*{list}{%
      {\theLabel=label,\theDesc=description}%
      \item[\theLabel]\theDesc
    }%
  \end{description}%
}
```

As an example,

```
\begin{sorteddesc}
\sortitem{zz}{description of zz}
\sortitem{mm}{description of mm}
\sortitem{aa}{description of aa}
\end{sorteddesc}
```

results in:

aa description of aa
mm description of mm
zz description of zz

Reza wanted to know if there was a way to enumerate the entries. That is, so the output would look like:

1. aa description of aa
2. mm description of mm
3. zz description of zz

Christian Anderson [1] responded with the following based on using the `enumerate` environment (I have changed the name of the sorting environment to distinguish it from the other proposals):

```
\newenvironment{sortedenum}{%
  \DTLifdbexists{list}%
  { \DTLcleardb{list}\{\DTLnewdb{list}\}%
  }%
  \DTLsort{label}{list}%
  \begin{enumerate}%
    \DTLforeach*{list}%
    {\theLabel=label,\theDesc=description}{%
      \item \theLabel\ \theDesc% original
      % bolds the descriptive label
      \item \textbf{\theLabel}\ \theDesc%
    }%
  \end{enumerate}%
}
```

As an example of Christian's proposal

```
\begin{sortedenum}
\sortitem{zz}{description of zz}
\sortitem{mm}{description of mm}
\sortitem{aa}{description of aa}
\end{sortedenum}
```

results in:

1. aa description of aa
2. mm description of mm
3. zz description of zz

It was unclear as to how Raza wanted the combination of number and label to be typeset. Christian indicated how the label could be typeset in a bold font, but the number would still be set in the normal font.

Alan Munn [9] also responded as follows, taking advantage of the `\DTLcurrentindex` macro from `datatool` (again I have changed the name of the sorting environment).

```
\newenvironment{sorteditemdesc}{%
  \DTLifdbexists{list}%
  { \DTLcleardb{list}\{\DTLnewdb{list}\}%
  }%
  \DTLsort{label}{list}%
  \begin{description}%
    \DTLforeach*{list}%
    {\theLabel=label,\theDesc=description}{%
      \item[\normalfont
        \DTLcurrentindex.\ }
        \theLabel]\theDesc%
    }%
  \end{description}%
}
```

As an example of Alan's proposal

```
\begin{sorteditemdesc}
\sortitem{zz}{description of zz}
\sortitem{mm}{description of mm}
```

```
\sortitem{aa}{description of aa}
\end{sorteditemdesc}
```

results in:

1. aa description of aa
2. mm description of mm
3. zz description of zz

Alan also indicated how the number could be set in bold to match the label. By suitable application of `\normalfont` both could instead be set in the normal font. In this sense Alan's solution is slightly more general than Christian's.

2.2 Autotab

Jeremy wrote to `ctt` [7]:

I want to create a table and I want the first column of the table to have incrementing numbers ... I would like to avoid having to manually enter the numbers in the first column. Is there a way to do this automatically?

Alan Munn responded with [10]:

```
\usepackage{booktabs}
\usepackage{array}
\newcounter{rownum}
\newcolumntype{N}[%
  >{\stepcounter{rownum}\therownum.\ }]{}
\newcommand*{\resetrownum}[%
  \setcounter{rownum}{0}}
\begin{tabular}{N1l} \toprule
\multicolumn{1}{>{\resetrownum}l}{I} \\
& A & B \\ \midrule
& blah & blah \\
& foo & foo \\ \bottomrule
\end{tabular}
```

I	A	B
1.	blah	blah
2.	foo	foo

In the same thread Heiko Oberdiek noted that the `\resetrownum` could be taken out of the `tabular`:

```
\resetrownum
\begin{tabular}{N1l} \toprule
\multicolumn{1}{l}{I} & A & B \\ \midrule
& blah & blah \\ \bottomrule
& foo & foo
\end{tabular}
```

In a later thread on a similar but extended topic Romildo posed [13],

I want to typeset an enumeration list in a tabular format, building a table with automatic numbered items and sub-items in different rows. [An example layout followed.]

Jean-François Burnol [3] responded with:

```
\newcounter{bitmi}
\renewcommand{\thebitmi}{%
  \arabic{bitmi}}
\newcounter{bitmii}[bitmi]
\renewcommand{\thebitmii}{%
  \thebitmi.\arabic{bitmii}}
\newcounter{bitmiii}[bitmii]
\renewcommand{\thebitmiii}{%
  \thebitmii.\arabic{bitmiii}}
\newcommand{\bitm}{\stepcounter{bitmi}%
  \hbox to 1.5em{\thebitmi.\hfil}}
\newcommand{\bsubitm}{\stepcounter{bitmii}%
  \hbox to 1.5em{%
    \hbox to 2.5em{\thebitmii\hfil}}}
\newcommand{\bsubsubitm}{\stepcounter{bitmiii}%
  \hbox to 4em{%
    \hbox to 3.5em{\thebitmiii\hfil}}}

\begin{tabular}{lllll}
Subject & Class & Total & Notes \\ 
\bitm First topic & 2 & 2 & a, b \\ 
\bitm Second topic & 12 & 14 & \\ 
  \bsubitm Aaaa & & & b, c, d \\ 
    \bsubsubitm M1 & & & a \\ 
      \bsubsubitm M2 & & & b, e, f \\ 
  \bsubitm Bbbb & & & a, b \\ 
\bitm Third topic & \ldots & & \\ 
\end{tabular}
```

Subject	Class	Total	Notes
1. First topic	2	2	a, b
2. Second topic	12	14	
2.1 Aaaa			b, c, d
2.1.1 M1			a
2.1.2 M2			b, e, f
2.2 Bbbb			a, b
3. Third topic			...

Just a little later Peter Flynn proposed [5]:

```
\usepackage{array}
\newcounter{topic}
\renewcommand{\thetopic}{\arabic{topic}}
\newcounter{topici}[topic]
\renewcommand{\thetopici}{%
  \thetopic.\arabic{topici}}
\newcounter{topicii}[topici]
\renewcommand{\thetopicii}{%
  \thetopici.\arabic{topicii}}
\newcommand{\topic}[1]{%
  \stepcounter{topic}%
  \vrule height1.2em width0pt\thetopic. \&
  \multicolumn{3}{l}{\#1}}
\newcommand{\subtopic}[1]{%
  \stepcounter{topici}%
  \vrule height1em width0pt \thetopici \&
  \multicolumn{2}{l}{\#1}}
\newcommand{\subsubtopic}[1]{%
```

```
\stepcounter{topicii} \& \thetopicii\#1

\begin{tabular}{rrrlr<{\quad}r<{\enspace}l}
\multicolumn{4}{l}{\textbf{Subject}} & & \\
\multicolumn{1}{r}{\textbf{Class}} & & \multicolumn{1}{r}{\textbf{Total}} & & \\
\multicolumn{1}{r}{\textbf{Notes}} & & & & \\
\topic{First topic} & 2 & 2 & a, b & \\
\topic{Second topic} & 12 & 14 & & \\
  \subtopic{Aaaa} & & & b, c, d & \\
    \subsubtopic{M1} & & & a & \\
      \subsubtopic{M2} & & & b, e, f & \\
  \subtopic{Bbbb} & & & a, b & \\
\topic{Third topic} & \ldots & & & \\
\end{tabular}
```

Subject	Class	Total	Notes
1. First topic	2	2	a, b
2. Second topic	12	14	
2.1 Aaaa			b, c, d
2.1.1 M1			a
2.1.2 M2			b, e, f
2.2 Bbbb			a, b
3. Third topic			...

Jean-François's approach to me is the simpler and easier of the two as he designed it so that all the main entries go into the first column and internally uses empty \hboxes to indent the several (sub) item levels. Peter's approach is more complex in that it involves several \multicolumns to control the (sub) topic indentations; he also uses zero-width vertical rules to enable different vertical spacing between the topic levels. My feeling is that the combination of Jean-François's horizontal positioning and Peter's vertical adjustments might be closest to an optimum solution to Romildo's needs.

Tenants of life's middle state,
Securely placed between the small and great.

Tirocinium, WILLIAM COWPER

3 Real number comparison

On *ctt* Pluto noted that \ifnum could be used to compare integer values and wondered if there was a similar method for comparing real numbers [11]. This led to a spirited discussion involving several people and over 50 postings at my last count. The two that struck me the most were from Donald Arseneau and 'GL'.

For demonstration purposes assume that the following are defined:

```
\def\fourfive{4.5}
```

```
\def\fivefive{5.5}
\def\sixseven{6.7}
\def\threetwo{3.2}
```

Donald gave concise and elegant code,¹ shown later, that could be used like this:

```
\ifrnum <condition> \then... \else... \fi
as in:
\ifrnum 4.5 > 5.5
\then 4.5 is larger than 5.5
\else 4.5 is not larger than 5.5\fi. \\
\ifrnum \sixseven > \threetwo
\then \sixseven\ is larger than \threetwo
\else \sixseven\ is not larger than
\threetwo \fi.
```

4.5 is not larger than 5.5.
6.7 is larger than 3.2.

Here is Donald Arseneau's code [2]:

```
\let\then\iffalse
\def\gobblejunk#1\delimeter{}
\def\ifrnum#1\then{\ifdim
  \ptlt\ptgt\ppeq #1pt\gobblejunk<=>\delimeter}
\def\ptlt#1<{\#1pt<}
\def\ptgt#1>{\#1pt>}
\def\ppeq#1={\#1pt=}
```

GL also provided code, shown later, that could be used like this:

```
\unless\Realnums\ifnum <condition>
  \Then ... \else ... \fi
```

as in:

```
\unless\ifnum\Realnums\fourfive>\fivefive
  \Then \fourfive\ is not larger than \fivefive
  \else \fourfive\ is larger than
  \fivefive\fi. \\
\unless\ifnum\Realnums 6.7 > 3.2
  \Then 6.7 is not larger than 3.2
  \else 6.7 is larger than 3.2\fi.
```

4.5 is not larger than 5.5.
6.7 is larger than 3.2.

In GL's original posting [6], which was in response to Donald, he reused, modified, and extended Donald's code. I have renamed some of the macros in GL's code so that the two sets are distinct, which means that both styles can be used in the same document, as I have done here.

```
\def\gobblejunk#1\delimeter{}
\def\Realnums#1\Then{\dimexpr
  \Ptlt\Ptgt\Ppeq #1pt\gobblejunk<=>\delimeter}
\def\Ptlt#1<{\#1pt<\dimexpr}
\def\Ptgt#1>{\#1pt>\dimexpr}
\def\Ppeq#1={\#1pt=\dimexpr}
```

The `\unless` macro is defined in ε -*TEX*, which all recent *LATEX* systems automatically utilise. Personally I find the `\unless` construct hard to get my mind around; it's the reverse of the traditional `\if...`

References

- [1] Christian Andersen. Re: Sorted items in description environment. `comp.text.tex`, 10 October 2010.
- [2] Donald Arseneau. Re: `\ifnum` for real numbers. `comp.text.tex`, 16 October 2010.
- [3] Jean-François Burnol. Re: Tabular with enumerated items and sub-items in different rows. `comp.text.tex`, 27 April 2011.
- [4] Pehong Chen and Michael A. Harrison. Index preparation and processing. *Software Practice and Experience*, 19(8):897–915, September 1988. <http://mirror.ctan.org/indexing/makeindex/paper>.
- [5] Peter Flynn. Re: Tabular with enumerated items and sub-items in different rows. `comp.text.tex`, 27 April 2011.
- [6] GL. Re: `\ifnum` for real numbers. `comp.text.tex`, 16 October 2010.
- [7] Jeremy. Table with auto-incrementing column. `comp.text.tex`, 11 April 2011.
- [8] Frank Mittelbach and Michel Goossens. *The LATEX Companion*. Addison Wesley, second edition, 2004. ISBN 0-201-36299-6.
- [9] Alan Munn. Re: Sorted items in description environment. `comp.text.tex`, 10 October 2010.
- [10] Alan Munn. Re: Table with auto-incrementing column. `comp.text.tex`, 11 April 2011.
- [11] Pluto. `\ifnum` for real numbers. `comp.text.tex`, 10 October 2010.
- [12] Reza. Sorted items in description environment. `comp.text.tex`, 10 October 2010.
- [13] Romildo. Tabular with enumerated items and sub-items in different rows. `comp.text.tex`, 27 April 2011.
- [14] Nicola L.C. Talbot. `datatool`: Databases and data manipulation, 2016. <http://ctan.org/pkg/datatool>.
- [15] Piet van Oostrum. Page layout in *LATEX*, 2016. <http://ctan.org/pkg/fancyhdr>.
- [16] Peter Wilson. The `memoir` class for configurable typesetting, 2016. <http://ctan.org/pkg/memoir>.

◊ Peter Wilson
 12 Sovereign Close
 Kenilworth, CV8 1SQ UK
 herries dot press (at)
 earthlink dot net

¹ Which I cannot interpret for you.

Messing with endnotes

David Walden

The two journals with which I am involved, *IEEE Annals of the History of Computing* and *TUGboat*, both prefer endnotes to footnotes. The *TUGboat* editors don't like footnotes because they create complications with the two-column format. The editors of the *Annals* allow no deviation from their specified style which comprises a single endnote list of both notes and references in order of use. In books I have typeset and self-published, I also have mostly used endnotes rather than footnotes to avoid dealing with page breaks in the context of footnotes.

With references in footnotes or in endnotes, one needs sometimes to refer to an earlier note to avoid repeating the full reference. This is handled by putting a label in the footnote or endnote call, e.g.,

```
\footnote{\label{uniquename}Note text.}
```

and then inserting commands such as

```
\textsuperscript{\ref{uniquename}}
```

in the main text where it is necessary to reference the same note again. One could define a macro

```
\def\ReRef#1{\textsuperscript{\ref{#1}}}
```

in order to have less to type in those other instances of the same reference.

You might ask, "Why not use BibTeX?". Well, I used it for a couple of big projects and found its formatting more tedious to use and not much of a labor saver compared to doing my own brute force thing. Nevertheless, I have used BibTeX when notes and references were required to be mixed in a single end-of-document list, by adding .bib entries of the form

```
@misc{NoteA, note="Whatever..."}
```

When not using BibTeX (for either footnotes or endnotes), I don't like to have the long author-title-publisher-etc. items in the flow of the main text. Also, I like at least the bibliographic items to be available for me to peruse in order of first author's last name. Thus, I have taken to putting the bibliographic data near the beginning of my .tex or .ltx source file, or perhaps in its own file included with any other files I include (\include) near the beginning of my document. An example follows.

```
\def\createnote#1#2{%
  \expandafter\newcommand\csname en#1%
  \endcsname{#2}}}

\createnote{Akera}{Atsushi Akera,
Voluntarism and the Fruits of
Collaboration: The IBM User Group,
Share, ..., pp.^710--736.}
```

```
\createnote{Armer}{Paul Armer,
SHARE---A Eulogy to Cooperative
Effort, \textit{Annals of the History...}}
```

The \createnote command in the above example defines the content of a future endnote (I am using the endnote package) that can be summoned by a macro call. For example, the first call above to \createnote defines a macro named \enAkera. (The "en" prefix is to reduce the chance of unanticipated conflicts. For some insight into the use of csname and \expandafter, see Amy Hendrickson's paper in the TUG 2012 proceedings.¹) At the appropriate place later in my document I create and reference the endnote with a command such as \endnote{\enAkera}.

If I expect another reference to the same endnote, I could do the initial creation with a command such as \endnote{label{en:akera}\enAkera} and reference the endnote again with a command such as \textsuperscript{\ref{en:Akera}}. If I want two endnotes at the same place, e.g., here,^{2,3} I put the following after "here":

```
\endnote{\enAkera}{$^{\{ , \}}$}\endnote{\enArmer}
```

Rather than explicitly including a \label in the first call to an endnote used more than once, I can create a macro that includes both the label and the macro call to use a particular endnote:

```
\def\bionote#1{\endnote{%
  \label{en:#1}\csname en#1\endcsname}}
```

to be called, e.g., like \bionote{enBright}⁴ (given that \createnote had been used earlier to define \enBright) and which can be referenced later with \ref{en:Bright} in another endnote.^{5,6}

Confession: Only after the above mechanisms were mostly developed and this written was I reminded that I could have used BibTeX for the above without the usual name/title/publisher fields by only using @misc and typing my full bibliographic entries into the note field, as I have done with \createnote.

Notes

¹Amy Hendrickson, "The joy of \csname...\endcsname". *TUGboat* 33:2, 2012. tug.org/TUGboat/tb33-2/tb104hendrickson.pdf

²Atsushi Akera, Voluntarism and the Fruits of Collaboration: The IBM User Group, Share, ..., pp. 710–736.

³Paul Armer, SHARE—A Eulogy to Co-op Effort, *Annals of the History of Computing*, ... April 1980, pp. 122–129.

⁴Herbert Bright, Computer User Groups, *Annals of the History of Computing*, vol. 12, no. 1, 1990, pp. 56–61.

⁵See note 4 again.

⁶Thanks to Karl Berry for his help with this note.

◇ David Walden
walden-family.com/texland

Tracing paragraphs

Udo Wermuth

Abstract

The program **T_EX** provides more than a dozen control words for diagnostic and debugging purposes. Some of them are used often, others handle special tasks and are less frequently applied. In the latter case falls the parameter `\tracingparagraphs` that seems to be a hidden gem. This article explains what the parameter triggers if set and how an author can use the trace data to check and improve his text.

1 Introduction

The **T_EX** software, described in *T_EX: The Program* [8], implements several control sequences to show information about its work. The commands and parameters form a set of powerful tools to help diagnose errors. **T_EX** itself contains nine primitive integer parameters for tracing ([6, p. 273]) and four primitive show commands ([6, p. 279]). The plain format defines additional macros ([6, p. 364]).

The tracing parameters might be classified into different groups: Some look at the settings of the installation, like `\tracingstats`, others are used mainly for developers, like `\tracingmacros`, and some (or all) can be used to get a better understanding how **T_EX** operates. For example, the parameter `\tracingparagraphs` gives detailed insights into the inner workings of **T_EX**'s line-breaking algorithm.

Four tracing parameters are optional, i.e., their value might be ignored by a working **T_EX** program ([6, p. 303]) without violating one of the conditions which allow a program to carry the name **T_EX**, called the TRIP test [7]. For example, **T_EX**'s two control sequences `\tracingstats` and `\tracingparagraphs` are this kind of parameter. They were deemed optional as the code to implement them slows down the program even when they are not in use, i.e., set to their default value 0. That might have been annoying in the early days of **T_EX**, but today with faster machines the effect is small and so most implementations of **T_EX** provide them. Speed is only one aspect as some parameters trigger a huge amount of output to be written in the log file of the run. The setting `\tracingstats=1` adds only eight lines to the log file, but setting `\tracingparagraphs=1` increases the size of the log file significantly as every paragraph in the scope of this parameter is copied at least once into the log file, accompanied by several lines of trace data.

Udo Wermuth

I don't remember how long I played with the parameter `\tracingparagraphs` when I learned **T_EX**. After I gained some experience with **T_EX** I probably read only pages 98–99 of *The T_EXbook* [6], a couple of double dangerous-bend paragraphs, and did not see how to benefit from the output in my work. (Somehow I missed the hint of a real world application on page 317 of [6].) Other tracing parameters, for example, `\tracingmacros`, became an important tool to diagnose problems. Years later I read the protocol of a Q&A session with Donald E. Knuth ([12] or [13], Ch. 32) and I realized that the parameter `\tracingparagraphs` is a powerful tool too.

Section 2 gives a high-level overview of **T_EX**'s line-breaking algorithm and its parameters and in section 3 the format of the trace data for this algorithm is explained. Section 4 gives examples how I make use of the trace and section 5 shows some aspects of `\looseness` that makes it difficult to extract precisely from the trace data the range of lines in which a paragraph can be typeset without violating the current parameter settings.

All the figures except Figs. 8, 9, 10, and 12 represent data from a previous run of this article; all examples are typeset and traced by **T_EX** during the last compilation.

2 **T_EX**'s line-breaking algorithm

This is not the place to present the details of **T_EX**'s sophisticated line-breaking algorithm but as the parameter `\tracingparagraphs` creates a trace of this algorithm, an overview will be given.

The procedure tries to break the text into lines of a given length. This length is usually `\hsize` but also `\leftskip`, `\rightskip`, `\hangindent`, or `\parshape` must be considered. Each line gets a *badness* value which is calculated as a function of the change of the available white space, i.e., the glue, which is necessary to place the content of the line into the given length. A *finite* badness is an integer between 0 and 9999, larger values are considered to be *infinite* and are not distinguished anymore ([6], p. 97). The badness is approximately $\min(100 \times r^3, 10000)$ if the ratio of used to available amount of change is called r .

Based on the badness each line is assigned a *fitness class*. A line is called

- C0. *very loose* if the badness value is 100 or more and the glue has to stretch;
- C1. *loose* if the badness value is between 13 and 99 and the glue has to stretch;
- C2. *decent* if the badness value is between 0 and 12;

C3. *tight* if the badness value is 13 or more and the glue has to shrink.

A line break can occur only at certain points; there are five possibilities ([6], p. 96). A line can be broken at

- B1. *glue*, i.e., at white space, if a non-discardable item (not glue, kern, penalty or a math switch) appears before the glue;
- B2. a *kern*, if it is followed by glue;
- B3. *math-off*, i.e., at the end of a formula, if it is followed by glue;
- B4. a *penalty* which is either entered directly into the text as an indication how desirable a break at this point is or inserted by *TeX* automatically, for example, in a formula;
- B5. a *discretionary break* when *TeX* splits a word either at an explicit or an inserted implicit hyphen.

Note that *TeX* controls white space in math mode; in this mode B1 and B2 are not used.

Certain line breaks get a penalty based on the following parameters, listed here with their default values which the plain *TeX* format sets:

- P1. *\exhyphenpenalty* (default 50) which is used if a break occurs after an explicit hyphen, i.e., a hyphen that is present in the input;
- P2. *\hyphenpenalty* (default 50) which is applied at an automatically inserted hyphen;
- P3. *\binoppenalty* (default 700) which is applied if a formula is broken after a binary operation;
- P4. *\relpenalty* (default 300) which is applied if a formula is broken after a relation symbol.

And there is a special penalty called *\linepenalty* (default 10) that is applied to every line.

Finally, each line gets a value called *demerits* by which *TeX* rates the constructed lines and sets the breakpoints ([6], pp. 97–98). *TeX*'s goal is to select those line breaks that minimize the sum of the line demerits. (*TeX*'s decision can be changed via the integer parameter *\looseness* ([6], p. 103) to select a set of line breaks that might result in a different number of lines for the paragraph.) Demerits combine two aspects: $d = d_1 + d_2$. The first summand d_1 is based on badness and penalty. The formula by which *TeX* calculates the demerits d_1 follows. It shows the special role that the *\linepenalty*, let's call it l , plays. If b stands for the badness of a line and p for the penalty assigned to the break then

$$d_1 = (l + b)^2 + \begin{cases} \text{sign}(p) p^2, & -10000 < p < 10000 \\ 0, & p \leq -10000. \end{cases}$$

No line break occurs if $p \geq 10000$, and $p \leq -10000$ represents a forced break.

The second aspect d_2 is the sum of fixed values:

- D1. *\adjdemerits* (default 10000) is added either to the second line if adjacent lines fall in one of the fitness class pairs (very loose, decent), (very loose, tight), or (loose, tight) or if the first line is very loose.
- D2. *\doublehyphendemerits* (default 10000); it is added to the second line if two consecutive lines end with a discretionary break.
- D3. *\finalhyphendemerits* (default 5000) which is added to the last line if the second-last line ends with a discretionary break.

Now all but one ingredient of the algorithm has been described. The last item is a limit for the badness which the algorithm uses to decide if a line is acceptable. *TeX* knows two limits ([6], p. 96):

- T1. *\pretolerance* (default 100) which is used as the limit in *TeX*'s attempt to break the paragraph without hyphenation of words (breaks are still allowed at explicit hyphens, i.e., a ‘-’ or a ‘\-’);
- T2. *\tolerance* (default 200) which is used as the limit when hyphenation of words is allowed.

The line-breaking algorithm tries in up to three passes to cut the paragraph into lines whose badness values are less than the current limit. In the first pass the limit T1 counts and no words are given to *TeX*'s hyphenation algorithm. If this pass fails then a second pass with the limit T2 and word hyphenation is made. This pass outputs the paragraph even if it fails, except if the dimen *\emergencystretch* has a positive value. In the first case an overfull line with infinite badness is constructed. In the second case the failed second pass is followed by a third pass with word hyphenation, badness limit T2, and additional stretchability per line given by the value of the dimen *\emergencystretch*. This pass may fail too but then either the value of the dimen must be increased or “the line-breaking task is truly impossible” ([6], p. 107).

3 Format of *\tracingparagraphs*'s output

The *TeXbook* [6] explains on pages 98–99 the main aspects of the trace data. The full details are contained in *Computers & Typesetting*, Volume B [8] in §§ 813–890 together with the code for general printing routines like §§ 174–175, and 245.

The single assignment *\tracingparagraphs=1* triggers if trace data is written in the log file. Here is an overview of the kind of data that is output:

- *optional header* which identifies for which pass of the line-breaking algorithm the output is written.

It is one of the words `@firstpass`, `@secondpass`, or `@emergencypass`. The headers of the first and second pass are not output if the limit of T1 in section 2, the parameter `\pretolerance`, prevents the first pass, i.e., if it is negative (see § 863). It would be nice to add a hook to the log file in this situation, let's say `@hyphenationpass` to signal in a unique way the start of the trace data. But such a change violates the TRIP test [7] although it changes only the log file.

- *break candidates* which are considered by the algorithm as there is a valid way to break the line at this point using a previously found feasible breakpoint. The output has the form “`@<w> via @<m> b=<x> p=<y> d=<z>`” (§ 856) where the placeholders `<x>`, `<y>`, and `<z>` are the values of the badness of the line, the penalty for the break (see P1–P4) if applicable, and the demerits of the line, whose calculation uses the `\linepenalty` and adds the values of D1–D3 if the conditions are met. The `@<m>` documents the feasible breakpoint, after which the current line starts (the value 0 stands for the start of the paragraph). The first parameter `<w>` indicates the kind of break: It is empty if the break occurs at glue between words; otherwise it is (see B2–B5) `\kern`, `\math`, `\penalty`, `\discretionary`, or, at the end of the paragraph, `\par`. When the badness values `<x>` are calculated for an emergency pass, the values represent the data that the line-breaking algorithm uses to get the demerits, i.e., one of `TEX`'s input values for its rating function. The “real” values for the badness as “seen in the output” depend heavily on the used stretchability given by the dimension `\emergencystretch`, but these values are not shown in the trace data. See below for an example.

Principally, three types must be distinguished:

- *inter-word breaks* that are line breaks between words or symbols, i.e., the cases B1–B4.
- *discretionary breaks* (case B5) indicated by the word `\discretionary` for `<w>`, which signals that the line break occurs within a word. Then pre-break and post-break information must be considered to construct the contents of the line.
- *end-of-par breaks* indicated by `\par`. This shows that the line break algorithm was able to process the whole paragraph. `TEX` rates the end of a paragraph as a forced break and assigns therefore a penalty of -10000 , which does not add to the demerits (see the formula for the calculation of demerits in section 2).

Note it is possible to have several break candidates at the end of a line for different feasible breakpoints.

- *feasible breakpoint* which gives the best way to break the paragraph up to this point using the current settings of the line-breaking algorithm for one of the break candidates that appear above this feasible breakpoint. The output in the log file is the string “`@<n>: line <a>. <c> t=<d> -> @<m>`” (§ 846) where `<n>` is the new sequence number of the current feasible breakpoint, and `<m>` states the number of the feasible breakpoint which the new breakpoint needs as the previous line break. The content data between these two breakpoints is then line number `<a>`. It belongs to the fitness class `` (range is 0–3 (§ 817); name is given in `C`), and ends with a hyphen if `<c>` is ‘-’. The value `<d>` states the total demerits of the whole paragraph up to this line, i.e., it is the sum of the `<z>` values of the break candidates for the set of lines ending with this feasible breakpoint.

Note it is not yet determined if this feasible breakpoint will be used to construct the paragraph. The best end-of-par break names the previous feasible breakpoint for the last feasible breakpoint.

The final feasible breakpoints are treated as having a hyphen as the value of `<c>` (§ 829).

- *content data* which is the text seen by the algorithm (§ 857). It is split in small parts as the breakpoints are listed in the output too. In passes that try to hyphenate the words all hyphenation points of `TEX`'s hyphenation algorithm are inserted.

The trace data ends with an empty line (§ 245). Note: Except for the content data all trace lines start with the symbol `@`.

Final remark: Values for the badness are sometimes stated as * which means that it is *infinite* according to `TEX`'s rules. For demerits such an asterisk means that the calculation was not performed because of certain forced conditions (§ 856).

The format of the trace lines is rather terse and a lot of trace lines are written even if they do not contribute to the final line breaks. An example will help to understand the above stated description of the data.

Example 1: `TEX` input

```
\tracingparagraphs=1
\noindent
Note: {\$1 pretolerance\$} is \the\pretolerance\
and {\$1 tolerance\$} is \the\tolerance, the
{\$it hsize\$} is~\the\hsize.
```

This is a nonsense text to serve as a constructed example that shows all kind of trace lines. It contains inline mathematics and text in columns. The formula $\$2 \times 2^2 = 8\$$ is simple mathematics as well as formula $\$\\sqrt{3} \\of 8 = 2\$$ or what do you think? Now a declaration or

```
definition for a three columns tabbing
environment is made.
\settabs 3 \columns \+&&End of example:\cr
```

TeX output

Note: *pretolerance* is 100 and *tolerance* is 200, the *hsize* is 225.0pt.

This is a nonsense text to serve as a constructed example that shows all kind of trace lines. It contains inline mathematics and text in columns. The formula $2 \times 2^2 = 8$ is simple mathematics as well as formula $\sqrt[3]{8} = 2$ or what do you think? Now a declaration or definition for a three columns tabbing environment is made.

End of example: □

Note: The small rectangle at the end of the previous line indicates the end of an example.

The trace data was written in the log file; here are all trace lines with numbers for identification.

Example 1 continued: Log file contents

```
1. @firstpass
2. \ninerm Note: \ninesl pretolerance \ninerm
   is 100 and \ninesl tolerance \ninerm is
   200, the \nineit hsize
3. @kern via @00 b=2 p=0 d=144
4. @01: line 1.2 t=144 -> @00
5. \ninerm is 225.0pt.
6. @par via @01 b=0 p=-10000 d=100
7. @02: line 2.2- t=244 -> @01
8.
9. @firstpass
10. []\ninerm This is a nonsense text to serve
    as a constructed
11. @ via @00 b=23 p=0 d=1089
12. @01: line 1.1 t=1089 -> @00
13. @secondpass
14. []\ninerm This is a non-sense text to serve
    as a con-structed
15. @ via @00 b=23 p=0 d=1089
16. @01: line 1.1 t=1089 -> @00
17. ex-
18. @discretionary via @00 b=27 p=50 d=3869
19. @02: line 1.3- t=3869 -> @00
20. am-ple that shows all kind of trace lines.
   It con-tains in-
21. @discretionary via @01 b=38 p=50 d=14804
22. @discretionary via @02 b=0 p=50 d=12600
23. @03: line 2.2- t=16469 -> @02
24. @04: line 2.3- t=15893 -> @01
25. line
26. @ via @02 b=91 p=0 d=10201
27. @05: line 2.3 t=14070 -> @02
28. math-e-mat-ics and text in col-umns. The
   for-mula
29. @ via @03 b=137 p=0 d=31609
30. @ via @04 b=137 p=0 d=31609
31. @06: line 3.0 t=47502 -> @04
32. $2 \ninesy ^B
33. @penalty via @03 b=0 p=700 d=490100
34. @penalty via @04 b=0 p=700 d=490100
35. @penalty via @05 b=123 p=700 d=517689
36. @07: line 3.2 t=505993 -> @04
37. \ninerm 2[] =
38. @penalty via @05 b=10 p=500 d=250400
39. @08: line 3.2 t=264470 -> @05
40. 8$ is sim-ple math-e-mat-ics as well as
   for-mula
41. @ via @06 b=57 p=0 d=4489
42. @09: line 4.1 t=51991 -> @06
43. $[] [] =
44. @penalty via @06 b=72 p=500 d=266724
45. @penalty via @07 b=1 p=500 d=250121
46. @010: line 4.3 t=314226 -> @06
47. 2$
48. @math via @07 b=2 p=0 d=144
49. @math via @08 b=130 p=0 d=29600
50. @011: line 4.0 t=294070 -> @08
51. or
52. @ via @08 b=7 p=0 d=289
53. @012: line 4.2 t=264759 -> @08
54. what do you think? Now a dec-la-ra-tion or
55. @ via @09 b=31 p=0 d=1681
56. @013: line 5.1 t=53672 -> @09
57. def-i
58. @discretionary via @09 b=4 p=50 d=2696
59. @discretionary via @010 b=119 p=50 d=29141
60. @014: line 5.2- t=54687 -> @09
61. -
62. @discretionary via @09 b=20 p=50 d=13400
63. @discretionary via @010 b=82 p=50 d=20964
64. @015: line 5.3- t=65391 -> @09
65. ni-
66. @discretionary via @010 b=14 p=50 d=13076
67. @discretionary via @011 b=106 p=50 d=15956
68. @016: line 5.0 t=310026 -> @011
69. tion
70. @ via @010 b=4 p=0 d=196
71. @ via @011 b=2 p=0 d=10144
72. @ via @012 b=107 p=0 d=23689
73. @017: line 5.0 t=288448 -> @012
74. for
75. @ via @012 b=0 p=0 d=100
76. @018: line 5.2 t=264859 -> @012
77. a
78. @ via @012 b=25 p=0 d=1225
79. @019: line 5.3 t=265984 -> @012
80. three col-umns tab-bing en-vi-ron-ment is
81. @ via @013 b=57 p=0 d=4489
82. @020: line 6.1 t=58161 -> @013
83. made.
84. @par via @014 b=48 p=-10000 d=8364
85. @par via @015 b=10 p=-10000 d=5400
86. @par via @016 b=0 p=-10000 d=15100
87. @par via @017 b=0 p=-10000 d=10100
88. @par via @018 b=0 p=-10000 d=100
89. @par via @019 b=0 p=-10000 d=100
90. @par via @020 b=0 p=-10000 d=100
91. @021: line 7.2- t=58261 -> @020
92. @022: line 6.3- t=63051 -> @014
93.
```

□

As expected, the trace starts with *@firstpass* for the first paragraph. Line 2 is the content data preceded by a *\ninerm*, which was added by TeX's routines (§ 174); it was not part of the input. As you see all font switching commands are spelled out

explicitly with the control sequence that *T_EX* associates with the requested font. Line 3 outputs the first break candidate; it is a break at the italic correction and so <w> is \kern. The badness is 2, i.e., the line is decent, penalty is 0 and therefore the demerits are $(10 + 2)^2 = 144$. That the line is decent can be seen in line 4 of the listing as a “.2” appears after the line number (see C2). Line 6 documents an end-of-par break, so the first pass was successful. As explained above the <c> is ‘-’ merely because of the end of the paragraph; it does not indicate that the final line ends with an hyphen. The penalty is -10000 to mark a forced break; this value is ignored as stated in the formula for the calculation of demerits and therefore the demerits of the second line are $(10 + 0)^2 = 100$. The total demerits are the sum of the line demerits: $144 + 100 = 244$ (see line 7).

The data of the second paragraph is much more interesting. Again the header line of the first pass is printed (line 9), but only one feasible breakpoint is found and output. This means *T_EX*’s algorithm was unable to find a second break candidate, so it stops this pass, and starts the second pass, which outputs its header line (line 13). Note however, that the content data in line 10 has in front of the font information, which is inserted as described above, the construction `[]`. This stands for output that *T_EX* cannot show; in this case it is the white space created by the indentation. This is the normal behavior for all non-printable items (see § 175). In the second pass hyphenation is tried so *T_EX* shows all hyphenation points in the words by inserting hyphens. Compare line 14 with line 10: *T_EX* has placed hyphens in the words “nonsense” and “constructed”. The breakpoint in the lines 17–19 is a discretionary break. In line 18 the penalty is 50, the value of \hyphenpenalty. In line 19 the 1.3- states that it is a tight line ending in an hyphen.

Let’s look at some interesting points without going through all the details of the trace.

- Lines 21 and 22 show that more than one break candidate can occur but they must use different feasible breakpoints after the “via”.
- In lines 33–35 and in lines 38 & 44–45 penalty breaks are shown in math mode. In the first set the break occurs after a binary operation and the \binoppenalty is applied. The break in the second set is made after a relation and the \relpenalty is used.
- An example for a line break after a math-off is given in lines 47–50.
- Lines 57ff. show a discretionary break in the word “definition” which contains the ligature “fi”. An implicit discretionary break is used for

the ligature with the pre-break text “f-”, the post-break text “i” and the no-break text “fi”. Both pre-break and post-break text are stated in the content data of line 57. After feasible breakpoint 14 just a hyphen is added to the line.

- The end-of-line break candidates in the lines 84–90 signal the successful completion of the second pass. Lines 88–90 seem to be equivalent judged by the data in the lines, but the path via feasible breakpoint 20 has the lowest total demerits and therefore it is stated in the feasible breakpoint @021.
- Feasible breakpoint @022 gives an alternative path via feasible breakpoint @014. Its total demerits are higher than for feasible breakpoint @021 therefore it is not used by *T_EX*. But the number of lines is lower and so it would be a valid path if the author states \looseness=-1 (see example 2 below).
- The trace ends with an empty line. It is shown here but the other examples will omit it.

The paragraph is now built from bottom to top: The last line is between @020 and @021, its content is “made” (see line 83). The second last line starts at @013 with the concatenation of the content in lines 57, 61, 65, 69, 74, 77, and 80. The third last line begins at @09, the next at @06, then at @04, the second line of the paragraphs starts at @01 and the first at the beginning of the paragraph, of course. Their content is built from the lines carrying the content data between the mentioned feasible breakpoints.

So the feasible breakpoints @02, @03, @05, @07, @08, @010 to @012, and @015 to @019 are never used in the line breaks of the paragraph chosen by *T_EX* (@014 and @022 are used if only six lines are built).

The text entered in the tabbing environment is not repeated in the trace as the line-breaking algorithm is not called and so no trace lines are output.

4 Applications

The description in the previous section makes clear that the task of decoding the tracing information by hand is difficult, or at least time consuming. Moreover, the trace data must be enhanced as example 1 has shown: There are two ways to break the paragraph and the selection involves the knowledge of the current setting of \looseness. So its value has to be put in the log file too. The value is reset to 0 after each paragraph; *T_EX* uses the value that it has at the end of the paragraph ([6, p. 349]). The following code writes the parameter to the log file; here it is applied to the second paragraph of example 1.

Example 2: TeX input

```
\let\orglooseness=\looseness
\def\writeloose{\% output looseness to log
  \immediate\wlog{@ looseness \the\orglooseness}}
\def\setlooseness{\% enhanced version of looseness
  \afterassignment\writeloose\orglooseness}
\let\looseness=\setlooseness
\tracingparagraphs=1 \looseness=-1
This is a nonsense text to serve as a ...
```

TeX output

This is a nonsense text to serve as a constructed example that shows all kind of trace lines. It contains inline mathematics and text in columns. The formula $2 \times 2^2 = 8$ is simple mathematics as well as formula $\sqrt[3]{8} = 2$ or what do you think? Now a declaration or definition for a three columns tabbing environment is made. □

The first line of the excerpt from the log file shows the output of the macros; note that the line might not appear directly in front of the start of the data but it is always first. The trace data is nearly identical to the data shown for example 1. All trace lines are present but their sequence is changed and at the end one more feasible breakpoint is added for a six-line paragraph; in this case the ligature in “definition” is resolved and the break is after the ‘f’.

Example 2 continued: Log file contents

```
1. @ looseness -1
2. @firstpass
3. []\ninerm This is a nonsense text to serve
   as a constructed
4. @ via @00 b=23 p=0 d=1089
5. @@1: line 1.1 t=1089 -> @00
6. @secondpass
...
76. made.
77. @\par via @019 b=0 p=-10000 d=100
78. @\par via @018 b=0 p=-10000 d=100
79. @\par via @017 b=0 p=-10000 d=10100
80. @\par via @016 b=0 p=-10000 d=15100
81. @\par via @015 b=10 p=-10000 d=5400
82. @\par via @014 b=48 p=-10000 d=8364
83. @021: line 6.2- t=70791 -> @015
84. @022: line 6.3- t=63051 -> @014
85. @\par via @020 b=0 p=-10000 d=100
86. @023: line 7.2- t=58261 -> @020
```

The integer parameter `looseness` influences the line-breaking algorithm and makes it select paragraph lines that are not the optimum. Most often this price is worth being paid to improve the page breaking. It would be useful to inform an author about the possibilities to shorten or to lengthen a paragraph. Section 5 discusses this topic in more detail.

In the following I do not discuss lines with overfull boxes, etc. These are reported by TeX during the run. My recommendation is that an author react to these messages. Moreover, only the multiline

paragraphs are handled in the figures. This article contains many single-line paragraphs through the verbatim listings but they are not discussed. The following tasks are addressed:

1. Find hyphenated words.
2. Find the longest sequence of hyphenated lines.
3. Find paragraphs that contain visually incompatible lines.
4. Find sequences of lines starting or ending with the same word.
5. Use statistics to learn about the overall appearance of the text.
6. Perform actions to eliminate detected problems.

List of hyphenated words. In a Q&A session Donald E. Knuth was asked why TeX does not provide a way to generate a list of hyphenated words of a text. He answered that a little filter program can do the work if all relevant tracing is switched on ([12], p. 365 or [13], pp. 620–621).

There are several ways to get the hyphenated words, for example, one could set `\hbadness=-1` and check the output for lines ending in a hyphen. See for example, exercise 28 of [11]. (Note: Such lines often start with the words “loose” and “tight” but that does not refer to C1 and C3, resp. See [6], p. 302.) A problem might be that the part of a word at the beginning in the last line is not output. Another approach is to filter the output of `dviotype` [10]. I decided to use the trace data of the command `\tracingparagraphs`. Of course, that meant writing the “little filter program” for efficient extraction of data. Such a program reads the trace data, chooses the right final feasible breakpoint, goes back through the chain of feasible breakpoints looking at the content data and saves all hyphenated words. The script might output a list like the one in Fig. 1.

It is much easier to check such a file for bad hyphenation than to go through the DVI output and check every end of line. Changes to this list between runs can be analyzed by a `diff` command. And more is possible: As TeX hyphenates all the words in certain passes they can be collected in a list, supplied with corrected hyphenation points if necessary, and saved in a database. With time this database

A ‘=’ marks the hyphenation point.

```
1: con=trol
2: pa=ram-e-ters
3: di-ag=nose
...
160: Stan=ford
161: Pro=gram
```

Figure 1: List of hyphenated words

```
Show line-break statistics based on the log file of a TeX run.

texput.log analyzed on 05/11/16, 14:48:16

Hsize: 225.0pt; Parindent: 20.0pt; Parfillskip: 168.75pt plus 13.49945pt minus 168.75pt

Parameter settings for line breaking:

Pretolerance: 100; Tolerance: 200; Emergencystretch: 0.0pt
Linepenalty: 10; Exhyphenpenalty: 49; Hyphenpenalty: 50
Binoppenalty: 700; Relpenalty: 500
Adjdemerits: 10000; Doublehyphendemerits: 10000; Finalhyphendemerits: 5000

Single-line paragraphs : 570
vloose loose decent tight
Line categories: 0 0 563 7

Multiline Paragraphs : 201
with total lines : 1155
and lines pro par between : 2 and 25
with demerits between : -156 and 545991

Demerits in range >1,000,000 200,000 50,000 20,000 10,000 5,000 0
Positive values: 0 2 17 46 25 26 84
Negative values: 0 0 0 0 0 0 1
```

Figure 2: General information about this article

represents an author's active vocabulary and the list of all hyphenated words can be checked against the database to find wrong hyphenation points that can be given as exceptions using the control word `\hyphenation`. And new entries enlarge the database (for this article by 56 words). The list can also be generated by `\pretolerance=-1` (no first pass), set `\emergencystretch=\hsize` (allow awful lines), apply `\looseness=1000` to every paragraph, and run `dvitype` [10] to find all hyphenated words.

A wrong hyphen in a word must be corrected, of course. Either declare the word at the beginning of the document as a hyphenation exception, or add `\-` to the word at the right place, or put a short word into an `\hbox` to avoid the hyphen. The last two methods are useful if the word occurs only a few times in the text. For this text three hyphenation exceptions are specified: `Eng-lish`, `stretch-abil-ity` and `Mas-sa-chu-setts`.

Note: In order to distinguish between explicit, i.e., author entered, and implicit, i.e., TeX inserted, hyphens I subtract 1 from `\exhyphenpenalty` if it equals `\hyphenpenalty` (see Fig. 2). This changes the calculations of TeX during the tracing compared

```
13% lines with implicit hyphen : 161
0% lines with explicit hyphen : 2
with longest sequence : 3
and hyphenated final lines : 22
```

Figure 3: Global statistics about hyphenated lines

to the normal run, but the impact is usually small, at most 99 demerits for a break at an explicit hyphen with the defaults of plain TeX.

Martin Budaj wrote a script in Perl [1] that finds the hyphenated words in the trace data and outputs a list similar to Fig. 1. A LuaTeX solution is described in [4]. Its author, Patrick Gundlach, developed also the package [3] for LuaLaTeX to show all hyphenation points using tiny vertical marks inside the text similar to the triangles in Fig. 1 of [5].

Counting consecutive hyphenated lines. As all the lines are checked for discretionary breakpoints, overall statistics can be collected to give information on the longest sequence of consecutive lines that are hyphenated. For example, the report for an early draft of this article showed that the longest sequence of hyphenated lines was 5, which is too long according to [2], 3.11: *When four or more lines end with a hyphen or the same word, word spacing should be adjusted to prevent such “stacks.”* The current count for this article is shown in Fig. 3.

The length of the longest sequence of hyphenated lines is valuable and easily output by the script. The author decides if this length is acceptable or not. How can the hyphen stack be reduced? An author has several possibilities:

- A0. change the words of the paragraph;
- A1. increase the penalties and demerits that have to do with hyphenation for this paragraph;

- A2. lower the `\tolerance` and use a positive value for `\emergencystretch` in this paragraph;
- A3. try to make the paragraph a line longer (or shorter) using `\looseness` (maybe supported by a positive value of `\emergencystretch`);
- A4. improve the chance of the first pass by increasing `\pretolerance` for this paragraph;
- A5. put the third or fourth hyphenated word in an `\hbox` if it is a short word.

Action A0 is always an option and it is guaranteed to be successful; the other actions might fail. Actions A1, A2, and A4 should not be made for the whole text; apply the parameter setting only to the “bad” paragraph; see below.

Relationships between lines. Instead of counting and printing numbers of lines of a certain type the relationships between lines can be documented. As an example I use the data of the fitness classes C0–C3.

4% very loose lines	:	53
21% loose lines	:	243
62% decent lines	:	713
13% tight lines	:	146

Figure 4: Distribution of fitness classes

Figure 4 reports on the distribution of the lines into the four classes. The distribution shows that less than two-thirds of all lines in multiline paragraphs are decent. Michael F. Plass and Donald E. Knuth gave in [5] some results for the second volume of *The Art of Computer Programming*, a book with 702 pages, 5526 paragraphs, and 21057 lines. (I refer to this article through the reprint in [13].) The article was published in 1981 one year before `TEX82` was released and the algorithm was changed a little bit for `TEX82`, see [8, § 813]. But the algorithm is close enough that the data can serve as an example. In [13], p. 125, Fig. 19, Donald E. Knuth shows the distribution of lines into the fitness classes. A rough measurement of this data together with the definitions on page 112 of [13] gives the distribution (2, 14, 79, 5)% for (very loose, loose, decent, tight). So the data for the present article is worse. On the other hand the book has an `hsize` of 468 pt compared to the 225 pt of this column; the line-breaking job is easier for the book.

The numbers for transitions from one class to another provides additional insights (see Fig. 5). It gives the volume of visually incompatible lines. Only a few lines are incompatible, about 2.9%. In summary the data looks acceptable to me. Only a few jumps from very loose or loose to tight occur. The majority of transitions is listed on the “diagonal”,

From / To:	vloose	loose	decent	tight
vmode	1	2	169	29
vloose	18	11	12	3
loose	15	57	109	15
decent	16	157	339	75
tight	3	16	84	24

Figure 5: Transitions between fitness classes

so lines of the same class follow most often. One intentionally bad paragraph is the second paragraph in the introduction. There, lines 3 and 4 are very loose, lines 5 and 7 are tight, and line 6 is loose.

To improve a paragraph with excessive transitions between visually incompatible lines the above-mentioned action A2 seems to be the best choice. Usually it reduces the number of very loose lines if the parameters are chosen carefully; see below.

Distribution of demerits. When the topic “distribution” is considered, the idea of showing the data of `TEX`’s rating values, the demerits, comes to mind. Of course, a script can easily document them and I use a set of ranges for positive and negative values to categorize the data points. The scripts calculate the distribution as shown in Fig. 2. Negative values can occur as an author can specify them via `\penalty`. (The paragraphs with negative demerits appear in the references where URLs are broken by macros.)

The problem that I have with the data is the lack of a trigger for action. A twenty-line paragraph with lines all having badness 10 and no hyphenations has the same demerits as a two-line paragraph with lines having badnesses of 10 and 0 and a hyphen after the first line. Which one is better? What can be done to improve the situation? Is there a problem at all?

But the statistic is useful to give a general overview. When the default values of `TEX` are active, paragraphs with demerits in the range 0–5000 can have only one hyphenated word, which is not at the end of the second last line: The penalty for a hyphenated word is $50^2 = 2500$ so there cannot be two if the total demerits are at most 5000 and, of course, the value of `\finalhyphendemerits` was not applied. Similarly the paragraphs in the range 5001–10000 have no stack of two hyphens and no visually incompatible lines. The next range might have just one of such things but only once.

So one can concentrate on the few paragraphs that have very high values of demerits. But let me state again: A high value does not imply a problem. In this article most paragraphs with high demerits appear in examples. The paragraph above the description of the fitness classes on the first page is an exception (see Fig. 11): It has the highest demerits

(see Fig. 2) because of 13 lines, three have a badness above 50, one more than 100, one \binoppenalty and one \hyphenpenalty are charged, and it has one pair of visually incompatible lines. But only the break in the formula might trigger a change.

```

number of words in paragraphs      : 8417
max. words in one line           : 14
one-word lines (multiline par)   : 35

Repeat word >4 chars at start   : 3
        or end of line          : 2
Repeat shorter word at start    : 10
        or end of line          : 7
Longest sequence at start of line: 2
Longest sequence at end of line  : 2

```

Figure 6: Several global counts

More global statistics. Other counts can be calculated. For example, I use an experimental count of lines that start or end with the same word or syllables as the previous line (see Fig. 6). In this text most occurrences of stacks of words with at least five letters appear in the examples. As the longest sequence is two, there is no problem according to [2]. Otherwise an author should tie one of the words to the previous or the following word if the stack appears at the start or the end of the line, resp. It seems best to connect the tie to the shortest possible word or syllable. The number of hyphenated lines might increase or an overfull line is created if the stack appears near the beginning of the paragraph; then the text must be rewritten to avoid it.

Other general statistics can be generated and they may trigger actions by an author although the interpretation is more complicated. Such statistics do not point to a certain situation in the input which might be changed.

```

50% successful in the first pass : 100
43% successful in second pass   : 86
2% successful without first pass: 5
5% needed an emergency pass    : 10

```

Figure 7: Global statistics about passes

Figure 7 shows the distribution of passes for this article. But how can this data be interpreted? The emergency pass is used in several examples and the list of references where a positive value for the \emergencystretch is specified. The main question is: Is the shown distribution between first and second pass OK? In [13], p. 123, Donald E. Knuth writes that in the second volume of *The Art of Computer Programming* (TAOCP) only 5% of all paragraphs needed a second pass and only 2.26% lines ended in a hyphen. So compared to these data the values are

55% successful in the first pass	:	108
44% successful in second pass	:	87
0% successful without first pass	:	0
1% needed an emergency pass	:	2
5% very loose lines	:	49
20% loose lines	:	197
64% decent lines	:	643
11% tight lines	:	110
11% hyphenated lines	:	109
11% lines with implicit hyphen	:	105
0% lines with explicit hyphen	:	4
with longest sequence	:	3
and hyphenated final lines	:	18

Figure 8: Global statistics of another article [15]

bad. On the other hand the columns of this journal are much smaller than the \hsize of the book. It needs some judgment to decide if the values are acceptable or if some parameters should be changed. As I have written another article for this journal [15] its data can be used for a comparison. Figure 8 shows its values for the data shown in Figs. 7, 4, and 3. The values of the present article are worse. But I do not change anything as, for example, some paragraphs have been *designed* to make the first pass fail.

What options does an author have to respond to unwanted global statistics? Of course, when an author has to use a given format with given parameters often only rewriting of the text is possible. The distributions are useful to detect problems in the general setup. If a small percentage is seen for the

```

\newtoks\TRsavedLBparameters
\newif\ifprotectLBparameters
\def\SaveallLBparameters{\% store 11 parameters
\ifprotectLBparameters
\else\protectLBparameterstrue
\edef\saveparameters{\%
\global\TRsavedLBparameters=\expandafter
{\the\emergencystretch:\the\pretolerance:%
\the\tolerance:\the\linepenalty:%
\the\hyphenpenalty:\the\exhyphenpenalty:%
\the\binoppenalty:\the\relpenalty:%
\the\adjdemerits:\the\doublehyphendemerits:%
\the\finalhyphendemerits}}\saveparameters\fi}
% usage:\afterassignment\RestoreLBparameters
\def % \emergencystretch=\the\TRsavedLBparameters
\RestoreLBparameters:#1:#2:#3:#4:#5:#6:#7:#8:#9:{%
\pretolerance=#1 \tolerance=#2 \linepenalty=#3
\hyphenpenalty=#4 \exhyphenpenalty=#5
\binoppenalty=#6 \relpenalty=#7
\adjdemerits=#8 \doublehyphendemerits=#9
\finalhyphendemerits=}
% #1: i - s c o ii i- ... (is for --) or \hskip<x>pt
\def\setEMstr(#1){SaveallLBparameters
\setbox0=\hbox{\#1}\emergencystretch=\wd0 }

```

Figure 9: Support macros for Fig. 10

#Pars	P	Demerits	Breakpoints	Lines	opt	L	Lines-found	B-inf	B-min	B-max	vloos	loose	decnt	tight	hyphen	seq	last
1:	1	100		1	1	1	0	1	0	0	0	0	0	1			
2:	1	100		1	1	1	0	1	0	0	0	0	0	1			
3:	1	100		1	1	1	0	1	0	0	0	0	0	1			
4:	2	15554	(0)/25	8	8	8	0	(0)/8-9	0	69		1	5	2	1/ 1		
5:	1	100		1	1	1	0	1	0	0	0	0	0	1			
6:	2	40159	(2)/15	8	8	8	0	(2)/8	0	79		3	5	3/ 3	3		
7:	2	120270	(2)/13	8	8	8	0	(2)/8	3	178	2	2	1	3	2/ 2	2	
8:	2	44163	(5)/52	21	21	21	0	(5)/21	0	89		4	13	4	3/ 3		
9:	2	63786	(1)/46	13	13	13	0	(1)/13-14	0	86		3	6	4	6/ 6	3	Y
10:	2	18559	(2)/16	9	9	9	0	(2)/9	0	30		4	4	1	3/ 3		Y
11:	2	3565	(0)/9	4	4	4	0	(0)/4	0	14		1	3	1	1/ 1		
12:	1	100		1	1	1	0	1	0	0	0	0	0	1			
13:	2	4290	(1)/4	4	4	4	0	(1)/4	0	29		2	2	2	1/ 1		
14:	2	545991	(14)/31	13	13	13	0	(10)/13	0	146	1	3	8	1	1/ 1		

Figure 11: Information about the first several paragraphs of this article

first pass check that (1) no non-breakable items like large hboxes or verbatim strings (as in this article) are present, (2) `\pretolerance` is not too low, and (3) the `\hsize` is appropriate, i.e., not too small for justified text. In such situations where the hyphenation passes appear in the expected amount but the number of hyphenated lines is high, check additionally that (4) the values of the parameters for hyphenation are set reasonably.

A few examples. The actions that change some line-breaking parameters should apply that change only for a single paragraph. The technically named `try...` macros in Fig. 10 (supported by those in Fig. 9) change parameters and with the command `\defaultlinebreaking` after an empty line or a `\par` the old parameters are reset. In most cases action A0, i.e., change the wording, is the best solution; only if this is not possible the actions A2, A3, or A4 should be tried.

In the first example the action A2, i.e., a lower `\tolerance` with `\emergencystretch`, is applied to the second paragraph of the introduction. The parameter to the macro is a string to specify the length of the `\emergencystretch`; an “`\hskip<dimen>`” is

```
\def\defaultlinebreaking{\% reset parameters
\ifprotectLBparameters\protectLBparametersfalse
\afterassignment\RestoreLBparameters
\emergencystretch=\the\TRsavedLBparameters\fi}
\def\tryonlyfirstpass{\SaveallLBparameters
\pretolerance=125 }
\def\tryonlysecondpass{\SaveallLBparameters
\pretolerance=-1 }
\def\trythirdpassD#1{\setEMstr(#1)\%
\tolerance=125 \ignorespaces}
\def\trylesshyphens{\SaveallLBparameters
\hyphenpenalty=75 \doublehyphendemerits=20000
\exhyphenpenalty=55 \finalhyphendemerits=7500 }
```

Figure 10: Set of useful macros

possible too. The characters “i-sco” cover the range of 5–9 basic units of width in `cmt10`; see [9]. This makes the additional stretchability individual to the paragraph.

Example 3: **TEX** input

```
\trythirdpassD{oo}
The tracing parameters might be classified ...
```

TEX output

The tracing parameters might be classified into different groups: Some look at the settings of the installation, like `\tracingstats`, others are used mainly for developers, like `\tracingmacros`, and some (or all) can be used to get a better understanding how **TEX** operates. For example, the parameter `\tracingparagraphs` gives detailed insights into the inner workings of **TEX**’s line-breaking algorithm. □

The paragraph is one line longer; two lines are loose, five decent, then follows a tight and a decent line. No visually incompatible pair is reported. The documented badness values are 91, 43, 1, 7, 5, 8, 3, 19, and 0 instead of 3, 25, 171, 178, 37, 21, 41, and 28. Six lines have a lower value than before. But the reporting does not include the additional stretchability. The true badness values can be seen using `\hbadness=-1`. Then the line badnesses are 1019, 239, 11, 69, 18, 41, 53, 19, and 0; two visually incompatible pairs are present. The first line is so bad that **TEX** reports an underfull hbox. An author should expect looser lines if a paragraph is lengthened.

```
\def\trylongerparD#1{\setEMstr(#1)\%
\finalhyphendemerits=0 \adjdemerits=5000
\looseness=1 \ignorespaces}
```

Figure 12: Stronger than `\looseness=1`

The macros can be combined and more macros are possible, for example, Fig. 12 increases forces

```
--#Par---(#Lines [#per pass Looseness+# for min demerits])---
#Line  Badness  Penalty Demerits FitClass -? ST #w
=====
--1---(1 [1 L0+1])
 1:      0   -10000    100  decent      2 Tracing paragraphs
--2---(1 [1 L0+1])
 2:      0   -10000    100  decent      2 [] Udo Wermuth
--3---(1 [1 L0+1])
 3:      0   -10000    100  decent      1 Abstract
--4---(8 [(0)/8-9 L0+8])
 4:      0     50    2600  decent  Y  9 The pro-gram T[]X pro-vides more than a dozen con-
 5:     29     0    1521  loose   6 trol words for di-ag-nos-tic and de-bug-ging pur-poses.
 6:      0     0    100  decent   9 Some of them are used of-ten, oth-ers han-dle spe-cial
 7:      9     0    361  decent   9 tasks and are less fre-quently ap-plied. In the lat-ter
 8:      1     0    121  decent   5 case falls the pa-param-e-ter \tracingparagraphs that
 9:     57     0   4489  tight   10 seems to be a hid-den gem. This ar-ti-cle ex-plains what
10:      1     0    121  decent   10 the pa-param-e-ter trig-gers if set and how an au-thor can
11:     69   -10000   6241  tight   10 use the trace data to check and im-prove his text.
--5---(1 [1 L0+1])
12:      0   -10000    100  decent      1 1 Introduction
--6---(8 [(2)/8 L0+8])
13:      0     0    100  decent      8 The T[]X soft-ware, de-scribed in T[]X : The Pro-gram
```

Figure 13: Information about the lines of the first paragraphs of the article

to lengthen a paragraph, i.e., it implements action A3. Note that the paragraph does not change with a simple `\looseness=1`.

Example 3 continued: TeX input

```
\trylongerparD(i)
The tracing parameters might be classified ...
```

TeX output

The tracing parameters might be classified into different groups: Some look at the settings of the installation, like `\tracingstats`, others are used mainly for developers, like `\tracingmacros`, and some (or all) can be used to get a better understanding how TeX operates. For example, the parameter `\tracingparagraphs` gives detailed insights into the inner workings of TeX's line-breaking algorithm. □

The typical pattern of `\looseness=1` appears: The new line contains only the last word or a part of it. But the result looks better than before.

Next, stacks of hyphens are removed.

Example 4: TeX input

```
\trylesshyphens\noindent
The program \TeX, described in \TP\ [...
```

TeX output

The TeX software, described in *TeX: The Program* [8], implements several control sequences to show information about its work. The commands and parameters form a set of powerful tools to help diagnose errors. TeX itself contains nine primitive integer parameters for tracing ([6, p. 273]) and four primitive show commands ([6, p. 279]). The plain

format defines additional macros ([6, p. 364]). □

The original text appears as the first paragraph of section 1. It is one of the paragraphs with the longest sequence of hyphenated lines in this text (see Fig. 11). The best solution is to insert “the book” after “in” in the first line, but here `\trylesshyphens` is also successful. Sometimes this command does not work, for example, if the stack is at the beginning of the paragraph; more penalties and demerits might not change the first line break. The macro of Fig. 12 might help; next it is applied to a statement in [12], p. 358, where `\trylesshyphens` is not successful. I use `\trylongerparD(Ar)` for the second paragraph.

Example 5: TeX output

So instead, I worked only at Stanford, at the Artificial Intelligence Laboratory with the very primitive equipment there. We did have television cameras, and my publisher, Addison-Wesley, was very helpful — they sent me the original press-printed proofs of my book, from which *The Art of Computer Programming* had been made. The process in the 60s ...

So instead, I worked only at Stanford, at the Artificial Intelligence Laboratory with the very primitive equipment there. We did have television cameras, and my publisher, Addison-Wesley, was very helpful — they sent me the original press-printed proofs of my book, from which *The Art of Computer Programming* had been made. The process in the 60s ... □

Details for paragraphs and lines. How to find the word or section in the text which is responsible

for a bad value in the statistics? My solution involves the scripts creating two additional files, one with the data about the paragraphs, the other listing the values of all lines (see Figs. 11 and 13).

Figure 11 shows the following information for each paragraph in one line: a sequential number, the number of passes, the number of breakpoints in each pass, the number of lines used, the optimal number of lines, the active looseness, lines found in each pass, number of lines with infinite badness, the minimal badness, the maximal badness, the number of lines that are very loose, loose, decent, and tight, the number of hyphens and implicit hyphens, the longest sequence of hyphenated lines, and a flag to indicate if the second last line is hyphenated. Entries in parentheses stand for failed passes, slashes separate the data of the passes. Figure 13 lists all the details about the lines separated by dashed lines that repeat some data of the paragraph. The dashed lines show the number of the paragraph, the number of lines created (for all passes), lines found per pass, the active looseness, and the optimal number of lines. For each line the line number, the line badness, the penalty at the end of the line, the line demerits, the fitness class, three flags for a hyphen at the end of the line, stacks at the start or end of the line, an approximation to the number of words and the content of the line is output.

To locate, for example, paragraphs that have the longest sequence of lines that end with a hyphen, check the column “seq” in the list of paragraphs (Fig. 11) and go to the entries for this paragraph in the list of lines (Fig. 13) to see the text. In some cases the list of lines can be consulted directly. For example, in the column “FitClass” the word “tight” is moved to the left and the word “loose” to the right. This helps to find visually incompatible lines.

My REXX scripts output probably too much data. Everything that the trace data shows is printed. At least it serves as educational material.

5 Remarks about \looseness

Let's look a little bit closer at the integer parameter `\looseness` and how it influences the line-breaking algorithm and as a consequence also the trace data output. It would be nice to inform an author about the number of lines his paragraph can have.

Example 2 has shown that the looseness does not force the algorithm to make a second pass. Only when a pass cannot provide the desired number of lines does `TeX` start the next pass because the previous pass counts as failed. Therefore, some statements in [14] are wrong in general.

In this section the following facts are shown.

1. In different passes a paragraph can have different number of lines.
2. The use of looseness might result in the execution of a second or third pass.
3. A possible emergency pass is not executed if a previous pass is successful.
4. A possibility to shorten a paragraph with the same pass is not always reported explicitly in the trace data.
5. Similarly, a possibility to lengthen a paragraph might not be reported.
6. The use of looseness might result in different line breaks even if no additional pass is run.
7. This can also happen with a “neutral” parshape.

Different number of lines in the passes. When `TeX` has successfully finished a pass, it builds from the feasible breakpoints the paragraph with the lowest total demerits. During this process the best number of lines N for the paragraph is also determined. A non-zero looseness forces `TeX`'s algorithm to go again through the feasible breakpoints but this time it picks those that change the number of lines by the given value of `\looseness`. If this is not possible, the pass fails and, if it is not the final pass, the next pass is executed. The last pass outputs the paragraph even in a failed situation. The number of lines is then the best approximation that `TeX` has found to the sum of lines for lowest total demerits and the looseness value. Note that the number N is determined individually for each pass. The value for the second pass might be equal to the value of a successful first pass. But other cases are possible too, as the following example shows.

Example 6: `TeX` input

```
Hi \TeX. Tell me how is the following long
word hyphenated: 'antidisestablishmentarianism'?
Now do it.
```

```
\noindent Hi! \TeX! Tell me: How is the
following long word broken
'pneumonoultramicroscopicsilicovolcanoconiosis'?
I am sure that you are an expert in hyphenation,
right \TeX?
```

```
\smallskip \pretolerance=-1
```

```
Hi \TeX. Tell me how is the following long ...
```

`TeX` output

```
Hi \TeX. Tell me how is the following long word
hyphenated: 'antidisestablishmentarianism'? Now do it.
```

```
Hi! \TeX! Tell me: How is the following long word broken
'pneumonoultramicroscopicsilicovolcanoconiosis'? I am
sure that you are an expert in hyphenation, right \TeX?
```

Hi TeX. Tell me how is the following long word hyphenated: 'antidisestablishmentarianism'? Now do it. Hi! TeX! Tell me: How is the following long word broken 'pneumonoultramicroscopicsilicovolcanoconiosis'? I am sure that you are an expert in hyphenation, right TeX?

The first two paragraphs are typeset in the first pass. When that pass is suppressed, as in the third and fourth paragraphs, the pass which tries hyphenation is the only available pass. As the output shows, instead of three lines once two and once four are built. Larger differences between passes are possible as [14] points out. In the case that the second pass creates N lines and the first pass $N + 2$, a `\looseness=1` which the first pass cannot fulfill but the second can result in a shorter paragraph—and TeX claims success. The shortest paragraph with this property that I was able to construct with text in `cmmr9` and an `\hsize` of 225 pt has $N = 39$.

Therefore, if `\looseness=-1` is applied to the second paragraph of example 6 with three lines the result is a successful (based on TeX's rating) second pass that shortened a four line paragraph to three lines. Even if `\emergencystretch` has a positive value TeX does not run a third pass. The paragraph looks identical to the output of the first pass.

An emergency pass is made if the second pass fails to create the requested number of lines.

Example 7: TeX input

```
\pretolerance=-1
\looseness=1 \noindent Hi! \TeX! Tell me:
How is the following long word broken
'pneumonoultramicroscopicsilicovolcanoconiosis'?
I am sure that you are an expert in hyphenation,
right?

\emergencystretch=6.75pt
\looseness=1 \noindent Hi! \TeX! Tell me: ...
```

TeX output

Hi! TeX! Tell me: How is the following long word broken 'pneumonoultramicroscopicsilicovolcanoconiosis'? I am sure that you are an expert in hyphenation, right? Hi! TeX! Tell me: How is the following long word broken 'pneumonoultramicroscopicsilicovolcanoconiosis'? I am sure that you are an expert in hyphenation, right?

The pass can build only three lines, so TeX executes for the second paragraph an emergency pass as the `\emergencystretch` is positive. The next example specifies a positive value for `\emergencystretch` but no emergency pass is executed.

Example 8: TeX input

```
\tracingparagraphs=1 \emergencystretch=4.5pt
\looseness=1
This is a short paragraph and two words can
```

have a hyphen in it. The rest of the text is made up of short words only. Well, I think the first sentence is wrong. Wait then one more must be wrong. Two are wrong.

TeX output

This is a short paragraph and two words can have a hyphen in it. The rest of the text is made up of short words only. Well, I think the first sentence is wrong. Wait then one more must be wrong. Two are wrong.

As the trace data proves in line 17 the line-breaking algorithm is successful in the first pass. It creates a paragraph of four lines. In order to increase this number TeX performs a second pass.

Example 8 continued: Log file contents

```
1. @firstpass
2. []\ninemr This is a short paragraph and two
   words can have
3. @ via @00 b=0 p=0 d=100
...
16. Wait then one more must be wrong. Two are
   wrong.
17. @\par via @04 b=0 p=-10000 d=100
18. @05: line 4.2- t=2509 -> @04
19. @secondpass
20. []\ninemr This is a short para-graph and two
   words can have
21. @ via @00 b=0 p=0 d=100
...
39. Wait then one more must be wrong. Two are
40. @ via @05 b=12 p=0 d=10484
41. @07: line 4.2 t=60914 -> @05
42. wrong.
43. @\par via @06 b=0 p=-10000 d=100
44. @08: line 4.2- t=2509 -> @06
45. @\par via @07 b=0 p=-10000 d=100
46. @09: line 5.2- t=61014 -> @07
```

No information in the trace data. The information in the trace of example 1 that the paragraph could be typeset with six instead of seven lines was part of the construction of the example. Here are some examples which demonstrate that this is not always reported. The first example uses a negative value for `\looseness`.

Example 9: TeX input

```
\tracingparagraphs=1
This is a short paragraph and two words can
have a hyphen in it. The rest of the text
is made up of short words only. I think the
first sentence is wrong. Wait then the next
one must be wrong too. Two are wrong, or?
```

TeX output

This is a short paragraph and two words can have a hyphen in it. The rest of the text is made up of short words only. I think the first sentence is wrong. Wait then the next one must be wrong too. Two are wrong, or?

The line-breaking algorithm finds seven feasible breakpoints and the reported breaks are for a paragraph of five lines. It is not reported that the paragraph can be set in four lines.

Example 9 continued: Log file contents

```

1. @firstpass
2. []\ninelrm This is a short paragraph and two
   words can have
3. @ via @00 b=0 p=0 d=100
4. @@1: line 1.2 t=100 -> @00
5. a
6. @ via @00 b=19 p=0 d=841
7. @@2: line 1.3 t=841 -> @00
8. hyphen in it. The rest of the text is made
   up of short
9. @ via @01 b=0 p=0 d=100
10. @ via @02 b=4 p=0 d=196
11. @@3: line 2.2 t=200 -> @01
12. words only. I think the first sentence is
   wrong. Wait
13. @ via @03 b=24 p=0 d=1156
14. @@4: line 3.1 t=1356 -> @03
15. then
16. @ via @03 b=89 p=0 d=9801
17. @@5: line 3.3 t=10001 -> @03
18. the next one must be wrong too. Two are
   wrong,
19. @ via @04 b=1 p=0 d=121
20. @@6: line 4.2 t=1477 -> @04
21. or?
22. @\par via @05 b=0 p=-10000 d=100
23. @\par via @06 b=0 p=-10000 d=100
24. @@7: line 5.2- t=1577 -> @06

```

Nevertheless, the setting `\looseness=-1` succeeds in the first pass and a four line paragraph is output.

Example 9 continued: TeX input

```
\tracingparagraphs=1
\looseness=-1
This is a short paragraph and two words can ...
```

TeX output

This is a short paragraph and two words can have a hyphen in it. The rest of the text is made up of short words only. I think the first sentence is wrong. Wait then the next one must be wrong too. Two are wrong, or? □

The log file contains only one additional line, the feasible breakpoint for a shorter paragraph in line 23.

Example 9 continued: Log file contents

```

1. @firstpass
2. []\ninelrm This is a short paragraph and two
   words can have
3. @ via @00 b=0 p=0 d=100
...
21. or?
22. @\par via @05 b=0 p=-10000 d=100
23. @@7: line 4.2- t=10101 -> @05
24. @\par via @06 b=0 p=-10000 d=100
25. @@8: line 5.2- t=1577 -> @06

```

The next example sets the looseness parameter to 1, i.e., the number of lines for the paragraph should be one more than the optimum.

Example 10: TeX input

```
\tracingparagraphs=1
```

Let's look at another example. We saw that $\sqrt[3]{8}=2$ and $2^3=8$. What happens when 2 and 3 are switched? The equal sign is wrong! So write $\sqrt[3]{2}\neq 8$ and $3^2\neq 8$.

```
\looseness=1
```

Let's look at another example. We saw that ...

TeX output

Let's look at another example. We saw that $\sqrt[3]{8}=2$ and $2^3=8$. What happens when 2 and 3 are switched? The equal sign is wrong! So write $\sqrt[3]{8}\neq 3$ and $3^2\neq 8$.

Let's look at another example. We saw that $\sqrt[3]{8}=2$ and $2^3=8$. What happens when 2 and 3 are switched? The equal sign is wrong! So write $\sqrt[3]{8}\neq 3$ and $3^2\neq 8$. □

Again only one additional feasible breakpoint appears in the trace for the longer paragraph (see line 38). In both cases only the first pass is executed.

Example 10 continued: Log file contents

```

1. @firstpass
2. []\ninelrm Let's look at another example. We
   saw that $[][] =
3. @\penalty via @00 b=0 p=500 d=250100
4. @@1: line 1.2 t=250100 -> @00
5. 2$
6. @\math via @00 b=73 p=0 d=6889
7. @@2: line 1.3 t=6889 -> @00
8. and $2[] = 8$. What happens when 2 and 3
   are switched?
9. @ via @01 b=53 p=0 d=3969
10. @ via @02 b=0 p=0 d=100
11. @@3: line 2.2 t=6989 -> @02
12. The equal sign is wrong! So write $[][]
   \ninesy 6\ninelrm = 3$ and $3[] \ninesy
   6\ninelrm =
13. @\penalty via @03 b=23 p=500 d=251089
14. @@4: line 3.1 t=258078 -> @03
15. 8$.
16. @\par via @03 b=0 p=-10000 d=100
17. @\par via @04 b=0 p=-10000 d=100
18. @@5: line 3.2- t=7089 -> @03
19.
20. @firstpass
...
35. @\par via @03 b=0 p=-10000 d=100
36. @@5: line 3.2- t=7089 -> @03
37. @\par via @04 b=0 p=-10000 d=100
38. @@6: line 4.2- t=258178 -> @04

```

If we want more information in the trace data, we have to find a way to have TeX report feasible breakpoints without setting `\looseness`. Unfortunately, this is not possible. A non-zero `\looseness` does two things: a) it changes the number of “easy”

lines to \TeX 's maximum and b) it forces the execution of a slightly more complicated loop to find breakpoints. The code of this loop is shown in § 875 of [8] and it is only executed if the looseness parameter has a non-zero value (§ 873). But as both examples show, the possibility to shorten or lengthen a paragraph seems to be indirectly included in the end-of-par break candidates. The line number of the feasible breakpoints associated with an end-of-par break candidate can simply be increased by one and that value gives a possible alternative.

A digression. The change in item a) can be simulated and it has an interesting side effect: \TeX might change the output of a paragraph with a non-zero $\backslash\text{looseness}$ even if the looseness command cannot be obeyed.

Example 11: \TeX input

```
\tracingparagraphs=1 \pretolerance=-1
A one! Or two! Oh! A one! A two! A three!
It is a lovely day and I've got a feeling!
A new feeling! Yes it's a sunny day! Good
day! Sunshine! Sunshine! Sun! I'm in ---
hey, the text of the song sounds familiar.

\looseness=-1
A one! Or two! Oh! A one! A two! A three! ...
```

\TeX output

A one! Or two! Oh! A one! A two! A three! It is a lovely day and I've got a feeling! A new feeling! Yes it's a sunny day! Good day! Sunshine! Sunshine! Sun! I'm in — hey, the text of the song sounds familiar.

A one! Or two! Oh! A one! A two! A three! It is a lovely day and I've got a feeling! A new feeling! Yes it's a sunny day! Good day! Sunshine! Sunshine! Sun! I'm in — hey, the text of the song sounds familiar. □

The line-breaking algorithm uses an internal counter to mark certain lines as “easy.” \TeX 's algorithm saves space and time by the fact that after a certain point all lines have the same length ([8, § 818]) and this point is given by that counter. As stated in a), the counter is set to its maximum if $\backslash\text{looseness}$ is used. In example 2 we observed that the sequence of breakpoints in the trace output was changed compared to example 1. This effect has to do with the counter for easy lines (see [8, § 819]). As \TeX picks the first break candidate that minimizes the total demerits the sequence is important.

The effect is seen not only when $\backslash\text{looseness}$ is non-zero, as the internal counter for easy lines is also set by $\backslash\text{hangindent}$ and $\backslash\text{parshape}$ (see [8, §§ 848–849]). A “neutral” $\backslash\text{parshape}$ specification—all lines have length $\backslash\text{hsize}$ and there are more lines than the paragraph will have—increases the counter for easy lines high enough to stimulate the same output as a non-zero value for $\backslash\text{looseness}$.

A five-line parshape outputs the paragraph in the style of the second paragraph above.

Example 12: \TeX input

```
\def\fivelineparshape{\parshape 5 0pt \hsize
0pt \hsize 0pt \hsize 0pt \hsize }
\tracingparagraphs=1
A one! Or two! Oh! A one! A two! A three! ...
\fivelineparshape
A one! Or two! Oh! A one! A two! A three! ...
```

\TeX output

A one! Or two! Oh! A one! A two! A three! It is a lovely day and I've got a feeling! A new feeling! Yes it's a sunny day! Good day! Sunshine! Sunshine! Sun! I'm in — hey, the text of the song sounds familiar.

A one! Or two! Oh! A one! A two! A three! It is a lovely day and I've got a feeling! A new feeling! Yes it's a sunny day! Good day! Sunshine! Sunshine! Sun! I'm in — hey, the text of the song sounds familiar. □

Let's look at the trace data. In this example two different sets of line breaks produce exactly the same total demerits, but in the first the line badnesses are 2, 1, 0, and in the second, 0, 2, 1.

Example 12 continued: Log file contents

```
1. @firstpass
2. []\ninemr A one! Or two! Oh! A one! A two!
   A three!
3. @ via @00 b=57 p=0 d=4489
4. @@1: line 1.1 t=4489 -> @00
5. It
6. @ via @00 b=7 p=0 d=289
7. @@2: line 1.2 t=289 -> @00
8. is
9. @ via @00 b=0 p=0 d=100
10. @@3: line 1.2 t=100 -> @00
11. a
12. @ via @00 b=2 p=0 d=144
13. @@4: line 1.2 t=144 -> @00
14. lovely day and I've got a feeling! A new
   feeling!
15. @ via @01 b=1 p=0 d=121
16. @ via @02 b=40 p=0 d=2500
17. @@5: line 2.1 t=2789 -> @02
18. @@6: line 2.2 t=4610 -> @01
19. Yes
20. @ via @02 b=3 p=0 d=169
21. @ via @03 b=2 p=0 d=144
22. @ via @04 b=25 p=0 d=1225
23. @@7: line 2.1 t=1369 -> @04
24. @@8: line 2.2 t=244 -> @03
25. it's
26. @ via @03 b=64 p=0 d=5476
27. @ via @04 b=1 p=0 d=121
28. @@9: line 2.2 t=265 -> @04
29. @@10: line 2.3 t=5576 -> @03
30. a
31. @ via @04 b=64 p=0 d=5476
32. @@11: line 2.3 t=5620 -> @04
33. sunny day! Good day! Sunshine! Sunshine!
34. @ via @05 b=8 p=0 d=324
35. @ via @06 b=8 p=0 d=324
```

```

36. @@12: line 3.2 t=3113 -> @@5
37. Sun!
38. @ via @@7 b=1 p=0 d=121
39. @ via @@8 b=1 p=0 d=121
40. @ via @@9 b=62 p=0 d=5184
41. @ via @@10 b=62 p=0 d=15184
42. @@13: line 3.1 t=5449 -> @@9
43. @@14: line 3.2 t=365 -> @@8
44. I'm
45. @ via @@9 b=0 p=0 d=100
46. @ via @@10 b=0 p=0 d=100
47. @ via @@11 b=5 p=0 d=225
48. @@15: line 3.2 t=365 -> @@9
49. in
50. @ via @@11 b=1 p=0 d=121
51. @@16: line 3.2 t=5741 -> @@11
52. --- hey, the text of the song sounds
   familiar.
53. @\par via @@12 b=2 p=-10000 d=144
54. @\par via @@13 b=0 p=-10000 d=100
55. @\par via @@14 b=0 p=-10000 d=100
56. @\par via @@15 b=0 p=-10000 d=100
57. @\par via @@16 b=0 p=-10000 d=100
58. @@17: line 4.2- t=465 -> @@15
59.
60. @firstpass
61. []\ninem A one! Or two! Oh! A one! A two!
   A three!
62. @ via @@0 b=57 p=0 d=4489
63. @@1: line 1.1 t=4489 -> @@0
64. It
...
74. @ via @@2 b=40 p=0 d=2500
75. @ via @@1 b=1 p=0 d=121
76. @@5: line 2.1 t=2789 -> @@2
77. @@6: line 2.2 t=4610 -> @@1
78. Yes
...
112. @\par via @@16 b=0 p=-10000 d=100
113. @\par via @@15 b=0 p=-10000 d=100
114. @\par via @@13 b=0 p=-10000 d=100
115. @\par via @@14 b=0 p=-10000 d=100
116. @\par via @@12 b=2 p=-10000 d=144
117. @@17: line 4.2- t=465 -> @@14

```

Look at lines 53–58 and 112–117: the best final feasible breakpoints select different previous feasible breakpoints, as the order of the break candidates is not the same. In other places of the trace this happens too but without consequence.

References

- [1] Martin Budaj, *findhyph*, V3.4, 18.10.2015
<http://ctan.org/pkg/findhyph>
- [2] *The Chicago Manual of Style*, 15th edition, Chicago, Illinois: University of Chicago Press, 2003
- [3] Patrick Gundlach, *showhyphens*, V0.5c, 19.02.2016
<http://ctan.org/pkg/showhyphens>
- [4] Patrick Gundlach, *lua-check-hyphen*, V0.4, 02.04.2016
<http://ctan.org/pkg/lua-check-hyphen>

- [5] Donald E. Knuth and Michael F. Plass, “Breaking paragraphs into lines,” *Software—Practice and Experience* **11** (1981), 1119–1184; reprinted with an addendum as Chapter 3 in [13], 67–155
- [6] Donald E. Knuth, *The T_EXbook*, Volume A of *Computers & Typesetting*, Boston, Massachusetts: Addison-Wesley, 1984
- [7] Donald E. Knuth, “A torture test for T_EX,” Stanford Computer Science Report *STAN-CS-84-1027*, Stanford, California: Stanford University, 1984
- [8] Donald E. Knuth, *T_EX: The Program*, Volume B of *Computers & Typesetting*, Boston, Massachusetts: Addison-Wesley, 1986
- [9] Donald E. Knuth, *Computer Modern Typefaces*, Volume E of *Computers & Typesetting*, Boston, Massachusetts: Addison-Wesley, 1986
- [10] Donald E. Knuth, “The DV_Itype processor,” in *T_EXware*, Stanford Computer Science Report *STAN-CS-86-1097*, Stanford, California: Stanford University, 1986 (David R. Fuchs designed the first program, Peter Breitenlohner helped with the latest revisions)
<http://ctan.org/pkg/dvitype>
- [11] Donald E. Knuth, “Exercises for T_EX: The Program”, *TUGboat* **11**:2 (1990), 165–170; answers are given in: *TUGboat* **11**:4 (1990), 499–511; reprinted together as Chapter 10 in [13], 197–223 (exercise 28 is in the reprint exercise 25)
<http://tug.org/TUGboat/tb11-2/tb28knut.pdf>
<http://tug.org/TUGboat/tb11-4/tb30knut-exercises.pdf>
- [12] Donald E. Knuth, “C_STUG, Charles University, Prague, March 1996: Questions and Answers with Prof. Donald E. Knuth,” *TUGboat* **17**:4 (1996), 355–367; reprinted as Chapter 32 in [13], 601–624
<http://tug.org/TUGboat/tb17-4/tb53knuc.pdf>
- [13] Donald E. Knuth, *Digital Typography*, Stanford, California: Center for the Study of Language and Information, CSLI Lecture Notes No. 78, 1999
- [14] Frank Mittelbach, “\looseness on the loose,” *TUGboat* **29**:2 (2008), 334; also published in *Die T_EXnische Komödie* **19**:4 (2007), 41; and in: “Pearls of T_EX programming,” *TUGboat* **26**:3 (2005), 256–263, as “\looseness not so loose” (p. 259)
<http://tug.org/TUGboat/tb29-2/tb92mitt.pdf>
- [15] Udo Wermuth, “Typesetting the ‘Begriffsschrift’ by Gottlob Frege in plain T_EX”, *TUGboat* **36**:3 (2015), 243–256
<http://tug.org/TUGboat/tb36-3/tb114wermuth.pdf>

◇ Udo Wermuth
Babenhäuser Straße 6
63128 Dietzenbach
Germany
u dot wermuth (at) icloud dot com



The Treasure Chest

This is a selection of the new packages posted to CTAN (ctan.org) from April–October 2016, with descriptions based on the announcements and edited for extreme brevity.

Entries are listed alphabetically within CTAN directories. More information about any package can be found at ctan.org/pkg/pkgname. A few entries which the editors subjectively believe to be of especially wide interest or otherwise notable are starred; of course, this is not intended to slight the other contributions.

We hope this column and its companions will help to make CTAN a more accessible resource to the TeX community. (See also ctan.org/topic.) Comments are welcome, as always.

◊ Karl Berry
 tugboat (at) tug dot org

biblio

apalike-german in `biblio/bibtex/contrib`
 apalike.bst with German localization.
biolett-bst in `biblio/bibtex/contrib`
 BiBTeX style for *Biology Letters*.
ietfibs in `biblio/bibtex/utils`
 Generate BiBTeX entries for IETF index files.
pbibtex-base in `biblio/pbibtex`
 Base files for the Japanese pBiBTeX.

fonts

baekmuk in `fonts`
 Baekmuk Korean fonts, in TrueType.
beuron in `fonts`
 Monumental capitals from the Beuron art school.
chivo in `fonts`
 The contemporary grotesque Chivo.
***cormorantgaramond** in `fonts`
 A contemporary Garamond-based family in several weights and styles.
eulerpx in `fonts`
 Modern interface for the Euler math fonts.
fonts-churchslavonic in `fonts`
 Fonts for the Church Slavonic language.
frederika2016 in `fonts`
 Zapf's calligraphic Greek companion to his Virtuosa, in OpenType.
oldstandardt1 in `fonts`
 Type 1 versions of the Old Standard fonts previously common in Russia.
ptex-fonts in `fonts`
 Fonts used with pTeX, originally from `ptex-texmf`.

fonts/rosario

rosario in `fonts`

The contemporary semiserif Rosario.

unfonts in `fonts`

Korean Un-fonts collection, in TrueType.

unfonts-extra in `fonts`

Korean Un-fonts collection extras, in TrueType.

uppunctlm in `fonts`

Keep upright shape for punctuation and numerals.

uptex-fonts in `fonts`

Japanese fonts used with upTeX.

yinit-otf in `fonts/gothic`

Haralambous' Old German initials in OpenType.

zhmetrics-uptex in `fonts`

Chinese font metrics for upTeX.

graphics

axodraw2 in `graphics`

Feynman diagrams in LATEX.

binarytree in `graphics/pgf/contrib`

Draw binary trees with TikZ.

mgltex in `graphics`

Create graphics from MathGL scripts embedded directly in a document.

pgf-spectra in `graphics/pgf/contrib`

Draw spectra of elements in PGF.

pst-cie in `graphics/pstricks/contrib`

Draw (many kinds of) arrows in PSTRicks.

table-fct in `graphics/pstricks/contrib`

Draw variations table of a function and convexity table of its graph.

tikz-page in `graphics`

Help visualize page layout.

info

latex2e-help-texinfo-fr in `info`

French translation of `latex2e-help-texinfo`.

pstricks-examples-7 in `info`

Examples from the 7th edition PSTRicks book by Herbert Voß.

texproposal in `info`

LATEX promotion proposal for Chinese universities.

language

banglatex in `language/bengali`

Enhanced LATEX integration for Bangla.

bxjalipsum in `language/japanese`

Dummy Japanese text, counterpart of `lipsum`.

churchslavonic in `language/churchslavonic`

Typesetting in the Church Slavonic language,

platex-base in `language/japanese`

Format and other support for pLATEX.

ptex-base in `language/japanese`

Format and other support for (e)pTeX.

ptex-tools in `language/japanese`

(u)pLATEX standard tools bundle.

uplatex-base in `language/japanese`

Format and other support for upLATEX.

uptex-base in **language/japanese**

Format and other support for upTeX.

macros/generic**autoaligne** in **macros/generic**

Align terms and members in math expressions.

listofitems in **macros/generic**

Operate on lists with user-specified separator.

markdown in **macros/generic**

Support for Markdown across engines via Lua.

olsak-misc in **macros/generic**

Single-file plain TeX macros by Petr Olsák.

randomlist in **macros/generic**

Operate on random list elements.

macros/latex/contrib**acmart** in **macros/latex/contrib**

Support for ACM publications.

*** addfont** in **macros/latex/contrib**

Easier use of fonts without explicit LATEX support.

artthreads in **macros/latex/contrib**

Support for PDF article threads across drivers.

aucklandthesis in **macros/latex/contrib**

Support for University of Auckland theses.

aurl in **macros/latex/contrib**

Semantic Web hyperlinks, such as for `rdf:type`.

autobreak in **macros/latex/contrib**

Simple line/page breaking in `align` environments.

bangorexam in **macros/latex/contrib**

Support for Bangor University examinations.

bxenclose in **macros/latex/contrib**

Hooks for beginning/end of document body.

bxnewfont in **macros/latex/contrib**

\newfontx command allowing font size changes.

coloring in **macros/latex/contrib**

Implicitly define named colors.

cookingunits in **macros/latex/contrib**

Convert and typeset cooking units.

cqthesis in **macros/latex/contrib**

Thesis template for Chongqing University.

datepicker-pro in **macros/latex/contrib**

Date chooser for Adobe Reader and related.

diffcoeff in **macros/latex/contrib**

Write differential coefficients easily.

graphics-def in **macros/latex/contrib**

LATEX graphics drivers: `pdftex.def`, `luatex.def`,

`xetex.def`, `dvipdfmx.def`, `dvisvgm.def`.

ecgdraw in **macros/latex/contrib**

Draw electrocardiograms.

emf in **macros/latex/contrib**

Support for the EMF (electromotive force) symbol.

fetchbibles in **macros/latex/contrib**

Fetch Bible passages from self-defined collection.

filecontentsdef in **macros/latex/contrib**

Display verbatim TeX and make PDF attachment.

footnotehyper in **macros/latex/contrib**

Make `footnote.sty` compatible with `hyperref` and `(x)color`.

fvextra in **macros/latex/contrib**

Automatic line breaking, improved math mode, and other `fancyvrb` enhancements.

getargs in **macros/latex/contrib**

Flexible list-parsing macro.

glossaries-finnish in **macros/latex/contrib**

Finnish language module for `glossaries`.

grant in **macros/latex/contrib**

Support for US government grant proposals.

hustthesis in **macros/latex/contrib**

Unofficial thesis template for Huazhong University. *International Journal of Student Research in Archaeology*.

ijsra in **macros/latex/contrib**

Support for the *International Journal of Student Research in Archaeology*.

jacow in **macros/latex/contrib**

Support for submissions to the Joint Accelerator Conferences Website.

latexit in **macros/latex/contrib**

Fetch and typeset Git information.

ling-macros in **macros/latex/contrib**

Macros for typesetting formal linguistics.

linop in **macros/latex/contrib**

Linear operators as in quantum theory, etc.

makebase in **macros/latex/contrib**

Typeset a counter in any numeric base (default 16).

milog in **macros/latex/contrib**

Fulfill German minimum wage law requirements.

navydocs in **macros/latex/contrib**

Support for US Navy technical reports.

notespage in **macros/latex/contrib**

Fill documents with notes pages and/or notes areas.

nwejm in **macros/latex/contrib**

Support for the new journal *North-Western European Journal of Mathematics*.

optidef in **macros/latex/contrib**

Support for writing minimization problems.

overlays in **macros/latex/contrib**

Lightweight alternative for incremental slides.

phffullpagefigure in **macros/latex/contrib**

Figure content on a full page, with separate caption.

*** phfnote** in **macros/latex/contrib**

Simple but flexible formatting for short documents.

phfparen in **macros/latex/contrib**

Simpler writing of parenthetic math expressions.

phfqit in **macros/latex/contrib**

Support for quantum information theory.

phfquotetext in **macros/latex/contrib**

Quote verbatim text without whitespace formatting.

phfsnwatermark in **macros/latex/contrib**

Watermarks of version control data from Subversion.

phfthm in **macros/latex/contrib**

Enhanced theorem and proof environments.

phonenumbers in **macros/latex/contrib**

Typeset telephone numbers in LATEX.

quicktype in **macros/latex/contrib**

Abbreviations for quick typesetting.

revquantum in **macros/latex/contrib**

Easier writing of quantum papers for `revtex4-1`.

richtext in macros/latex/contrib

Rich text for fields made by the `eforms` package.

rpg-module in macros/latex/contrib

Old-school Dungeons & Dragons modules.

sanitize-umlaut in macros/latex/contrib

Sanitize umlauts in index entries for `makeindex`.

semantic-markup in macros/latex/contrib

Semantic markup in the spirit of the Text Encoding Initiative, especially for humanities and music.

spalign in macros/latex/contrib

Typeset matrices and arrays with spaces and semicolons as delimiters.

testidx in macros/latex/contrib

Dummy text for testing any indexing code.

tocdata in macros/latex/contrib

Add names or other text per sectioning entry in contents and lists of figures.

typed-checklist in macros/latex/contrib

Checklists with different types of items.

umbclegislation in macros/latex/contrib

UMBC Student Government Association bills.

uspace in macros/latex/contrib

Giving meaning to Unicode space characters.

xcntperchap in macros/latex/contrib

Track multiple levels of sectioning.

xcolor-material in macros/latex/contrib

The colors of the Google Material Color Palette.

macros/latex/contrib/babel-contrib**babel-belarusian in m/1/c/babel-contrib**

Babel support for Belarusian.

macros/latex/contrib/beamer-contrib**beamerswitch in m/1/c/beamer-contrib**

Support for one Beamer document producing slides, a handout, a reference, or other variants.

beamertHEME-cuerna in m/1/c/beamer-contrib/themes

Beamer theme with four-color palette.

macros/latex/contrib/biblatex-contrib**biblatex-abnt in m/1/c/biblatex-contrib**

Support for Brazil's ABNT rules.

biblatex-bookinother in m/1/c/biblatex-contrib

New entry types and fields for books contained in another work.

biblatex-claves in m/1/c/biblatex-contrib

Manage claves of old literature.

biblatex-gb7714-2015 in m/1/c/biblatex-contrib

Support for the Chinese GBT7714-2015.

biblatex-ijsra in m/1/c/biblatex-contrib

Support for the *International Journal of Student Research in Archaeology*.

biblatex-iso690 in m/1/c/biblatex-contrib

Support for ISO 690:2010.

biblatex-lni in m/1/c/biblatex-contrib

Support for the *Lecture Notes in Informatics*.

biblatex-morenames in m/1/c/biblatex-contrib

New name fields for BIBLATEX.

biblatex-nottsclassic in m/1/c/biblatex-contrib

Support for the University of Nottingham style.

biblatex-sbl in m/1/c/biblatex-contrib

Support for the Society of Biblical Literature.

macros/luatex**cstypo in macros/luatex/generic**

Czech typography enforced via LuaTeX hooks.

nodetree in macros/luatex/generic

Visualize structure of node lists.

macros/xetex**font-change-xetex in macros/xetex/plain**

Change text and math fonts in plain XETEX.

langsci in macros/xetex/latex

Support for Language Science Press works.

support**byzantinemusic in support**

Support for writing Byzantine music.

extractpdfmark in support

Extract page mode and destinations as PDFmarks.

gitfile-info in support

Get Git metadata for a specific file.

gregoriotex in support

Engraving Gregorian chant scores.

*** latex2nemeth in support**

Convert LATEX to Braille with Nemeth math.

latex-papersize in support

Compute LATEX settings for font and paper sizes.

pdflatexpicscale in support

Downscale graphics for smaller PDF sizes.

pdfxup in support

Compose n -up documents while removing margins.

*** texosquery in support**

Query basic locale, environment, file information.



Cartoon by John Atkinson (<http://wronghands1.com>).

Die *TeXnische Komödie* 2–3/2016

*Die *TeXnische Komödie** is the journal of DANTE e.V., the German-language *TeX* user group (<http://www.dante.de>). (Non-technical items are omitted.)

Die *TeXnische Komödie* 2/2016

UWE ZIEGENHAGEN, Klausuren erstellen mit der Dokumentenklasse *exam* [Creating examinations with the *exam* document class]; pp. 40–51

The *exam* package provides a versatile and powerful way to typeset examinations for use in schools and universities. This article introduces the reader to the basic functions of this class.

ALEXANDER SENGER, Schmuckfarben für *Xe^{\textit{L}}AT_EX* [Spot Colours für *Xe^{\textit{L}}AT_EX*]; pp. 52–56

The *xespotcolor* package is used to display spot colours in *Xe^{\textit{L}}AT_EX* and *L^{\textit{A}}T_EX+dvipdfmx*. Developed by Apostolos Syropoulos in 2016, it is a reimplemention of the *spotcolor* package, published by Jens Elstner in 2006.

CHRISTINE RÖMER, Konstituentenstrukturen einfach und schön mit *forest* [Constituent structures made simple and beautiful with *forest*]; pp. 57–62

Sašo Živanović, the author of the *forest* package, describes the package as follows: “It is due to the awesome power of the supplementary facilities of PGF/TikZ that Forest is now, I believe, the most flexible tree typesetting package for L^{\textit{A}}T_EX you can get.” In perhaps all cases it is much easier to handle than other packages; even complex structures are not a problem.

ROLF NIEPRASCHK, Kalender mit persönlichen Daten [Calendars with personal information]; pp. 63–66

A nice proposal for a calendar based on TikZ is presented at www.texexample.net/tikz/examples/a-calender-for-doublesided-din-a4. One example shows how personal calendar dates such as birthdays or vacations can be highlighted, unfortunately in a rather theoretical way. To prepare a calendar for a colleague who wished to have such a calendar, I created a document class which keeps the dates in a separate file.

HERBERT VOSS, Geometrische Konstruktionen [Geometric constructions]; pp. 67–69

The *pst-eucl* package allows one to construct geometrical objects on the basis of defined points on the plane. In so-called Voronoi diagrams (mathworld.wolfram.com/VoronoiDiagram.html), named after Georgi Feodosjewitsch Woronoi, only the circumcenter of triangles is needed.

Die *TeXnische Komödie* 3/2016

LUKAS C. BOSSERT, UWE ZIEGENHAGEN, HERBERT VOSS, Integration von Python in *TeX* am Beispiel von Katalogeinträgen [Catalogue entries with Python and *TeX*]; pp. 7–20

Many dissertations in archeology contain a catalogue at the end, which shows the analyzed data in a certain scheme. In this article we implement an efficient catalogue with the help of Python and show a *TeX*-only solution as well.

UWE ZIEGENHAGEN, Parallel *TeX*en mit Python [Parallel *TeX*ing with Python]; pp. 21–23

In this article I briefly show, how—with the help of Python—the typical multiple CPU cores in a modern PC can be used to compile files in parallel to save a significant amount of time.

HERBERT VOSS, Trennmuster und deren Anwendung [Hyphenation patterns and their application]; pp. 24–28

Since for most languages hyphenation rules can hardly be expressed in algorithmic form, one can only make use of database- or probability-based procedures. In general one does not care about the internal mechanisms of the hyphenation algorithm, but there are times when one would like to know why a specific word was hyphenated as it was or what hyphenations are possible at all.

HERBERT VOSS, Im Netz gefunden [Found in the net]; pp. 29–41

In various mailing lists, web forums, newsgroups, et al., one finds plenty of helpful information around the topics of typesetting with *TeX*, *L^{\textit{A}}T_EX*, *ConTeXt*, etc. Following are two recent items.

“How *TeX* reads source code”, by Ulrich Diez on November 22, 2014 in [news://de.comp.text.tex: \[...\]](http://news://de.comp.text.tex: [...]) I want to show how *TeX* reads the source file and creates the tokens.

“Slanted characters with a bar”, by Heiko Oberdiek on June 11, 2016 in <http://tex.stackexchange.com/questions/314238/bar-and-overline>: The slanting makes the correct length of the bar a little more complicated. [An] example measures the width of an upright X and uses this for the length of the bar.

[Received from Herbert Voß.]

2017 T_EX Users Group election

Barbara Beeton
for the Elections Committee

The positions of TUG President and nine members of the Board of Directors will be open as of the 2017 Annual Meeting, which will be held in April–May 2017 in Bachotek, Poland.

The terms of these individuals will expire in 2017: Karl Berry, Kaja Christiansen, Steve Grathwohl, Jim Hefferon, Klaus Höppner, Steve Peter, Geoffrey Poore, Arthur Reutenauer, Michael Sofka.

Continuing directors, with terms ending in 2019: Barbara Beeton, Susan DeMeritt, Michael Doob, Cheryl Ponchin, Norbert Preining, Boris Veysman.

The election to choose the new President and Board members will be held in early Spring of 2017. Nominations for these openings are now invited.

The Bylaws provide that “Any member may be nominated for election to the office of TUG President/ to the Board by submitting a nomination petition in accordance with the TUG Election Procedures. Election ... shall be by ... ballot of the entire membership, carried out in accordance with those same Procedures.”

The name of any member may be placed in nomination for election to one of the open offices by submission of a petition, signed by two other members in good standing, to the TUG office; the petition and all signatures must be received by the deadline published below. A candidate’s membership dues for 2017 must be paid before the nomination deadline. The term of President is two years, and the term of a member of the TUG Board is four years.

A nomination form follows this announcement; forms may also be obtained from the TUG office, or via <http://tug.org/election>.

Along with a nomination form, each candidate must supply a passport-size photograph, a short biography, and a statement of intent to be included with the ballot; the biography and statement of intent together may not exceed 400 words. The deadline for receipt of complete nomination forms and ballot information is **5 p.m. (PST) 1 February 2017** at the TUG office in Portland, Oregon, USA. No exceptions will be made. Forms may be submitted by fax, or scanned and submitted by email to office@tug.org; receipt will be confirmed by email.

Information for obtaining ballot forms from the TUG website will be distributed by email to all members within 21 days after the close of nominations. It will be possible to vote electronically. Members preferring to receive a paper ballot may make arrangements by notifying the TUG office; see address on the form. Marked ballots must be received by the date noted on the ballots.

Ballots will be counted by a disinterested party not affiliated with the TUG organization. The results of the election should be available by mid-April, and will be announced in a future issue of *TUGboat* and through various T_EX-related electronic media.

2017 TUG Election—Nomination Form

Only TUG members whose dues have been paid for 2017 will be eligible to participate in the election. The signatures of two (2) members in good standing at the time they sign the nomination form are required in addition to that of the nominee. **Type or print** names clearly, using the name by which you are known to TUG. Names that cannot be identified from the TUG membership records will not be accepted as valid.

The undersigned TUG members propose the nomination of:

Name of Nominee: _____

Signature: _____

Date: _____

for the position of (check one):

TUG President
 Member of the TUG Board of Directors
 for a term beginning with the 2017 Annual Meeting,
April–May 2017.

1. _____
 (please print)

(signature)

(date)

2. _____
 (please print)

(signature)

(date)

Return this nomination form to the TUG office via postal mail, fax, or scanned and sent by email. Nomination forms and all required supplementary material (photograph, biography and personal statement for inclusion on the ballot) must be received at the TUG office in Portland, Oregon, USA, no later than **5 p.m. (PST) 1 February 2017**.¹ It is the responsibility of the candidate to ensure that this deadline is met. Under no circumstances will late or incomplete applications be accepted.

- nomination form
- photograph
- biography/personal statement

T_EX Users Group
Nominations for 2017 Election
 P. O. Box 2311
 Portland, OR 97208-2311
 U.S.A.

(email: office@tug.org; fax: +1 815 301-3568)

¹ Supplementary material may be sent separately from the form, and supporting signatures need not all appear on the same form.

TeX Consultants

The information here comes from the consultants themselves. We do not include information we know to be false, but we cannot check out any of the information; we are transmitting it to you as it was given to us and do not promise it is correct. Also, this is not an official endorsement of the people listed here. We provide this list to enable you to contact service providers and decide for yourself whether to hire one.

TUG also provides an online list of consultants at tug.org/consultants.html. If you'd like to be listed, please see there.

Aicart Martinez, Mercè

Tarragona 102 4^o 2^a
08015 Barcelona, Spain
+34 932267827
Email: [m.aicart \(at\) ono.com](mailto:m.aicart(at)ono.com)
Web: <http://www.edilatex.com>

We provide, at reasonable low cost, L^AT_EX or T_EX page layout and typesetting services to authors or publishers world-wide. We have been in business since the beginning of 1990. For more information visit our web site.

Dangerous Curve

PO Box 532281
Los Angeles, CA 90053
+1 213-617-8483
Email: [typesetting \(at\) dangerouscurve.org](mailto:typesetting(at)dangerouscurve.org)

We are your macro specialists for T_EX or L^AT_EX fine typography specs beyond those of the average L^AT_EX macro package. If you use X_T_EX, we are your microtypography specialists. We take special care to typeset mathematics well.

Not that picky? We also handle most of your typical T_EX and L^AT_EX typesetting needs.

We have been typesetting in the commercial and academic worlds since 1979.

Our team includes Masters-level computer scientists, journeyman typographers, graphic designers, letterform/font designers, artists, and a co-author of a T_EX book.

de Bari, Onofrio and Dominici, Massimiliano

Email: [info \(at\) typotexnica.it](mailto:info(at)typotexnica.it)
Web: <http://www.typotexnica.it>

Our skills: layout of books, journals, articles; creation of L^AT_EX 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

4113 Planz Road Apt. C
Bakersfield, CA 93309-5935
+1 518-951-8786
Email: [david.latchman \(at\) technical-designs.com](mailto:david.latchman(at)technical-designs.com)
Web: <http://www.technical-designs.com>

L^AT_EX consultant specializing in the typesetting of books, manuscripts, articles, Word document conversions as well as creating the customized packages to meet your needs.

Call or email to discuss your project or visit my website for further details.

Peter, Steve

+1 732 306-6309
Email: [speter \(at\) mac.com](mailto:speter(at)mac.com)

Specializing in foreign language, multilingual, linguistic, and technical typesetting using most flavors of T_EX, I have typeset books for Pragmatic Programmers, Oxford University Press, Routledge, and Kluwer, among others, and have helped numerous authors turn rough manuscripts, some with dozens of languages, into beautiful camera-ready copy. In addition, I've helped publishers write, maintain, and streamline T_EX-based publishing systems. I have an MA in Linguistics from Harvard University and live in the New York metro area.

Sofka, Michael

8 Providence St.
Albany, NY 12203
+1 518 331-3457
Email: [michael.sofka \(at\) gmail.com](mailto:michael.sofka(at)gmail.com)

Skilled, personalized T_EX and L^AT_EX consulting and programming services.

I offer over 25 years of experience in programming, macro writing, and typesetting books, articles, newsletters, and theses in T_EX and L^AT_EX: Automated document conversion; Programming in Perl, C, C++ and other languages; Writing and customizing macro packages in T_EX or L^AT_EX; Generating custom output in PDF, HTML and XML; Data format conversion; Databases.

If you have a specialized T_EX or L^AT_EX 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

46871 Antioch Pl.
Sterling, VA 20164
+1 703 915-2406
Email: [borisv \(at\) lk.net](mailto:borisv(at)lk.net)
Web: <http://www.borisv.lk.net>

T_EX and L^AT_EX consulting, training and seminars. Integration with databases, automated document preparation, custom L^AT_EX packages, conversions and much more. I have about nineteen years of experience in T_EX and three decades of experience in teaching & training. I have authored several packages on CTAN, published papers in T_EX related journals, and conducted several workshops on T_EX and related subjects.

Webley, Jonathan

21 West Kilbride Road
Dalry, North Ayrshire, KA24 5DZ, UK
01294538225
Email: [jonathan.webley \(at\) gmail.com](mailto:jonathan.webley(at)gmail.com)

I specialize in math, physics and IT. However, I'm comfortable with most other science, engineering and technical material and I'm willing to undertake most L^AT_EX work. I'm good with equations and tricky tables. I can also proofread and copy-edit if required. I've done hundreds of papers for journals over the years. Samples of work can be supplied on request.

Calendar

2017

- Jan 13–14 College Book Art Association Annual Meeting, “Conspire, Collaboration, Cooperation and Collections”, Florida State University, Tallahassee, Florida. www.collegebookart.org
- Feb 1 **TUG election:** nominations due. tug.org/election
- Feb 5–8 CODEX 2017, 6th Biennial International Book Fair and Symposium, Richmond, California. www.codexfoundation.org
- Feb 23–25 Typography Day 2017, “Typography and Diversity”. Department of Integrated Design University of Moratuwa, Sri Lanka. www.typoday.in
- Feb 24 *TUGboat 38:1* (regular issue), submission deadline.
- Mar 22–24 DANTE 2017 Frühjahrstagung and 56th meeting, Deutsches Elektronen-Synchrotron (DESY), Zeuthen, Germany. www.dante.de/events.html
- Mar 30–31 Center for Printing History & Culture, “From Craft to Technology and Back Again: print’s progress in the twentieth century”, National Print Museum, Dublin, Ireland. <http://www.cphc.org.uk/events>

TUG 2017 & BachoTeX 2017 Bachotek, Poland.

- Apr 29– May 3 The 38th annual meeting of the T_EX Users Group, jointly with the 25th meeting of GUST and GUST’s 25th birthday. tug.org/tug2017

- May 12 *TUGboat 38:2* (proceedings issue), submission deadline.
- May 21–26 16th Annual Book History Workshop, Texas A & M University, College Station, Texas. cushing.library.tamu.edu/programs/bookhistoryworkshop
- May 25–27 TYPO Berlin 2017, “Wanderlust”, Berlin, Germany. typotalks.com/berlin
- Jun 9–12 SHARP 2017, “Technologies of the Book”. Society for the History of Authorship, Reading & Publishing. Victoria, BC, Canada. www.sharpweb.org/main
- Jul 5–7 The Fifteenth International Conference on New Directions in the Humanities (formerly Books, Publishing, and Libraries), “New Directions of the Humanities in the Knowledge Society”, Imperial College, London, UK. thehumanities.com/2017-conference
- Jul 30 – Aug 3 SIGGRAPH 2017, “At the ♥ of Computer Graphics & Interactive Techniques”, Los Angeles, California. s2017.siggraph.org
- Aug 8–11 Digital Humanities 2017, Alliance of Digital Humanities Organizations, “Access”, McGill University, Montréal, Canada. adho.org/conference
- Sep 1 *TUGboat 38:3* (regular issue), submission deadline.
- Sep 11–17 11th International ConTeXt Meeting, Butzbach-Maibach, Germany. meeting.contextgarden.net/2017

2018

- Mar 2 *TUGboat 39:1* (regular issue), submission deadline.

Status as of 15 October 2016

For additional information on TUG-sponsored events listed here, contact the TUG office (+1 503 223-9994, fax: +1 815 301-3568. e-mail: office@tug.org). For events sponsored by other organizations, please use the contact address provided.

User group meeting announcements are posted at lists.tug.org/tex-meetings. Interested users can subscribe and/or post to the list, and are encouraged to do so.

Other calendars of typographic interest are linked from tug.org/calendar.html.

Introductory

- 275 *Gareth Aye / Introducing LaTeX Base*
• web-based \LaTeX editor supporting offline editing and real-time preview
- 256 *Barbara Beeton / Editorial comments*
• typography and *TUGboat* news
- 305 *Hans Hagen / Colorful emojis via Unicode and OpenType*
• Unicode now includes many emojis, and OpenType allows for coloring them
- 255 *Jim Hefferon / President's note*
• TUG news and initiatives seeking help
- 259 *David Walden / Interview with Federico Garcia-De Castro*

Intermediate

- 374 *Karl Berry / The treasure chest*
• new CTAN packages, April–October 2016
- 264 *Peter Flynn / Typographers' Inn*
• dashing it off; \XeLaTeX ; logos
- 267 *Hans Hagen / LuaTeX 1.0.0*
• release of the first stable version of LuaTeX
- 306 *Taco Hoekwater and Hans Hagen / Cowfont (koeiletters) update*
• OpenType font with cows, sheep, math, logos, and other features
- 284 *Werner Lemburg / A survey of the history of musical notation*
• music notation from the earliest known to the present, across cultures, with many illustrations
- 277 *Martin Ruckert / Computer Modern Roman fonts for ebooks*
• careful comparison between METAFONT and other font formats for phones, laptops, etc.

Intermediate Plus

- 317 *Bogusław Jackowski, Piotr Strzelczyk, Piotr Pianowski / GUST e-foundry font projects*
• past, present, future of Latin Modern, \TeX Gyre , and more
- 337 *Nicola Talbot / Localisation of \TeX documents: `tracklang`*
• distributing creation of translations for packages; simplifying use in documents
- 357 *David Walden / Messing with endnotes*
• small macro hacks for convenient endnote references
- 281 *Peter Willadt / When image size matters*
• semi-automatic downscaling images to save space and time
- 352 *Peter Wilson / Glisterings: Index headers; Numerations; Real number comparison*
• fancy headers with marks; automatic numbering; comparing real numbers

Advanced

- 269 *Hans Hagen / LuaTeX 0.82 OpenType math enhancements*
• increased flexibility and extensions to OpenType math handling
- 311 *Linus Romer / Corrections for slanted stems in METAFONT and METAPOST*
• formulae and macros to correct stem widths and angles when slanting
- 358 *Udo Wermuth / Tracing paragraphs*
• help from `\tracingparagraphs` for more pleasing documents

Contents of other \TeX journals

- 377 *Die \TeX nische Komödie 2–3/2016*

Reports and notices

- 376 *John Atkinson / An asterisk's lament*
- 378 *TUG Election committee / TUG 2017 election*
- 254 *Institutional members*
- 379 *\TeX consulting and production services*
- 380 *Calendar*