

TUGBOAT

Volume 38, Number 3 / 2017

General Delivery	291	From the president / Boris Veytsman
	291	Editorial comments / Barbara Beeton Contents of <i>TUGboat</i> issues online; Birthday celebration for Donald Knuth; Public appearances by Don Knuth (online); The Doves Type: reprise; The Go fonts go Greek; Calcula, an experimental display typeface; Extra Bold: A forgery foiled
	293	Collecting memories of the beginnings of desktop publishing / David Walden
	294	Interview: Michael Sharpe / David Walden
	299	Advertising TeX / Hans Hagen
Tutorials	301	The DuckBoat—News from TeX.SE: Asking effective questions / Carla Maggi
Typography	306	Review and summaries: <i>The History of Typographic Writing—The 20th century</i> Volume 2 (ch. 6–8+), from 1950 to 2000 / Charles Bigelow
Fonts	312	Serifed Greek type: Is it “Greek”? / Antonis Tsolomitis
ConTeXt	315	ConTeXt for beginners / Willi Egger
Graphics	324	Art Concret, Basic Design and meta-design / Marcel Herbst 329 The current state of the PSTricks project, part II / Herbert Voß
LATeX	338	Glisternings: Reading lines; paragraph endings; in conclusion / Peter Wilson 342 DocVar: Manage and use document variables / Zunbeltz Izaola and Paulo Ney de Souza 345 Set my (pdf)pages free / David Walden 345 Automatic generation of herbarium labels from spreadsheet data using LATEX / R. Sean Thackurdeen and Boris Veytsman 350 Typesetting actuarial symbols easily and consistently with <code>actuarialsymbol</code> and <code>actuarialangle</code> / David Beauchemin and Vincent Goulet
Software & Tools	353	Converting TeX from WEB to cweb / Martin Ruckert 359 dvisvgm: Generating scalable vector graphics from DVI and EPS files / Martin Gieseking 369 Tricky fences / Hans Hagen
Macros	373	Testing indexes: <code>testidx.sty</code> / Nicola Talbot 400 A note on <code>\linepenalty</code> / Udo Wermuth
Hints & Tricks	415	The treasure chest / Karl Berry 416 Another seasonal puzzle: XII take II / David Carlisle
Cartoons	416	Typeface; Elefonts / John Atkinson
Book Reviews	417	Book reviews: <i>Shady Characters</i> and <i>The Book</i> by Keith Houston / Peter Wilson
Abstracts	420	Die <i>TEXnische Komödie</i> : Contents of issue 3/2017 420 MAPS: Contents of issue 44 (2013)
TUG Business	290	TUGboat editorial information 290 TUG institutional members
Advertisements	421	TeX consulting and production services
News	423	Practical TeX 2018 announcement 424 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.

[printing date: October 2017]

Printed in U.S.A.

Board of Directors

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

Boris Veytsman, *President*^{*}

Arthur Reutenaer*, *Vice President*

Karl Berry*, *Treasurer*

Susan DeMeritt*, *Secretary*

Barbara Beeton

Johannes Braams

Kaja Christiansen

Taco Hoekwater

Klaus Höppner

Frank Mittelbach

Ross Moore

Cheryl Ponchin

Norbert Preining

Will Robertson

Herbert Voß

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

Technical support for
T_EX users:
support@tug.org

Fax
+1 815 301-3568

Contact the
Board of Directors:
board@tug.org

Web

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

Copyright © 2017 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.

The font the *OED* uses has become as recognizable as an old friend. As have the myriad punctuations, symbols, and abbreviations that cover its pages, and which are varied enough to be known in full only to typesetters and longtime readers of this book.

Ammon Shea
Reading the OED
One Man, One Year, 21,370 Pages
(2008)

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP
EDITOR BARBARA BEETON

VOLUME 38, NUMBER 3, 2017
PORTLAND, OREGON, U.S.A.

TUGboat editorial information

This regular issue (Vol. 38, No. 3) is the last issue of the 2017 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.

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

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

TUG Institutional Members

TUG institutional members receive a discount on multiple memberships, site-wide electronic access, and other benefits:

tug.org/instmem.html

Thanks to all for their support!

American Mathematical Society,
Providence, Rhode Island

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

Center for Computing Sciences,
Bowie, Maryland

CSTUG, *Praha, Czech Republic*

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

Maluhy & Co., *São Paulo, Brazil*

Marquette University,
Milwaukee, Wisconsin

Masaryk University,
Faculty of Informatics,
Brno, Czech Republic

MOSEK ApS,
Copenhagen, Denmark

Nagwa Limited, *Windsor, UK*

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

Overleaf, *London, UK*

ShareLaTeX, *United Kingdom*

Springer-Verlag Heidelberg,
Heidelberg, Germany

StackExchange,
New York City, New York

Stockholm University,
Department of Mathematics,
Stockholm, Sweden

TeXFolio, *Trivandrum, India*

TNQ, *Chennai, India*

University College, Cork,
Computer Centre,
Cork, Ireland

Université Laval,
Ste-Foy, Québec, Canada

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

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

From the president

Boris Veytsman

By the time you receive this issue of *TUGboat*, two important conferences will be over: the 11th ConTeXt meeting in Butzbach-Maibach and GuIT 2017 near Venice. We hope to publish reports about them on these pages in future issues. Still, there will be time to register for 2018 meetings: PracTeX 2018 in Troy (NY, USA) in June and TUG 2018 in Rio de Janeiro (Brazil) in July. Both conferences are going to be very interesting. Kris Holmes has agreed to visit PracTeX, and lead a calligraphy workshop there. As to Rio—TUG will be a satellite conference of the International Congress of Mathematicians 2018, so expect many distinguished guests there.

The time since the last *TUGboat* has been busy. There were interesting discussions on the EduTeX list about the ways we can help teachers and other educators to use and teach TeX. I would like to repeat the call for participants: if you would like to help with TeX in schools and universities, please join the mailing list at lists.tug.org/edutex.

One initiative worth your attention is the improvement of the L^AT_EX wikibook (en.wikibooks.org/wiki/LaTeX). This is a first line resource for many TeX novices. Thus it is very beneficial to make it up to date, accurate and well written. If you can spare some time for checking and editing, please do.

The long time health of any organization depends on its membership. We have had several drives in recent years aimed at new individual members. Thus today I would like to talk about another part of our membership: institutional members. I am glad to report that the Association for Computing Machinery, the largest publisher of computer-related literature in the world, joined the ranks of our institutional members. It is my conviction that any organization seriously using TeX in its work should consider joining TUG, not primarily for the membership benefits—albeit we do provide several—but because it is the right thing to do. For many of these organizations: publishers, universities, research institutions, the continuing existence of TeX and its support is important. Thus giving back to the community makes perfect sense. I would like to ask our members to think whether their organization can be persuaded to join TUG.

This issue of *TUGboat* will reach you in the holiday season. So let me wish you a great holiday and happy & fruitful New Year! See you in 2018!

◊ Boris Veytsman
president (at) tug dot org

Editorial comments

Barbara Beeton

Contents of *TUGboat* issues online

In response to an inquiry from a reader of an old *TUGboat* issue online, the maintainers of the archive have determined to reformat the old full-issue files to improve access.

Until now, each issue has been posted with individual articles and the covers as separate files, with a “complete” file containing all the internal content. But the latter file lacks a table of contents, so a potential reader who has downloaded it to be read off-line would have to proceed page by page to find out what is there—inelegant, to say the least.

From this issue forward, and as time permits reprocessing earlier issues, the full-issue file will be arranged in the following manner:

- the TOC (cover 4);
- cover 2 (masthead and general information);
- the issue content;
- cover 3; from **26:2** (2004) onward, a contents list ordered by difficulty.

The front cover will be omitted; this is often quite large, and the space saved seems more important than appreciation of the design. Cover 1 can always be retrieved separately if desired.

Birthday celebration for Donald Knuth

Don’s 80th birthday will occur on 10 January 2018. In honor of this occasion, two celebratory events have been arranged by his friends; both will occur in the northern Swedish town of Piteå.

The first event will be a scientific symposium, “Knuth80: Algorithms, Combinatorics, and Information”, which will take place from Monday through Wednesday morning, January 8–10. Talks at the symposium will include contributions from scientists in “areas where Don’s influence has been important”.

Wednesday afternoon, January 10, will hold the world premiere of *Fantasia Apocalypтика*, a multi-media work for pipe organ and video written by Don. Canadian organist Jan Overduin will perform the work on the magnificent new pipe organ in Studio Acusticum in Piteå. More details concerning the symposium and concert can be found at knuth80.elfbrink.se and on Don’s home page at Stanford.¹ A Facebook “public group”, Knuth80, features photos and facilitates communication between people who are interested in joining the party.

All are welcome—participation is free of charge, but registration is mandatory.

¹ www-cs-faculty.stanford.edu/~knuth/news.html

Public appearances by Don Knuth (online)

Don has taken part in two notable (semi-)public events this year—the celebration of 50 years of Turing awards by the Association for Computing Machinery (Don was the 1974 laureate) and a meeting examining the origins of desktop publishing (DTP) at the Computer History Museum.

The home page for the Turing celebration is at www.acm.org/turing-award-50. Don's talk, on “computer science as a body of accumulated knowledge”, is linked from there, or can be viewed separately.² The brief talk is followed by a Q&A session in which one TeX-related question surfaced: when TeX was first published, Don made a bold wager regarding the number of bugs that would be found in the system, with a reward that would double every year; “How on earth did [he] ever manage to do it?” (This begins at about 14:20 into the recording.)

On May 22–23, a meeting at the Computer History Museum in Mountain View, California, brought together more than 15 participants who had been pioneers in the creation of the DTP industry. The first day of the meeting focused on the development of the underlying technology, and the second, on the history of the companies involved. Two participants of importance to TeX (which, not being commercial, was an anomaly) were Don Knuth and Chuck Bigelow. The proceedings were recorded by video, and are being transcribed. A more detailed report will appear in the next issue. The museum normally posts videos on its youtube channel once transcription is complete.

The Doves Type: reprise

Earlier reports of the recovery of the Doves Press type were covered in 36:1, page 5. Now an audio report has been broadcast as Podcast Episode 168 from *The Futility Closet*.

Listen at www.futilitycloset.com/2017/09/04/podcast-episode-168. It's a good story.

The Go fonts go Greek

In the last regular issue (38:1, pages 5–6), the Go fonts, by Bigelow & Holmes, were reviewed briefly. These fonts, created for the Go project and released under an open source license, have now appeared as the basic font for the web pages—in Greek—of Antonis Tsolomitis at the University of the Aegean, a mathematician and long-time L^AT_EX user. This is, says Chuck Bigelow, “just the sort of thing we were hoping for. Wide unrestricted use, in this case by an

intelligent and discerning mathematician who likes typography. :-)" Antonis' use of the fonts can be seen at myria.math.aegean.gr/~atsol/newpage/.

And while we're mentioning Bigelow & Holmes, their blog, at bigelowandholmes.typepad.com, has many interesting articles, including:

— “Digital Type Archaeology International: *Scientific American* 1983” returns to that article, mentioning what has happened in the intervening 32 years, and shows the opening page of the article as it appeared in the English, French, German, Spanish, Italian, Russian, Japanese and Chinese editions of the magazine.

— “More Zero versus Oh and ellipses versus superellipses” reviews what has happened since “Oh, oh, zero!” appeared in *TUGboat* (34:2, pages 168–181), as well as earlier efforts to distinguish these often confusing glyphs.

Calcula, an experimental display typeface

The Calcula typeface began as an assignment in a typeface design class at the Maryland Institute College of Art. Its creator, Shiva Nailaperumal, was interested in ancient Arabic calligraphic traditions, in particular the geometric Kufic style. In this style, the letters and the ground are in a strict positive/negative relationship, and the geometric character of the style lends itself to the creation of intertwined monograms.

The considerations that went into development of the typeface are explained clearly, with illustrations of how it is applied to words using the Latin alphabet, and to repeating patterns. I find that the effort needed to read the results is less than I would have expected, although this is clearly not intended for serious reading.

See www.typotheque.com/articles/calcula.

Extra Bold: A forgery foiled

The *New Yorker* issue of 31 July 2017 presents the response from the Dutch typeface designer, Lucas de Groot, to an allegation that the font, Calibri, which he had designed almost fifteen years earlier, had been used to forge a document that would clear the Pakistani Prime Minister of a charge of corruption.

The document supposedly had been printed in 2006, whereas the font was not generally released by Microsoft as part of its Office suite in 2007.

This was not the first time Calibri was involved in such allegations; others are listed in the article, at www.newyorker.com/magazine/2017/07/31/calibris-scandalous-history.

² facebook.com/AssociationForComputingMachinery/videos/10154936961388152/

Collecting memories of the beginnings of desktop publishing

David Walden

On May 22–23, 2017, a meeting was held at the Computer History Museum (CHM)¹ of a select set of pioneers from the early days of desktop publishing (DTP). Among the small group of participants were founders and key technologists from Frame Technology (creator of the FrameMaker DTP system), Aldus (creator of PageMaker), Ventura Software (creator of Ventura Publisher), Adobe (creator of the PostScript page description language and the software/firmware to drive a printer from PostScript images in a computer), Xerox PARC (where the first what-you-see-is-what-you-get, WYSIWYG, word processor interface was demonstrated and where early on they turned Xerox copier technology into laser printer technology), and Apple Computer (which promoted PageMaker running on the Mac with its early graphical user interface and a relatively low cost laser printer to become a highly popular early DTP system). A few other key pioneers also participated whose innovations preceded the liftoff of DTP or who moved among the DTP activities making important connections. Finally, a handful of professional and amateur computing historians were invited to be present. The meeting was organized and chaired by Burt Grad (co-founder of the CHM Software Industry Special Interest Group, SI SIG²) and David Brock (director of the CHM's Center for Software History³).

This meeting was the thirteenth pioneers meeting that the SI SIG has held since its founding 25 years ago. At these meetings, pioneers from various parts of the software industry share memories in meeting sessions that are videotaped and then transcribed into text for use by future historians. Individual oral histories are taken from the meeting participants and other significant software industry pioneers, and there has been ongoing collection of original documents. Prior pioneers meetings have resulted in six special issues of the *IEEE Annals of the History of Computing*, and hopefully another special issue will result from the DTP meeting.

While I have been studying the history of desktop text formatting systems for the past several years, I learned many new things as an observer at the DTP meeting. Below are a few examples.

I had not before heard of Rocappi (Research on Computer Applications in the Printing and Publish-

ing Industries) Incorporated. This company, founded by John Seybold in 1963, provided software and consulting for computer-based newspaper and magazine production systems, for instance to Atex. (Atex was an early and for a while a highly popular provider of computer-based newspaper production systems—more than just the page make-up function.) Rocappi's work was parallel to and independent of the RUNOFF-like stream of text processors developed in universities and industry research laboratories.

There were many other inter-company paths of connection.

The founder of Aldus (PageMaker) first got into computer-based publishing when the *Minneapolis Star Tribune* where he worked acquired an Atex system. He worked closely with Atex to specify what the computer-based newspaper system should do. Later he joined the Atex staff and, when Atex was sold to Kodak, started Aldus to build a DTP system.

People at these various companies tended to read the *Seybold Report on Publishing Systems* (begun in 1982) and, a little later, the *Seybold Report on Desktop Publishing* and to attend the Seybold Seminars (founded in 1981). Jonathan Seybold, who had joined his father at Rocappi a couple of years after its founding, was co-founder of the *Reports* with his father and the driving force behind the seminars.

The Seybold reports and meetings were one way the various DTP companies kept track of the market and of what each other were doing; and in collecting material for the reports and meetings Jonathan Seybold got wind of developments within various companies and in at least one important case put the key people in touch with each other. He told the appropriate Mac marketing manager at Apple, the founder of Aldus, and the PostScript people at Adobe that they needed to talk to each other. Out of this came an informal joint marketing campaign and Apple's push of the Mac as a DTP system.

Two pioneers at the DTP meeting are well known to the world of TeX: Don Knuth and Chuck Bigelow. While some meeting participants don't think of TeX as being a DTP system (not WYSIWYG, not commercial, not used on a massive scale), Don's description of his "business plan" for TeX (public domain, not for profit, more or less unchanging, widely portable) was a useful contrast to the goals and plans of the commercial systems. Chuck's story was interesting in how widely he was connected throughout the commercial DTP world as well as with TeX.

◊ David Walden
walden-family.com/texland

¹ computerhistory.org

² computerhistory.org/groups/sisig,
sites.google.com/site/softwareindustrySIG

³ computerhistory.org/softwarehistory

Interview: Michael Sharpe

David Walden



Michael Sharpe has been using \TeX since the mid-1980s. In more recent years he has been active in the \TeX fonts world.

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

Michael Sharpe, interviewee: I was born in Sydney, Australia in 1941. After 1945, my father joined the Commonwealth Public Service, which corresponds in the US to the federal civil service, and moved frequently in order to advance in the system. I had a disjointed schooling in various suburbs of Sydney, Melbourne and Hobart (Tasmania), completing high school and university in Tasmania. I began as a student in Electrical Engineering but found Physics and Mathematics more appealing, thanks in no small part to some inspiring faculty in those areas, and eventually graduated with a degree in Mathematics, after which I completed a Ph.D. in Mathematics at Yale specializing in Analysis and Probability. The interests I shared with some faculty members at the newly formed Mathematics Department at UCSD (University of California at San Diego) led to a position there which continued, except for a year at the University of Paris VI, until my retirement in 2004.

I fantasize sometimes about how different my life might have been had Electrical Engineering at UTAS encompassed what we now know as Computer Science at the time I was a beginning undergraduate in 1959. I also fantasize about what might have been had I continued in physics, though it seemed I had limited capabilities at lab work. (When I asked my friends working in experimental physics about their backgrounds, an unexpectedly high proportion responded that they were the sons of farmers, who understood and could repair all farm machinery.) My youth was misspent on sport, not on understanding

complex machinery, though I did spend a couple of years working as an assistant to a projectionist in the local movie theater during my high school years.

DW: When you say “misspent on sport”, what are you thinking of?

MS: Because we moved regularly, I was motivated to focus on making new friends as quickly as possible, and sport was a good way to do it in that environment. I played cricket, Australian Rules football and tennis. It was fortunate for my later career that I was not really good at any of them.

DW: Were you already doing electronics things as a hobby and enjoying high school math and science before university?

MS: I was not into electronics as a hobby, finding the analog radio of those days not very interesting. I did do well in sciences and math in high school. If there had been computers available in those days, it may have been a different story.

DW: What took you away from Australia and to Yale for your Ph.D. work?

MS: Just previous to my generation of college graduates in Australia, most students wanting to pursue an advanced degree in sciences and engineering went to Great Britain if they could manage it. In the early 1960s the US graduate programs in those areas expanded greatly due in no small part to the post-Sputnik flood of funding. From my limited knowledge of advanced mathematics as an undergraduate, Yale seemed to have a number of major figures in areas of mathematics I enjoyed and thought important. I planned to return to Australia after my Ph.D. and a post-doc, but I married an American while I was at graduate school. My wife did not want to live so far from her family, so I decided to stay in the US (semi-)permanently.

DW: How did you first become involved with \TeX ?

MS: From the mid-1970s, I was working on a manuscript on Markov processes that I hoped could be turned into a book. I was using TROFF and spending much of my research funding to process the source. In the early 1980s, I heard of \TeX and decided to convert to that system, largely because of cost. I made much use of UNIX sed scripts to transform the original source to \TeX . The book was eventually published under the title *General Theory of Markov Processes*, Academic Press (1988).

DW: Are you saying that \TeX was less expensive in terms of computer time? That's a bit of history I have not heard of before.

MS: It was more a matter that TROFF had to be run on the school's mainframe, for which there was a cost associated with each run. Even at the cheapest overnight rate, my research grant took a serious hit. I learned about Textures in the mid-1980s, and could purchase a license for about \$200 due to our site license. The idea of almost instant feedback was very appealing. (In fact, it took close to 40 minutes per page to process using my early generation Mac.)

DW: Do I gather correctly that you were using \TeX and not \LaTeX at that early date? (By the way, it appears to me that your book is available on-line for the cost of registering on a website¹.)

MS: I used plain \TeX with the AMSTEX additions for almost my entire academic career, as my experiments with early \LaTeX made it seem very slow and harder to modify than plain \TeX , which I understood well. Mine was one of the first books accepted in \TeX by Academic Press. They were happy to not have to reprocess it, but asked me to make the chapter headings and such conform to their standards using a Times font and a layout they prescribed. That was my first font job. I understand that they distributed my macros to their other authors whose manuscripts were still in progress.

DW: During your years of teaching at UCSD, what sorts of courses did you teach, and were they mostly to students of math or also students from other departments? Also, given your considerable knowledge of \TeX , etc., did you end up being a resource on \TeX for your department?

MS: I taught courses of many descriptions. Much calculus and advanced calculus of course, as the bulk of our workload was engineering and science majors. I also taught upper division courses in real and complex analysis, probability, mathematical statistics and the mathematics of financial models, mostly taken by majors in math, engineering and economics. At the graduate level, I taught the basic foundation courses in real and complex analysis, probability, as well as advanced topics designed to attract students to one's research area. Many of the students who took my probability courses were electrical engineering graduate students working in communication theory, and as a result, I ended up serving on the dissertation committees of many of them. (That area attracted a steady flow of very capable students thanks to the presence of some first rate researchers and the generous support of QUALCOMM. It was the first time I ever posed a question to a student that elicited the response: "I'm sorry, but that information is covered by a non-disclosure agreement.")

I did end up as a \TeX resource for my department, trying to bring the staff and graduate students up to speed with \TeX . In the years of the severe budgets of the early 1990s, the department had inadequate computer support and lacked skilled staff to do serious document preparation. It took some time for new staff to learn to use \TeX productively, and they found that processing graphics in \TeX documents was too time consuming, so I ended up trying to help them use PSTRicks. This was not really a success as PSTRicks required regular use to maintain proficiency. I also set up a number of automated processes to handle departmental information efficiently, much of it using \TeX for output processing.

My passage to \LaTeX was fairly sudden. The department was contacted by the Dean of Natural Sciences (this was about 2002) saying that one of his major donors was trying to write a math textbook for his grand-children, and was having serious problems getting his chosen fonts to work on his Mac using \LaTeX , and could we do something to help. As no one else stepped forward, I was volunteered. It turned out he was trying to follow one of the recipes in the Alan Hoenig book, *\TeX Unbound*, wanting to set up Adobe Garamond with math from MathTime using a fontinst script. (He was a very smart man in his late 80s who had a Ph.D. in Chemistry, and had set up a very successful corporation that developed rocket parts and fuels. He said he had always loved mathematics but felt it was never properly explained to pre-college students. I visited him in his retirement home, and he declared that I might be the first mathematician to make house calls since Bertrand Russell called on Lady Ottoline.) So, that was my first real font effort, getting things set up for proper fontinst run, and moving the output to the non-standard places expected by Textures. That meant I had to understand \LaTeX , and I found it much more compelling than I had earlier.

After I retired in 2004, I used \LaTeX for the first time to write a math paper, having the luxury of time to learn its fine points. I also became very interested in PSTRicks for a period and wrote some packages for it, some of which worked its way into the basic PSTRicks packages. That gave me my first real experience working with Bezier curves, which are the foundation for work with outline fonts. It was also a good learning experience with complex \TeX packages, as there is some fiendishly clever code in PSTRicks, mixed with non-trivial PostScript code. Its long-time maintainer, Herbert Voß, is very knowledgeable and helpful in all (\LaTeX) \TeX matters.

After I made some baby font packages, I saw an opportunity with the TXfonts and PXfonts families,

which had been developed but left in an unfinished (or, at least, unpolished) state. “How hard could it be to finish them?”, I thought. The answer is that it took about four months to be ready for an initial release, but work has continued for the last five years through today, fixing bugs, improving metrics, adding and making changes to symbols, and so forth. Once I learned what was necessary to produce suitable text and math fonts for L^AT_EX, I continued to look for opportunities to bring new fonts to the T_EX world. I feel strongly that the future of Unicode T_EX is not in making use of system fonts, which are not the same for everyone, and can be suddenly changed or withdrawn, but with free fonts that have been enhanced enough to meet the needs of the demanding academic user.

As a side note, I add that I see many journals abandoning proprietary fonts in favor of free packages. With submissions using their specified free-font packages, the editorial work is reduced and the submitter gains by a reduction in the number of errors caused by reworking the submission to make use of a different, sometimes not completely compatible, package.

DW: Since your “first real font effort, getting things set up for proper fontinst run, and moving the output to the non-standard places expected by Textures”, your CTAN entry² suggests you have done a *lot* more with fonts and a good bit with L^AT_EX. What led you to get so deeply involved with fonts and can you talk about the different things that were involved with two or three of your font efforts?

MS: I never got over the thrill of seeing a page of mathematical content pop up on the screen with relatively little effort, especially compared with what had to be done to prepare manuscripts in the years prior to T_EX. For day to day use, I found that Computer Modern had some drawbacks, not copying well on machines available in the late 1980s, so homework and exams had to be made at very large sizes to be guaranteed to be readable. I started looking for other font systems, and purchased MathTime and Lucida, solving the copying problem. The Garamond+MathTime project after that reawakened my curiosity about T_EX and other fonts.

A second factor was that, after my cleanup period following retirement, I needed a serious interest to occupy my time and my mind, and T_EX seemed to qualify. I studied Hoenig’s book at length and spent several weeks trying to modify Hoenig’s math-inst Perl script so that it would actually work as expected. Though I eventually dropped the project, I learned in the process a great deal about the use

of fontinst to produce L^AT_EX math font support files, all of which has influenced my subsequent work on math fonts for T_EX.

Creating a font, and especially a math font, from scratch is a project of several years duration. I don’t normally create fonts, I try to rescue them. There are a number of font projects that have free licenses but were abandoned, in many cases because the author eventually needed to make a living. I’ve picked up some of those projects that seemed promising and worked to turn them into more polished products that can, I hope, be used for professional work.

My biggest project has been to rescue the text/math families TXfonts and PXfonts by Young Ryu, renaming them to newtx/newpx. They are based on the URW clones of Times and Palatino and mostly share a common math core, modulo scaling. Initially, the main issue was the metrics, and that took several months to correct. Later, I turned to a complete reworking of the math extension fonts, where the extensible delimiters were not matched well. This required about six weeks of effort. There followed a complete revision of all the delimiters, redrawing the smaller ones so that they worked better with the larger ones. After that came the conversion of formerly 7-bit math fonts to 8-bit, in an effort to economize on math families. Following that, I dropped the TXfonts and PXfonts text fonts and based the text fonts on the T_EX Gyre fonts, which had much more to offer, extending them with larger small caps which, in the case of newtx, were metrically equivalent to Adobe Times Small Cap fonts. I would estimate that I’ve spent over a year working on the project so far, and more is to come as it has become a headache to administer, and I’m trying to rework the entire package so that it is organized more rationally and can be more easily taken over by another person when I’m no longer capable. At last count, these packages are used, sometimes in conjunction with Libertine, to typeset at least a hundred journals, so this is quite an important issue.

I would say that LibertinusT1Math has to be considered my next most important math font package, based on Khaled Hosny’s LibertinusMath, a Unicode math font to match Libertinus, his fork of Libertine. Going backwards, so to speak, from a Unicode math font to a L^AT_EX math font family, involved adding several dozens of glyphs, mostly slanted integrals and extensible delimiters. I used STIX as my model for constructing 8-bit math fonts and a sty file, spending nearly four months on the project from the beginning to first release. Recently, at the urging of Claudio Beccari, I added a parallel sans serif branch to the math fonts so that (a) Liberti-

nusT1Math could be used without compromise as an ISO-compliant math font, and (b) the font could be used as a math font in which all alphanumerics (Roman and Greek) are sans serif. This took close to two weeks of additional work.

DW: What kind of feedback have you gotten from the community regarding your font work, and have you created a big maintenance task for yourself by your development work?

MS: The positive responses I've had from people in the *T_EX* world for whom I have great respect—Karl Berry, Boris Veytsman and Claudio Beccari in particular—have been very significant to me, helping me to sustain my energies. I've had a steady stream (one every two to four weeks) of email from users about bugs, to which I try to respond as quickly as I can. This is less of a burden than I was expecting. I also receive requests from users who would like some new features added. A few of these ask me to take on some very large scale new projects, and, while there are some interesting proposals to consider, my personal interest would have to be very high to agree to such an effort.

DW: Please talk about the tools and methods, both computer and anything non-computer you use in your font work? For instance, do you print out big proofs of changes and view them off a monitor, study books and font specimens, do glyph-by-glyph design and spacing, or does FontForge make it practical to be more efficient?

MS: I use mainly FontForge, but I turn constantly to Python to create special scripts to analyze and gather information about glyphs in a font. (The sfd file format used by FontForge is plain text and not hard to parse.) The free ttx program (part of fonttools) has also been useful, but recent versions have in some cases required much digging to repair problems caused by the installer. The version of FontForge I use is the binary with all required libraries built-in. Unfortunately, it does not allow external python scripts—i.e., FontForge was not built as an extension of python. I've tried to get this functionality when needed by building FontForge on a Linux machine as an extension of python. I don't use it frequently and find it highly bothersome, as it seems that I often have to reinstall a new version of Ubuntu and FontForge just to run a couple of scripts.

DW: Do you have esthetic preferences among fonts? Which was the most actual fun to work on, and the least?

MS: My esthetic preferences are not constant in time. A few years ago, I was most enthusiastic about

old-style fonts, which I take to mean a font either designed prior to the early eighteenth century, or a later revival of such a font. I liked their playful qualities and their close relationship to pre-sixteenth century formal handwriting. While in this frame of mind, I found fonts like Utopia and Charter austere and rather dull, seeming to strive for the opposite of old-style. My attitude is no longer so rigid, and for most of my current typesetting purposes, I find myself leaning toward plainer text fonts. It may be that I find them much easier to modify.

I enjoyed working on Garamondx at the time I did it, as the payoff was very pleasing, turning a rather limited font family into one that many people seem to find useful for academic writing, especially in the humanities. The least fun I've had was with Cochineal. It's not that I didn't really like Crimson, the font family it extends, but I agonized a great deal about whether the many months spent to rework glyphs and metrics outside the basic Latin area (e.g., extending Greek and Cyrillic in italic, bold and bold italic) would matter to anyone except me. In the end though, I was happy with Cochineal, and even happier when I learned that the *suftesi* package used by scholars in Italian humanities had taken this as its default font package.

DW: You have also created many pieces of *T_EX* utility software³. Please speak to your general approach/philosophy of creating tools to help you, versus pushing ahead brute force with a primary task, perhaps using a couple of examples.

MS: With font work, much of what I have done is indeed best characterized as brute force, but in a number of cases it was useful to turn the brute force approach into a script. Most of the software I have on github is either earlier versions of font packages or Python scripts, shell scripts or AppleScripts designed to automate some common workflows in *T_EX* on the Mac. Many are specific to TeXShop, Dick Koch's very fine front end to *T_EX* Live and some Mac-specific *T_EX* binaries. TeXShop provides a menu interface to AppleScripts that can be used to modify the *T_EX* source or provide access to some of its features. Some *T_EX* luminaries (e.g., Will Robertson of *fontspec* fame) have contributed scripts. I find the available scripts indispensable in my daily *T_EX* life.

DW: What is your view of how *T_EX* and its derivatives fit into the world today, e.g., within the evolution of the *T_EX* user community and its place (if any) in the larger world of typesetting, type design, and fonts?

MS: You may be asking something here that is beyond my pay grade. I interact mostly with academic

types, whether my mathematical colleagues who, if they write their own papers, do it with some version of \TeX , or with people who use my font packages. So far, the latter have all been academics or graduate students writing dissertations. In the case of my more decorative text fonts (Garamondx, newpx, Cochineal, BaskervilleF) the users are almost always from the humanities, and with a good number of them, matching Greek and Cyrillic is important, as are such supposedly deprecated encodings as OT2 and LGR. This has encouraged me to add support for them in font families that support those encodings, and to add the glyphs required for polytonic rather than monotonic Greek, where possible. I would be overjoyed if \TeX became important to a larger world in publishing, but I see no interest in this except for publishers of mathematical content, save for some Indian companies who have been very innovative with automating their output streams using \TeX .

I think there are people out there who will find \TeX more attractive with a wider selection of fonts that allow the full range of typeset options expected in academic writing—choices of figure styles, superscript and subscript styles, small caps, Greek and Cyrillic alphabets, and an accompanying math font. It has been my goal to provide more options for that group, which I think is critical to \TeX 's future.

DW: Has your work with \TeX , etc., led to broader interest in book design, etc.? And, with your considerable efforts with fonts and other aspects of \TeX , does that leave you time for non-typesetting enjoyment?

MS: I have not left myself with much time for other activities outside what I already do in \TeX , fonts and programs to support TeXShop, but my wife and I are opera fans and spend much of our spare time attending performances or watching/listening to electronic versions of them. There is now an extraordinary collection of historic opera on youtube and other Internet sites, the exploration of which is so absorbing as to concern me about my font future.

DW: Thank you very much for taking the time to participate in our interview series.

[Interview completed 2017-09-11]

Links

¹ <http://ebooksdownloads.xyz/search/general-theory-of-markov-processes>

² <https://ctan.org/author/id/sharpe>

³ <https://mjsharpe.github.io/tex-software/>

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

DW: An addendum: is there something you would like to show from your recent packages or recent additions to your packages? In addition, could you give us some idea of any new packages you are thinking about?

MS: (The previous paragraph used Cochineal, similar to Minion Pro. This paragraph uses XCharter.) I think the newish changes to XCharter are notable: (a) the addition of Cyrillic alphabets, including small caps, in T2A encoding; (b) the addition of Serbian Cyrillic italic glyphs. These can be used with pdflatex as well as xelatex and lualatex. To my knowledge, this is currently the only package which offers correct rendering of Serbian Cyrillic italic under pdflatex. You can see from this paragraph set in XCharter that it needs to be scaled down a bit, and has a substantially larger x-height than Computer Modern and Times.

The font in this paragraph is AlgolRevived, designed for typesetting coding, but marginally suitable for brief sections of text. (Unlike most coding fonts, it is not monospaced, though its figures are. Don't use this for Fortran.)

(Back to Cochineal.) Here's a sample of some mathematical text using sans math available in the libertinust1math package with preamble

```
\usepackage{libertine}
\usepackage[sansmath]{libertinust1math}
```

The text font in this paragraph is Libertine and math is sans serif (derived from STIX math sans) with symbols from libertinust1math.

$$\tilde{f}(\alpha) = \frac{1}{\sqrt{2\pi}} \int_0^t e^{-\alpha x} f(x) dx$$

My plans for the immediate future are to complete my reworking of newtxmath and newpxmath to a form where it is not hard to provide options that will allow the user to select from a list of alphabets for calligraphic, script, double-struck and fraktur alphabets without needing additional math groups. I would hope to do the same with libertinust1math. With the latter package, I'm not fond of the existing very small math binary relation symbols, and would try to add alternatives more like those in CM and Times.

With text fonts that have a much larger x-height than Times and CM, such as XCharter, Utopia, PTSerif, Erewhon and Century Schoolbook, a matching math font should be heavier and of larger x-height than Times math, and as far as I know, the Fourier package, which works currently only with Utopia and Century Schoolbook, is the only free choice. Given the time, I would like to work to extend Fourier math to have the same interchangeable features as newtxmath.

Advertising T_EX

Hans Hagen

I can get upset when I hear T_EXies boast about the virtues of T_EX compared to for instance Microsoft Word. Not that I feel responsible for defending a program that I never use(d) but attacking something for no good reason makes not much sense to me. It is especially annoying when the attack is accompanied by a presentation that looks pretty bad in design and typography. The best advertisements for T_EX should of course come from outside the T_EX community, by people impressed by its capabilities. How many T_EXies can really claim that Word is bad when they never tried to make something in it with a similar learning curve as they had in T_EX or the same amount of energy spent in editing and perfecting a word-processor-made document.

In movies where computer technology plays a role one can encounter weird assumptions about what computers and programs can do. Run into a server room, pull one disk out of a RAID-5 array and get all information from it. Connect some magic device to a usb port of a phone and copy all data from it in seconds. Run a high speed picture or fingerprint scan on a computer (probably on a remote machine) and show all pictures flying by. Okay, it's not so far from other unrealistic aspects in movies, like talking animals, so maybe it is just a metaphor for complexity and speed. When zapping channels on my television I saw figure 1 and as the media box permits replay I could make a picture. I have no clue what the movie was about or what movie it was so a reference is lacking here. Anyway it's interesting that seeing a lot of T_EX code flying by can impress someone: the viewer, even if no T_EXie will ever see that on the console unless in some error or tracing message and even then it's hard to get that amount. So, the viewer will never realize that what is seen is definitely not what a T_EXie wants to see.

So, as that kind of free advertisement doesn't promote T_EX well, what of an occasional mentioning of T_EX in highly-regarded literature? When reading "From bacteria to Bach and back, the evolution of minds" by Daniel Dennett I ran into the following:

In Microsoft Word, for instance, there are the typographical operations of superscript and subscript, as illustrated by

$\text{base}^{\text{power}}$

and

$\text{human}_{\text{female}}$

But try to add another superscript to $\text{base}^{\text{power}}$ —it *should* work, but it doesn't! In

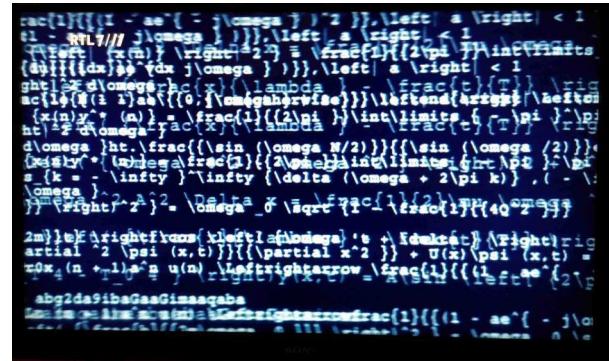


Figure 1: T_EX in a movie

mathematics, you can raise powers to powers to powers forever, but you can't get Microsoft Word to display these (there are other text-editing systems, such as TeX, that can). Now, are we sure that human languages make use of true recursion, or might some or all of them be more like Microsoft Word? Might our interpretation of grammars as recursive be rather an elegant mathematical idealization of the actual "moving parts" of a grammar?

Now, that book is a wonderfully interesting read and the author often refers to other sources. When one reads some reference (with a quote) then one assumes that what one reads is correct, and I have no reason to doubt Dennett in this. But this remark about T_EX has some curious inaccuracies.¹

First of all a textual raise or lower is normally not meant to be recursive. Nesting would have interesting consequences for the interline space so one will avoid it whenever possible. There are fonts that have superscript and subscript glyphs and even Unicode has slots for a bunch of characters. I'm not sure what Word does: take the special glyph or use a scaled down copy?

Then there is the reference to T_EX where we can accept that the "E" is not lowered but just kept as a regular "e". Actually the mentioning of nested scripts refers to typesetting math and that's what the superscripts and subscripts are for in T_EX. In math mode however, one will normally raise or lower symbols and numbers, not words: that happens in text mode.

While Word will use the regular text font when scripting in text mode, a T_EX user will either have to use a macro to make sure that the right size (and

¹ Of course one can wonder in general that when one encounters such an inaccuracy, how valid other examples and conclusions are. However, consistency in arguments and confirmation by other sources can help to counter this.

font) is used, or one can revert to math mode. But how to explain that one has to enter math and then explicitly choose the right font? Think of this:

```
efficient\high{efficient} or
efficient$^{\text{efficient}}$ or
\par
{\bf efficient\high{efficient} or
efficient$^{\text{efficient}}$}
```

Which gives (in Cambria)

efficient^{efficient} or **efficient^{efficient}** or
efficient^{efficient} or **efficient^{efficient}**

Now this,

```
efficient\high{efficient\high{efficient}} or
efficient$^{\text{efficient}}$^{\text{efficient}}$ or
\par
{\bf efficient\high{efficient\high{efficient}} or
efficient$^{\text{efficient}}$^{\text{efficient}}$}
```

will work okay but the math variant is probably quite frightening at a glance for an average Word user (or beginner in \TeX) and I can understand why someone would rather stick to click and point.

efficient^{efficient^{efficient}} or **efficient^{efficient^{efficient}}** or
efficient^{efficient^{efficient}} or **efficient^{efficient^{efficient}}**

Oh, and it's tempting to try the following:

```
efficient{\addff{f:superiors}efficient}
```

but that only works with fonts that have such a feature, like Cambria:

efficient^{efficient}

To come back to Dennett's remark: when typesetting math in Word, one just has to switch to the math editing mode and one can have nested scripts! And, when using \TeX one should not use math mode for text scripts. So in the end in both systems one has to know what one is doing, and both systems are equally capable.

The recursion example is needed in order to explain how (following recent ideas from Chomsky) for modern humans some recursive mechanism is needed in our wetware. Now, I won't go into details about that (as I can only mess up an excellent explanation) but if you want to refer to \TeX in some way, then expansion² of (either combined or not) snippets of knowledge might be a more interesting model than recursion, because much of what \TeX is capable of relates to expansion. But I leave that to others to explore.³

² Expanding macros actually works well with tail recursion.

³ One quickly starts thinking of how `\expandafter`, `\noexpand`, `\unexpanded`, `\protected` and other primitives can be applied to language, understanding and also misunderstanding.

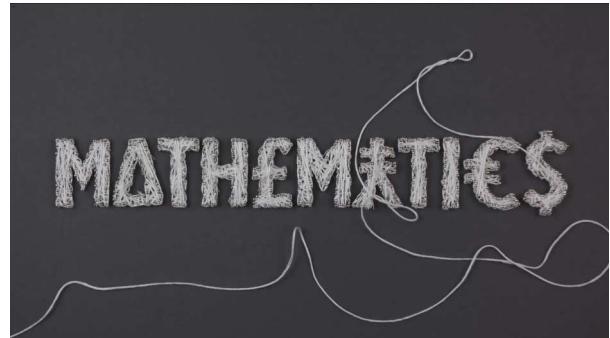


Figure 2: Nicer than \TeX

Now, comparing \TeX to Word is always kind of tricky: Word is a text editor with typesetting capabilities and \TeX is a typesetting engine with programming capabilities. Recursion is not really that relevant in this perspective. Endless recursion in scripts makes little sense and even \TeX has its limits there: the \TeX math engine only distinguishes three levels (text, script and scriptscript) and sometimes I'd like to have a level more. Deeper nesting is just more of scriptscript unless one explicitly enforces some style. So, it's recursive in the sense that there can be many levels, but it also sort of freezes at level three.

I love \TeX and I like what you can do with it and it keeps surprising me. And although mathematics is part of that, I seldom have to typeset math myself. So, I can't help that figure 2 impresses me more. It even has the so-familiar-to- \TeX ies dollar symbols in it: the poem "Poetry versus Orchestra" written by Hollie McNish, music composed by Jules Buckley and artwork by Martin Pyper (I have the DVD but you can also find it on YouTube). It reminds me of Don Knuth's talk at a TUG meeting. In *TUGboat* 31:2 (2010) you can read Don's announcement of his new typesetting engine *iTeX*: "Output can be automatically formatted for lasercutters, embroidery machines, 3D printers, milling machines, and other CNC devices ...". Now that is something that Word can't do!

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

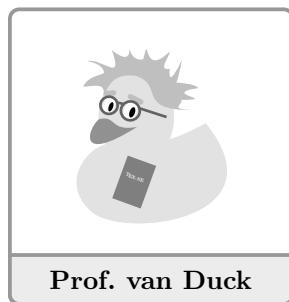
The DuckBoat — News from **T_EX.SE**: Asking effective questions

Herr Professor Paulinho van Duck

Abstract

Prof. van Duck would like to keep you up to date about the latest topics discussed in the chat of **T_EX StackExchange** (**T_EX.SE**), the famous Questions & Answers site dedicated to **T_EX**, **L^AT_EX** and friends. For this installment, he would also like to show you how to ask a good question on the same site, namely, how to have a rapid and smart answer by asking in the correct way.

1 Pleased to meet you!



Prof. van Duck

Hi, **T_EX/L^AT_EX** friends!
I am Herr Professor Paulinho van Duck, if you usually attend the **T_EX.SE** chat, you likely know me already.

I was born in São Paulo, Brazil, but now I live in Milan, Italy. I share a flat with my friend Carla, known as **Carl^AT_EX** on **T_EX.SE**. She also helps me writing my documents and managing my emails: typing is difficult when you do not have a pointy beak, quack!

I was named *Paulinho* after Paulo Cereda,¹ the author of the very convenient **arara** tool. He is developing a new version of it—we all are looking forward to using it.

Ulrike Fischer added the *van* before my surname as an allusion to the Dutch (not *Duck*) colonization of Brazil. I take this opportunity to thank her and her husband for making me become a donor for Mönchengladbach zoo.

Barbara Beeton—I think she needs no introduction—gives me the *Herr Professor* title, in order to make it (ridiculously) more formal.

Last but not least, I would like to thank samcarter for creating the **tikzducks** package;² my image and the other ducks you will find here are drawn by it.



I am a newbie with **L^AT_EX**, but I am very enthusiastic about it. I have read that beginner's level articles on this journal are very welcome. Hence, here I am!

¹ His advice about how to write this article was invaluable.

² <https://ctan.org/pkg/tikzducks>.

This is the first duck-column. If you like it, maybe there will be others. Do not hesitate to email me for suggestions about new topics (I have already in mind a **TikZ Quack Guide**) or for any criticisms.

2 Quacking in chat

If you haunt any programmers' milieux, you have probably already heard about the famous *editor war*: the rivalry between Emacs and Vim users always gives life to lively conversations also in our chat!

Recently a new war has come up, more or less with the same savagery: Italians vs. Rest of the World about the so-called *pineapple pizza*.

Since I live in Italy, I must say that that *thing* is not a *pizza* (also because, otherwise, Carla will throw me out). However, there are a lot of people, all around the world, who like these strange matched flavors.

If you join our chat in a quiet moment, just mention this subject to get a lively conversation.



Early this year, while we were quietly making fun of Paulo because of his never-ending thesis, my colleague Enrico Gregorio (egreg) shocked us by posting a question on **T_EX.SE Meta**. It happened that double backslashes disappeared from the code of some posts. If you know (L^A)**T_EX** at all, you will surely know how important backslashes are.

The issue was due to some software update by StackExchange, dated back to 2013, and it was impossible to address in an automated way.

Hence, David Carlisle, Moriambar, Barbara Beeton, and many others manually (or semi-manually, with some JavaScript) fixed the codes in about 10,000 questions and answers. We are grateful to all of them!

Now that all the errors are fixed, finally David can return to complaining about Enrico's stealing his ticks.



Another hot (even in the meteorological meaning) topic is the next **T_EX Users Group** meeting in Rio de Janeiro, Brazil.

Paulo Cereda is preparing some little (or should I say big?) surprises. Knowing him (and looking at the pictures³ he posted in chat), I think the next conference will be the funniest one ever organized.

There will also be a party at the famous Copacabana Beach and you will be able to enjoy the breath-taking sunsets at Ipanema!

³ From Twitter: <https://twitter.com/paulocereda/status/876420672683167745>.



Figure 1: TeX Users Group Meeting 2018 in Brazil.

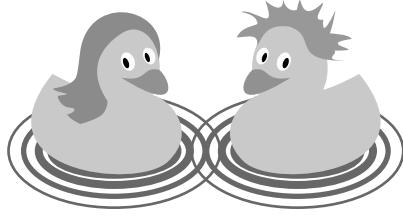
Therefore, prepare your bathing suit (if you have the *physique du rôle*, you can even dare a thong) and buy a flight to Brazil!

3 Quack Guide No. 1 Asking *effective* questions on TeX.SE

3.1 How you should *not* ask

Now and then, a question like the following appears on TeX.SE:

How can I draw this in LATEX?



Its flaws are (or should be) obvious: a generic title, followed by a more or less complicated picture (or table or whatever) without either an explanation of the problem or a single line of code.

We call these questions *just-do-it-for-me*, because they do not show any effort by the OP (i.e. the Original Poster, in TeX.SE jargon). Please, *never* post such stuff, a little duck cries when he sees it!

Another type of *bad* question is something like:

Why doesn't my code work?

My code worked till yesterday, but now it gives me an error. Why?

How can we answer that? Do you think we have a crystal ball? Or that we can read your mind?

What is the error? Where is an example of your code?



On the contrary, a smart user always follows these guidelines:

Van Duck's rules

1. Read the package manuals.
2. Look at the log generated by the code.
3. Search on TeX.SE and, in general, on the Internet.
4. Add a minimal working example (MWE) to your question.

The last point deserves to be examined in depth. Only a very few questions do not need a minimal working example, which is to say, a complete (but as short as possible) document which reproduces your problem. For example, if you are asking how to set a feature of your editor, probably an MWE is useless. For all the other ones, it is *indispensable*!

Nevertheless, it often happens that it is not added, especially by newbies.

My in-depth study of this phenomenon has led me to state:

Van Duck's equation

$$U_b = U_k + U_h + U_l$$

where the users who ask a *bad* question without an MWE (U_b) are divided into three categories:

U_k = who *do not know* what an MWE is

U_h = who vaguely know what an MWE is but do not know *how* to create it in a correct way

U_l = who know what an MWE is but are too *lazy* to add it.

Needless to say, we are particularly annoyed by the last type. For the rest of us, I hope this duck-column may be useful.



On TeX.SE there are a lot of pages which can help you with the asking process (see Table 1). I will try to sum them up very briefly in the next subsections. I would also like to suggest to you some little tricks, probably unknown by beginners.

3.2 What you should do *before* asking

One of the pluses of TeX & Co. is the abundant documentation — take advantage of it.

Table 1: Useful TeX.SE pages concerning the asking-a-question process

No.	URL	Description
1.	tex.meta.stackexchange.com/q/1436	Starter guide
2.	tex.stackexchange.com/q/162	List of online resources about TeX, L ^A T _E X and friends
3.	tex.stackexchange.com/help/searching	How to search within TeX.SE
4.	tex.stackexchange.com/help/how-to-ask	How to ask a good question
5.	tex.meta.stackexchange.com/q/228	What a minimal working example (MWE) is
6.	tex.meta.stackexchange.com/q/4407	How to create a minimal working example with bibliography (MWEB)
7.	tex.meta.stackexchange.com/q/1852	How to accept an answer

I know that reading the package manuals could be boring, sometimes even impossible. (I think nobody on earth has read all of the more than 1,000 pages of the TikZ & PGF manual!)

However, information such as incompatibilities or cautions in loading the package before/after others is usually written at the beginning. A rapid look at the documentation is mandatory!

In part I of the TikZ manual, for example, there is an excellent tutorial. Reading it is enough to start using this monster package.



Another fundamental tool is the log, first of all in the case of errors.

OK, TeX error descriptions are not the ultimate in clarity. A beginner is usually astonished when s/he discovers that *Undefined control sequence* merely means that a command is misspelled or the package it belongs to is not loaded.

However, if you search for those mysterious error messages on the Internet, almost always you find the solution or, at least, you understand what they mean.

In the log, there is also the position where the problem was detected, which (almost always) corresponds to the line of the wrong instruction.

Eventually, remember that the most important error is the first one; the others may be a consequence of that one.

But I don't want to teach you how to debug your code, there is already a gorgeous article by Barbara Beeton about it (*TUGboat* 38:2, p. 159, tug.org/TUGboat/tb38-2/tb119beet.pdf).

Instead, I would like to show you an effective command: `\listfiles`. It writes on your log the versions of all the packages you are using. It is useful in cases like: *Why does this code work on my friend's computer but not on mine?* or *Why does it work on ShareL^AT_EX but not on Overleaf?* Simply because there are different package versions. Many problems could be solved merely by updating your TeX distribution.

To see how it works, run this simple document:

```
\listfiles
\documentclass{article}
\usepackage{tikzducks}
\begin{document}
\begin{tikzpicture}
\duck
\end{tikzpicture}
\end{document}
```

Then, in your log, within

```
*File List*
...
*****
```

you will find many packages listed, with their version besides. Note that, in my example, I have loaded only `tikzducks` but the log shows all the packages loaded by `tikzducks` itself.

It allows you to avoid loading packages twice, unless you need to load them with a particular option, and it is also useful to detect hidden incompatibility.



Another huge source of information is the Internet. If you have a problem, it is very likely that someone else had the same problem before; search with your browser, an answer will probably appear. This may seem trivial, but I assure you that many times I've seen questions on TeX.SE which could have been rapidly solved that way, quack!

However, pay attention: like everything you find on the Internet, some information could be incorrect or obsolete. For instance, `\rm` in math has been deprecated for more than twenty years (you should use `\mathrm` instead) but it appears here and there on the net. Look at a couple of online resources before deciding to use anything. A list of reliable ones can be found in the post indicated in Table 1, No. 2.

You can also directly use the search field on the top right of TeX.SE main site home page (see Table 1, No. 3). Again, this may seem trivial, but every day we close questions because they are a duplicate of other ones!

3.3 How you *should* ask

If all your searching was fruitless, let us see how to ask a question.

First of all the title: be specific! Do not write things like *How can I do this in L^AT_EX?* or *Why doesn't my code work?* Many users read the questions only if they think (looking at the title) that they could answer. If your title is generic or unclear, you may miss the chance of having a prompt answer. Moreover, future users with the same problem as yours will not be able to easily find your post.

Second, in the body of your question give all the details needed to understand your problem. Don't put only the code or, even worse, only an image.

Third, for questions which need it (I dare say about 99.9% of the total on TeX.SE), add the most important thing: the MWE. Let us rapidly see the essential steps to follow to create it (for more details see Table 1, Nos. 5 and 6).



Do not forget your `\documentclass`. It is important to indicate it, even if you are asking something about a `tikzpicture` or a math expression. Many things can change if you are using `beamer` instead of `article`, for example.

Then, the packages: list all the packages needed to reproduce your problem, and only those, do not be verbose!

The same for your code. Put it within

```
\begin{document}
...
\end{document}
```

and add all and only the lines strictly necessary to reproduce your problem. Do not post only code snippets.

Remember to test your MWE before adding it to your question! You have to be sure it works or, if it does not work, it gives the same error you are struggling with.

It is useful not only for the ones who would like to answer but—believe me—also for you. I do not know how many times, while I was building my MWE, I found the solution by myself!

There are also many packages which help you to create an MWE.

For instance, `lipsum`⁴ and `blindtext`⁵ help you to produce some text with no meaning, just to fill your pages, preserving your privacy or copyright. `graphicx`⁶ allows you to use some example images. There is also an `mwe`⁷ package—guess what it is for!

Moreover, a testing-purpose one is `showframe`,⁸ which shows a simple diagram of the page layout. It is handy for detecting the infamous *Overfull \hbox* and for refining alignment in general. You have just to load it in your preamble and compile.

For example, this code gives an *Overfull \hbox*:

```
\documentclass{book}
\usepackage{showframe}
\usepackage{mwe}
\begin{document}
\blindtext

\includegraphics[width=\linewidth]{%
example-image-a}

\blindtext
\end{document}
```

If you run it, you will get the output shown in Figure 2. As you can see, the indentation error is clearly visible.



If your problem concerns bibliographies, you should also include your `.bib` file in your MWEB (minimal working example with bibliography). Of course, you do not have to include the complete file, but only the bibitems which cause your trouble.

The best practice is to create an MWEB with your `.bib` file embedded (in this way, people who would like to help you just have to cut and paste your code and compile it, and thus increase the probability of getting a quick answer).

The `filecontents*` environment⁹ is your friend in building such an MWEB.

It is also preferable, instead of inventing a new name for your test `.bib` file, to use `\jobname.bib`. This automatically uses the same name as the `.tex` file, changing only the suffix.

Here is a simple scheme you could follow:

⁴ <https://ctan.org/pkg/lipsum>.

⁵ <https://ctan.org/pkg/blindtext>.

⁶ <https://ctan.org/pkg/graphicx>.

⁷ <https://ctan.org/pkg/mwe>.

⁸ <https://ctan.org/pkg/showframe>.

⁹ There is also a `filecontents` package, <https://ctan.org/pkg/filecontents>.

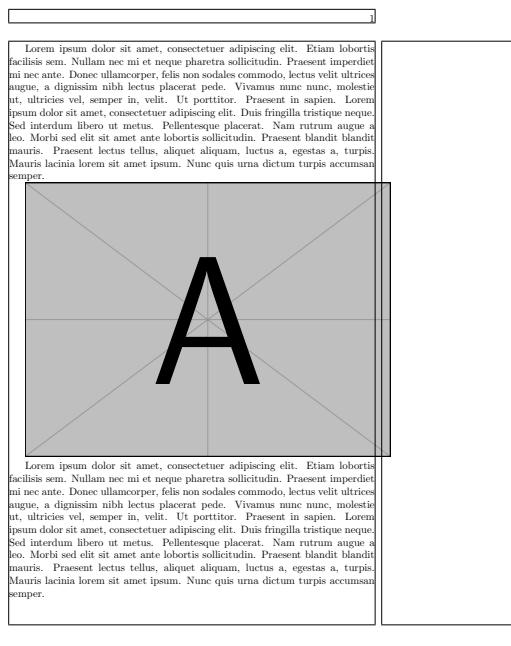


Figure 2: An example of usage of the `mwe` and `showframe` packages. The `Overfull \hbox` is immediately identifiable.

```
\begin{filecontents*}{\jobname.bib}
% put your bibitems here
...
\end{filecontents*}
\documentclass{...}
% put your packages here
% including the bibliographic one
...
% with biblatex:
\addbibresource{\jobname}
\begin{document}
% put here your code with your citation
...
% and bibliography printing, with biblatex:
\printbibliography
% or with other bibliographic packages:
\bibliography{\jobname}
\end{document}
```

Alternatively, if your problem is not strictly concerned with your own bibitems, you could use a test file included in the powerful `biblatex`¹⁰ package: `biblatex-examples.bib`.



¹⁰ <https://ctan.org/pkg/biblatex>.

Eventually, to complete your question, you have to add the correct tags. Again, there are users who read a question only if it is tagged in a way they think they could answer. Hence, tags are essential.

Look at the tag description before using it. For example, `latex3` concerns new material being developed by the L^AT_EX3 project, but it is often *wrongly* used as a generic tag for L^AT_EX questions.

3.4 What you should do *after* asking

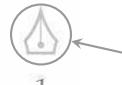
Once you have posted your question, your work is not over.

You have to pay attention to the possible comments of other users and reply to them; usually they ask for clarifications.

Finally, when someone posts the answer with the solution you were waiting for, you have to take a very important action: accept it by clicking the specific tick (see Table 1, No. 7). Remember that the users who answered are not paid for it; accepting the best answer is the correct way to say *Thank you!*

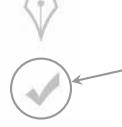
If you have more than 15 reputation points (and it is straightforward to get them if you post good questions), you can also upvote all the answers which are useful for you. Please do it, quack!

How to thank



1

Click here to *upvote* any useful post



✓

Click here to *accept* the best answer to your question

4 Conclusions

I hope you liked my explanation, and if you have trouble in asking a question, remember:

You are lucky, there's a ducky!

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

Review and summaries: *The History of Typographic Writing — The 20th century*
Volume 2 (ch. 6–8+), from 1950 to 2000

Charles Bigelow

Histoire de l'Écriture Typographique — le XXIème siècle; tome II/II, de 1950 à 2000. Jacques André, editorial direction. Atelier Perrouseaux, Gap, France, 2016, ISBN 978-2-36765-006-7, tinyurl.com/ja-xxieme-ii. 364 pp., 391 figures (illustrations, photos, diagrams, etc.), illustrated end papers. Also available as an ebook. The book is in French. Volume 1 (reviewed in *TUGboat* 38:1) covers the years 1900 to 1950; chapters 1–5 of volume 2 were reviewed in *TUGboat* 38:2.

Interpolated comments by the reviewer are in square brackets; the plain text summarizes and condenses the original writing, to the best of the reviewer's abilities.

6. Frank Adebaiye: The first commercial digital fonts (*Les premières fontes numériques commerciales*)

The first digital typesetting fonts were developed in the late 1960s for the Hell Digiset, the first digital typesetting machine. Digi Grotesk (1968), was a rasterized version of sans-serif Neuzeit Grotesk (c. 1930). In the 1970s, Hell produced original digital typeface families by Hermann Zapf, including Marconi (1976) and Edison (1978) for news display and text, respectively. [In the 1980s, Hell produced other original designs, including Aurelia (1982) by Hermann Zapf, and Isadora (1983) by Kris Holmes.]

American Linotype produced Bell Centennial (1976), a digital type family designed by Matthew Carter for telephone directories. Also for directories, Ladislás Mandel digitized his Galfra type in 1978.

In the 1980s, several manufacturers of digital typesetters plagiarized popular typefaces and marketed them under pseudonyms. [CB: The problem of typeface design protection is a recurrent thread in this chapter 6. For some history, see “Notes on typeface protection”, *TUGboat* 7:3, 1986, tug.org/TUGboat/tb07-3/tb16bigelow.pdf.]

In the mid-1980s, Xerox, Adobe, and Apple licensed and produced digital versions of Times and Helvetica for laser printers and personal computers.

Several other firms developed original digital designs in the 1980s. Among them were

- Bigelow & Holmes, with Lucida (1984) and Lucida Sans (1985);
- URW, with URW Grotesk (1985) and URW Antiqua (1985), both by Hermann Zapf;

Charles Bigelow

HERMANN ZAPF DESIGNS DIGISET

Marconi & Edison: Digital Types for Electric Communication

Marconi ABCDEFGHIJKLMNOPQRSTUVWXYZ
 abcdefghijklmnopqrstuvwxyz1234567890?

Marconi Italic ABCDEFGHIJKLMNOPQRSTUVWXYZ
 abcdefghijklmnopqrstuvwxyz1234567890?!

Marconi Bold ABCDEFGHIJKLMNOPQRSTUVWXYZ
 abcdefghijklmnopqrstuvwxyz123456789

Marconi Bold Italic ABCDEFGHIJKLMNOPQRSTUVWXYZ
 abcdefghijklmnopqrstuvwxyz12345678



In Digital Marconi, Hermann Zapf captures the elegance, symmetry, and brilliance of the classical Moderns cuts of Bodoni, Didot, & Walbaum.

For news display, Marconi has the large x-height, open counters, and strong detailing necessary for maximum readability at large sizes.

In designing Marconi for Digiset, Zapf drew directly on the original glock to achieve a complete matching of the typeface across all three weight levels.

Point by point, letter by letter, line by line, Marconi is a true digital typeface. It is also one of the world's most renowned type artes, for Digiset, the world's most experienced manufacturer of digital type.

Named in honor of Guglielmo Marconi, Nobel prize-winning pioneer of modern telegraphy, Digiset Marconi is a trademark of Hell & Reichenleitner.

Zapf's younger brother, Albert, is also a designer.

"Since childhood, I have had a special liking for electrical engineering. Everything connected with this caught my eye. For years I have studi-

ed electrical engineering and now have logic as a concept completely under digital control."

Hermann Zapf, About Alphabets

Digiset Editions, 1976, 198 pages, 19.5 x 27.5 cm, an intensive study of digital typography and his extensive work in designing digital rotary type.

This contains insights and examples for the toughest typographic tasks. Edition adds typefaces for the most important applications of the display. Edition on the signs of the times / from Hell & Reichenleitner.

Named in honor of electrical wizard, Thomas

Albert Edison, Digiset Editions in Rome, Italy.

Hell & Reichenleitner is a banner and electrolytic

firm in producing numerous

Edison ABCDEFGHIJKLMNOPQRSTUVWXYZ
 abcdefghijklmnopqrstuvwxyz12345678

Edison Italic ABCDEFGHIJKLMNOPQRSTUVWXYZ
 abcdefghijklmnopqrstuvwxyz1234567890

Edison Bold ABCDEFGHIJKLMNOPQRSTUVWXYZ
 abcdefghijklmnopqrstuvwxyz12345678

Edison Bold Italic ABCDEFGHIJKLMNOPQRSTUVWXYZ
 abcdefghijklmnopqrstuvwxyz12345678

- Donald Knuth, with Computer Modern (1980–1992);
- Bitstream, with Carmina (1987) by Gudrun Zapf von Hesse, Charter (1987) by Matthew Carter, and Amerigo (1987) by Gerard Unger;
- Adobe, with ITC Stone (1988) by Sumner Stone, Adobe Garamond (1989) and Utopia (1989), both by Rob Slimbach, Trajan (1989), Lithos (1989), and Charlemagne (1989), all by Carol Twombly;
- Agfa [formerly Compugraphic] produced the Rotis (1988) super-family of serif and sans-serif faces by Otl Aicher.
- “Punk” fonts issued from several firms, including: Emigré, with Matrix (1986) by Zuzana Licko, and FontFont, with Beowulf (1989) by Just van Rossum and Erik van Blokland.
- In France in the same decade, the independent type designers of typoGabor developed original fonts for Alphatype digital typesetters.

In the early 1990s, publication of Adobe's PostScript Type 1 font format and Apple–Microsoft's TrueType format resulted in a rush of new and original typefaces for the expanding digital typography market.

From FontShop and FontFont came FF Scala (1990) by Martin Majoor, FF Meta (1991) by Erik Spiekermann, and FF DIN (1995) by Albert-Jan Pool, among many others. From Apple came TrueType fonts for New York, Chicago, Monaco, and Geneva

(1991) by Bigelow & Holmes, first created as Macintosh bitmap fonts by Susan Kare (1984).

In 1992, Microsoft released TrueType fonts of Times New Roman, Arial, and Courier New, as well as a set of fonts equivalent to the Apple LaserWriter Plus set. Also in 1992, Microsoft released a large expansion of the Lucida family with new and original designs of Lucida Bright, Sans, Calligraphy, Blackletter, Handwriting, Fax, Typewriter, and mathematical symbol fonts, all from Bigelow & Holmes.

Windows users accepted Arial as a substitute for Helvetica, evidently being similar enough in appearance (and metrically identical), but Microsoft's distribution of "Book Antiqua" by Monotype (1992) sparked criticism from typographers that the face was a plagiarism of Zapf's Palatino. [The different reactions to Arial and Book Antiqua suggest that the grotesque sans-serif genre had become a "swarm" of vaguely similar and quasi-substitutable typefaces, whereas a distinctively artistic creation like Palatino could not be copied without being called either a travesty, or a rip-off, or both.]

Digital technology not only encouraged the creation of more typefaces but also enabled more complex designs, especially in the creation of alternate characters. In some cases, the design was ahead of the technology. In France, François Boltana created Champion, a joining script in English round-hand style, incorporating thousands of alternative glyphs, but a decade before OpenType made such fonts practical. Begun in 1989, the "Champion Pro" version was released in 2007. [In the late 1990s, other scripts with extensive alternate character sets included Apple Chancery (1993) by Kris Holmes (shown in the endpapers of this volume), Kolibri (1993) by Holmes for URW, and Zapfino (1999–2001) by Hermann Zapf.]

The growing free software movement also encouraged distribution of authorized free fonts (not piracies). The earliest (except for the CRT Hershey fonts) and most extensive free font family was Computer Modern by Donald Knuth, who also published the fonts' source code in his Metafont computer language. From Knuth's typography research lab and the American Mathematical Society came Hermann Zapf's AMS Euler (1985) family, freely distributed for mathematical composition. Adobe released Utopia for free distribution in 1993, and Bitstream followed suit with Charter. The 21st century would witness many more fonts released for free distribution.

[CB: In the article "Digital Typography" in *Scientific American*, August, 1983, this reviewer predicted that when digital font technology matured, there would "surely be a flowering of new letterforms

in the digital era." He is immensely grateful to the many industrious and imaginative type designers who subsequently created thousands of original typefaces to prove him correct. :-)]

Franck Jalleau: Third interlude: On the revival of typefaces (*Troisième pause : re-créer des caractères*)

"The revival of typefaces is an integral part of typographic evolution. It is a kind of art of heredity, of genetics, transmitted over generations, while introducing infinite variations and successive mutations."

Rehabilitation, revival, adaptation, interpretation, copy—there are many words to describe the translation of typefaces from one era to another, usually involving technological change. This includes the profound shift from handwriting to type. The process can be said to have begun with the earliest typography, when Gutenberg modeled his types on the formal handwriting of his era, and when capital letter forms based on surviving Roman inscriptions such as the Trajan column, were integrated with humanist minuscules, our lowercase. In the late 19th century, typeface revivals arose in an era of change in typesetting technology, and accelerated in the 20th century during further technological changes: a looking back to the past while questing toward the future. Each change in typographic technology imposed a need to adapt typefaces to new processes while preserving their inherent and inherited qualities created by designers of the past.

The revival of a typeface requires detailed analysis of the source materials, which may include different sizes of punches, matrices, and cast type, all preliminaries of the printed image, or, if the metal materials have been lost, the printed impression itself must be the sole guide. The shapes, their spacings, their "color" (gray tone in text) must be considered.

A revival is not a mere copy, but a new creation, an interpretation of the past. A revival in the digital era may need at least 256 characters to meet the de facto standard character repertoire, but the surviving materials of most historical typefaces have many fewer characters, so the modern designer must create new characters in the spirit of the originals. Moreover, modern typeface families often require a range of weights, but these do not exist for most classic typefaces, so again, they must be newly created in the spirit of the original. Different designers interpret classic typefaces differently. Revivals of the typefaces of Garamond, for example, differ greatly. Some are based on types of Jean Jannon, others on true specimens of Garamond, and may differ in weights, details, and proportions.

This chapter 6 concludes with illustrations of two examples of revivals with quite different approaches: Galliard (1978) by Matthew Carter, based on types cut by Robert Granjon circa 1570, and Francesco (2010) by Franck Jalleau, based on type cut by Francesco Griffó for Aldus (1499).

7. Olivier Jean: Working and office fonts from 1985 to 2000, between maturity and renewal (*Fontes de labeur et de bureautique de 1985 à 2000 : entre maturité et renouveau*)

In the 540 years of printing from Gutenberg to 1985, the craft of typography was practiced by a small number of people and the art of type design by even fewer. After 1985, however, mass marketing of personal computers and laser printers introduced typography to millions of ordinary users who became conversant with “fonts”, “points”, and other terms of typography. In this new era (termed “desktop publishing” in English), authors, editors, and publishers gained new freedoms as industrial obstacles fell away. As technical barriers to entry lowered, more people became authors, editors, and publishers. As the power of computer technology increased, the quality of fonts on screens and from printers improved. The path from personal computer to print media became more like a highway. [This trend did not stop with print. The vast expansion of the Internet and world wide web, along with improvements in screen resolutions and font rendering made text more readable, and more widely read, on computer screens, furthering the democratization of information compilation, organization, transmission, and reception.]

The influence of “system fonts”: by 1992, Apple and Microsoft bundled families of “default” system fonts: Times Roman (or Times New Roman), Helvetica (or Arial), Courier (or Courier New). These fostered a “meme” among computer users that there were three categories of fonts: serifed, sans-serif, and monospaced, each in a family of four variants: regular, italic, bold, bold italic.

Those basic families were initially sufficient for a majority of office workers and personal computer users, but as personal computing, printing, and publishing expanded and encompassed more applications and niches, the operating systems vendors added more and more fonts and developed fonts for other purposes, for example Lucida Console (1993) for terminal emulator windows, and Microsoft Verdana and Georgia by Matthew Carter (1996) for web usage. It is impossible to ignore Microsoft Comic Sans (1995) by Vincent Connare, which became highly popular yet widely reviled, like a beloved comic book villain, a font people love to hate.

A consequence of the lower cost of digital typography was its spread beyond the highly industrialized countries of North America and Western Europe to countries throughout the world, encouraging the development of the multilingual and multi-scriptal Unicode standard for encoding all the writing systems of the world.

When standardized font formats made typography cross-compatible, several proprietary font libraries were spun off as independent firms or were acquired by other firms. The proliferation of independent font designers and font vendors in the early days of desktop publishing began to coalesce by the end of the century, as smaller digital font firms were absorbed by larger ones. [This trend continued into the 21st century.]

In reaction to the loss of physical materials of traditional typographic heritage, some firms began programs of revival of classic typefaces. After Adobe Garamond, Adobe produced Adobe Caslon by Carol Twombly (1990). Matthew Carter revived large sizes of Caslon in Big Caslon (1994), and Adrian Frutiger reinterpreted Didot for Linotype (1991). There were also revivals of hand-written alphabets, including Adobe Trajan, Virgile (Roman rustics) by Franck Jalleau (1995), and Apple Chancery (Arrighi’s chancery cursive as taught by Lloyd Reynolds).

Digital technology enabled, and the burgeoning market encouraged, development of typeface “super-families” more extensive than in earlier eras. Examples include Lucida, ITC Stone, Rotis, Computer Modern, Thesis (1994) by Lucas de Groot, Le Monde (1994) by Jean François Porchez [the text fonts in these books]. Adobe released a new technology called “Multiple Master”, which enabled type users to modify and calibrate typefaces through a wide range of variations. This technology and its fonts were not commercially successful, however, and were cancelled before the year 2000.

Digital technology also enabled type designers to create typefaces for specific corporate clients and applications. Among many such, there were Colorado (1997) by Ladislás Mandel with Richard Southall, Telefont (1993) by Martin Majoor, and Le Monde and Parisine (1996) by Jean François Porchez.

As styles and variations expanded widely for Latin typography, digital tools in conjunction with the Unicode standard spurred expansion of non-Latin typography. Greek, Cyrillic, Arabic, Devanagari, Thai, Kanji, Chinese, Korean, and many other scripts were digitized. Lucida Sans Unicode (1994) was an early demonstration of Latin harmonized with non-Latin alphabets in a single TrueType font. The concept of integrating Latin and non-Latin scripts

in a single font was widely adopted and extended by the end of the century.

8. Hervé Aracil: Hybridization, (de)-construction and quotation—a view of typography from 1985 to 2000 (*Hybridation, (dé)-montage et citation – Un regard sur la typographie des années 1985–2000*)

Between the advent of the LaserWriter printer with Macintosh typographic fonts, and the end of the 20th century, typeface design enjoyed a period of richness and complexity that can be characterized by the words “transposition” and “reprise”. Translation from one medium to another is a form of transposition, and renewal of historical themes is a form of reprise. These interweaving tendencies in end-of-century typography, in opposition to prior movements like modernism, produced typographic phenomena equivalent to post-modernism and deconstruction in architecture, design, and music of the same period.

Digital technology, which made it easy to copy, cut, paste, and manipulate letterforms, aided these tendencies. One result was “hybridization”, the combination of characteristics from two or more different and distinct typeface categories. A forerunner was a hybrid letter ‘n’ combining sans-serif, slab-serif, and Elzevir serifs (think Times Roman serifs) in Thibaudeau’s *Manuel français de typographie moderne* (1915) [French Manual of Modern Typography; see accompanying figure].

Hybrid designs from the end of the century included, among many others:

- Prototype (1990) [capital + lowercase] by Jonathan Barnbrook,
 - Dead History (1990) [mixed bold rounded + modern] by P. Scott Makela,
 - Fudoni (1991) [Futura + Bodoni] by Max Kinsman,
 - Disturbance (1993) [capitals + lowercase] by Jeremy Tankard,
 - Walker (1993) (serif + ligature variations) by Matthew Carter,
 - Amplifier (1995) [slab + rounded + Clarendon] by Frank Heine;
 - variations of type design in coordination with literature included Quantage (1988) and Syntétik (1992) by Pierre di Sciullo.

In summary, such hybridizations, variations, and idiosyncrasies are not simply quaint experiments in typographic forms, but also constitute critical discourse on the philosophical bases of typography itself, for scholars, authors, readers, and designers to ponder and explore.



A spread from Thibaudeau's *Manuel*.

Alan Marshall: Fourth Interlude: On the preservation of typographic heritage (Quatrième pause : La préservation du patrimoine typographique)

The word “typography” has two meanings. Originally, it meant the composing and printing of texts with movable metal type, which stayed much the same from its invention by Gutenberg in mid-15th century to the 1970s. At the end of the 19th century, typography also came to mean publication layout or typographic design. Typography now means not only printing on paper but also text on signs, packages, media, and computer screens. Type is a fundamental element of visual communication affecting everyone in literate society, whether through ephemeral or enduring artifacts: cinema tickets, utility bills, restaurant menus, train schedules, posters, magazines, and books, printed or electronic.

The three pillars of typography are technology, aesthetics, and cognition. The technology of typography changed only incrementally from the 15th to the 19th century, but has since undergone a series of technological revolutions [described in earlier chapters of these volumes] which transformed typography into the basis of our information society.

Typographic aesthetics have also changed. The stiff typography of posters in the early 19th century became the exuberant letterforms on posters in the Belle Époque at the end of the 19th century. The graphical appearance of typographic documents comes from the co-evolution of type technology with the needs of society.

Cognition connects typographic technology to aesthetics. From the cuneiform tablet to the computer screen, text has always been interpreted by the same tools: our eyes and brain. The tools and media of visual communication are in constant evolution but perceived through our human instruments.

What is our typographic heritage, and why should we preserve it? Typographic heritage began with Gutenberg and continued in metal for centuries. Today, type is no longer metal, nor photographic, but computer data. Although type has thus been

dematerialized, it is essential to preserve as much as possible of the metal punches and matrices, as well as the negatives of the phototype era, as well as drawings, proofs, and specimens of characters, so that we can understand the ancestral processes, thoughts, aesthetics, and graphical forms that shaped our modern fonts and layouts. Type comprises not only letters, but also ornaments, fleurons, dingbats, and other graphical elements that have evolved over the centuries, tracing a rich history of abstractions and patterns, from Jean de Tournes to Giambattista Bodoni.

In addition to type itself, there are typographic manuals, catalogues, advertising, and lessons showing how type was classified, organized, understood, and intended to be used, as well as manifestos proclaiming how type should be used. These materials, publications, and documents of typographic history constitute a rich source of inspiration, information, and education for the future of typography, affected by constant reinvention not only of technology but of changes in taste, fashion, and social applications.

Typographic preservation is divided among diverse institutions, including libraries, archives, museums of paper, of printing, of computer history. Although it is impossible to preserve every sort of typographic and printing material in one establishment, such institutions prevent the total disappearance of the materials that allow us to analyze the evolution of the techniques and forms of graphic communication.

Typographic material from the pre-industrial era of typography, from Gutenberg to the 19th century, is now so rare and valuable that it is preserved without question, but more plentiful typographic materials from the 19th and early 20th century pose the question of what should be preserved with the regrettably limited funds available to museums and libraries. For the recent eras of phototype and early digital type, the question becomes evaluative—which of the now obsolete materials are more valuable and worth saving, and which can be lost?

[CB: The preservation of typographic materials illustrates a fundamental problem of “disruptive” versus “sustaining” technology. “Disruptive” technology is admired because it replaces older technology and institutions based on it, with newer, more efficient and effective methods. But, in typography, the disruptive shifts from metal type to phototype, and from photo to digital type caused the collapse of traditional metal font foundries and consequent loss of priceless collections of unique punches and matrices, hand-crafted by generations of uniquely skilled type artisans. The tangible, physical results of thousands of person-years spent carving and casting the

most intricate metalwork made by mankind—equivalent to centuries of fine jewelry making—were sold off for scrap. The fonts displayed on our personal computers, tablets, and smart phone, the fuel for worldwide social media, are in large part mere shadows of a deeper cultural heritage lost in a disruptive scramble.]

Thomas Huot Marchand:

Postface: The metamorphosis of typography
(*Postface : Les métamorphoses de la typographie*)

Despite several radical changes in technology, which accelerated in the 20th century, typography has shown an amazing permanence of forms. Many of the fonts in use today are modeled on typefaces of previous centuries. Technological changes have, however, altered several principles underlying the forms. These include flattening, abstraction, fluidity, and instantiation.

“Flattening” is a dematerialization of the formerly solid typographic object. Type was three-dimensional metal for 500 years, but phototype was two-dimensional film image for 50 years. Digital type describes two-dimensional forms but is not “material” per se, but is instead computer code. Flattened 2D type enables distortions and superimpositions not feasible in metal type.

Abstraction is a reduction of a form to a set of parameters and instructions instead of a graphic object engraved, drawn, or written. In particular, in the Metafont computer language devised by Donald Knuth in the late 1970s, the description of a character is based on variable parameters of a virtual path in a plane. Changing parameters alters the form. The concept of type as a prefabricated instance of writing is therefore opposed by the variability of programmatic typefaces like Knuth’s Computer Modern.

Fluidity: [CB: The French term here is “liquefaction”, evoking odd connotations in English.] Digital fonts and characters can “flow” from one computerized medium to another, e.g., pixel arrays on display screens, toner and ink arrays from laser and ink-jet printers. Digital text can be re-flowed on the screen when text block dimensions, kernings, line spacings, and other parameters are altered. Fonts and text are not bound to specific devices.

Instantiation: [CB: The French term here is “congélation”, meaning freezing. Although it makes a nice contrast to “liquefaction”, it doesn’t have a direct English translation.] In some digital font technology, pairs or sets of structurally similar characters along some dimension can be interpolated or extrapolated to generate new characters. For example, between a light weight letter ‘a’ and a bold

weight ‘a’, many other ‘a’s of intermediate weights can be interpolated. Interpolation was used in Ikarus software for type production, and was marketed to users by Adobe as Multiple Master fonts, and briefly supported by Apple in GX font technology. [The concepts have recently been reinvigorated as “Variable Fonts” (OpenType Font Variations).]

Emancipation: In the 20th century, designers were emancipated from the heavy machinery of typographic production [see previous chapters in this volume on phototype, transfer type, and digital type]. Type could be designed and produced more rapidly and less expensively, and digital type could be distributed through the Internet, enabling small independent digital type “foundries” to enter the font market.

Proliferation and concentration: Although emancipation enabled small type firms to proliferate, business circumstances within the font industry led to acquisitions of smaller firms by larger ones, resulting, for example, in the Monotype firm today, which, after its acquisitions of [ITC, Linotype, Bitstream, FontShop] now offers tens of thousands of digital fonts.

Streaming: [CB: The French term here is “évaporation”, which joins “congélation” and “liquéfaction” to make an analogy, whether intentional or not, between the physical states of water and fonts: solid, liquid, and vapor. My ad hoc translations do not capture this surprising analogy.] The widespread adoption of “web fonts” by most web browsers enables the streaming of fonts over the web. Adobe Typekit, Google Fonts, Monotype web fonts, and other firms provide on-the-fly downloading of fonts to documents, some for a fee, some free.

Is there a need for new fonts? Yes, more than ever. The great masterpieces of past type design should not lead us to believe there is no longer any place for invention. As Stanley Morison observed, “type design moves at the pace of the most conservative reader.” Over time, new designs appear and are added to our stock of earlier faces without rendering the latter obsolete. This series of books on the *History of Typographic Writing* reveals, in addition to major typographic trends, pathways seldom followed, type styles little known, designs rarely adopted. There should be a dialogue between typographic historians, theoreticians, and practitioners to integrate research in all these areas.

Extension to other writing systems: It may seem that there are more than enough fonts for Latin typography, with its history of typeface design and variation since Gutenberg, but non-Latin writing systems and scripts, including Arabic [as well as Indic scripts, Southeast Asian scripts, and East Asian scripts, which are often more complex and comprise more characters than Latin-based alphabets, open up new horizons for typographic creativity around the world. There are today more than 120,000 fonts in 129 writing systems.

Bibliography and end materials

The bibliography for volume II contains 405 entries, subdivided into: (a) earlier volumes in the series; (b) encyclopedias, dictionaries, and inventories on typography; (c) specimens; (d) printing, typography, book arts: general history, theory, technology; (e) history of typography and graphics arts 1900–2000; plus sections with bibliographic references for each chapter and interlude. This supplements, with some overlap, the 412 entries in the bibliography of Volume I.

Illustrations. As with the first volume, this second volume is profusely illustrated, containing some 391 figures and 7 miscellaneous images and endpapers.

Indexes. There is a three page index to typefaces cited in the texts, and a six page general and typographic index.

Awards. At the 2017 Perpignan International festival of books on art, architecture, photography, cinema, and graphics, the two volumes of *Histoire de l’Écriture Typographique* won the prize for best book on graphics—a well-deserved honor.

In conclusion. The reviewer has provided extensive summaries of the chapters because these two volumes are unique in their extensive survey of 20th century typography and therefore merit the attention of English language readers. Depending on subject matter and potential readership, certain chapters would be worthwhile in stand-alone English translations, and a translation of the whole would greatly benefit typographic scholarship.

Explicit Liber.

◊ Charles Bigelow
lucidafonts.com

Serifed Greek type: Is it "Greek"?

Antonis Tsolomitis

1 Introduction

I grew up with this idea around me: serifs are not "Greek". Fonts that use them in the Greek alphabet are "latinizations" of the form of the Greek letters. A kind of æsthetic imperialism. And indeed, many people still believe this. This idea has been pushed to extremes, so that for example Matthew Carter in his article "Which came first, the Greeks or the Romans?" (see [1]) feels the need to apologize for having fallen into this kind of "sin", saying more or less that the only excuse he has is that this was the demand of the Greek market at the time he designed beautiful fonts, such as his Greek Baskerville.

I am now convinced: this is simply false. In short, it is false because it is based on the way the first Greek fonts were developed and not on the history of the Greek forms themselves.

2 What is "Greek"?

Many changes have occurred on the Greek peninsula in the last 3000 years. Many things found in this vast amount of time may be considered non-Greek. What can not be considered to be "non-Greek" is the writing of Greek people until the time that the Greek Gods stopped being worshipped by the great majority of the inhabitants, approximately at the time of what came to be known as the "Byzantine Empire". Several arguments can be made against the Byzantine era being deemed a Greek era. But no such argument can be made for the era before that. Let us see such an important example. We are in 156 CE. Herodes Atticus, a rich and generous Roman, lives in Athens, isolated from Rome and fully integrated into Athenian society. His younger daughter Markia Athenais dies of an unknown illness. The supreme court of Athens (*Άρειος Πάγος*) before Athenais' death votes for putting a statue of her in the Asklepion, the temple of the God of Health and his daughter Hygeia, on the south slope of the Parthenon, to ask the God for his help. The Greek engravers write on Pentelic marble (the most famous marble of that era) the inscription seen in Figure 1. Is this Greek script? It definitely is. Does it have serifs? It is full of serifs of several kinds. The text (with spaces and punctuation to facilitate reading) is reproduced in Figure 2 with the new titling font "Athenais", in honour of the family of Herodes Atticus whose generosity offered so much to Athens. The text is fully serifed and more than that, it has three ligatures: Ν━━━Ν in the word "ΨΗΦΙΣΑΜΕΝΗΣ" (=voted), Α━━━Α in

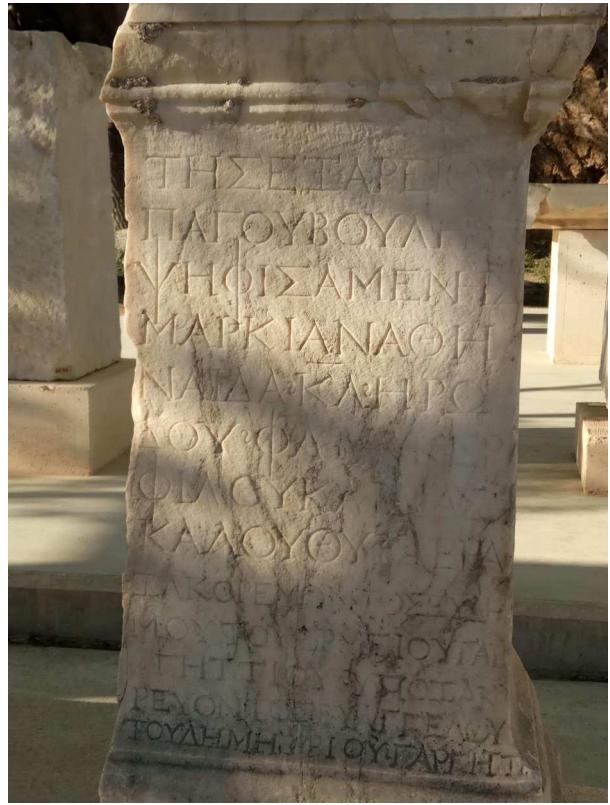


Figure 1: The pedestal NK14 from the archaeological site of the Athens Parthenon.

the word "ΔΙΔΑΣΚΑΛΟΥ" (=teacher), and Ξ━━Ξ in the word "ΘΥΓΑΤΕΡΑ" (=daughter). The letters Φ and Ψ dramatically extend below the baseline and above the capital X-height. The letter Ζ is the letter Zeta (crossing it becomes a Ξ, Xi). The symbol ϛ is an ornament. The inscription also contains many alternative characters that we will discuss below.

3 Origin of the pedestal

This pedestal is exposed today in the Asklepion in the south slope of Athens' Parthenon. It is located exactly on the "Peripatos" (=walking path), next to the Herodion theater. It was brought to my attention by friends from YSEE, the Supreme Council of Hellenes Ethnikoi, the people that continue to worship the Greek Gods and was recently recognized by the Greek State as one of the few religions in Greece that can produce legally binding results (e.g., marriages).

According to the Athens Ephorate of Antiquities [2], the pedestal is classified as NK14 (see [3], [4] or [5]), it is made of five fragments of Pentelic marble welded together. It was found in 1876 in the Asklepion area, built into the foundations of a Christian church. It has a rich nape and base. The top

ΤΗΣ ΕΞ ΑΡΕΙΟΥ
 ΠΑΓΟΥ ΒΟΥΛΗΣ
 ΨΗΦΙΣΑΜΕΝΗΣ
 ΜΑΡΚΙΑΝ ΑΘΗ-
 ΝΑΙΔΑΙΚΛΗΡΩ-
 ΔΟΥ·ΦΛ·ΜΑΚΕΡ,
 ΦΙΛΟΥ ΚΑΙ ΔΙΑΣ-
 ΚΑΛΟΥ ΘΥΓΑΤΕΡΑ.
 ΙΑΚΟΡΕΥΟΝΤΟΣ ΕΥΔΗ-
 ΜΟΥ ΤΟΥ ΕΡΜΕΙΟΥ ΓΑΡ-
 ΓΗΤΤΙΟΥ·ΥΠΟΙΑΚΟ-
 ΡΕΥΟΝΤΟΣ ΕΥΑΓΓΕΛΟΥ
 ΤΟΥ ΔΗΜΗΤΡΙΟΥ·ΓΑΡΓΗΤ.

Figure 2: The text from the pedestal, typeset in the new Athenais font.

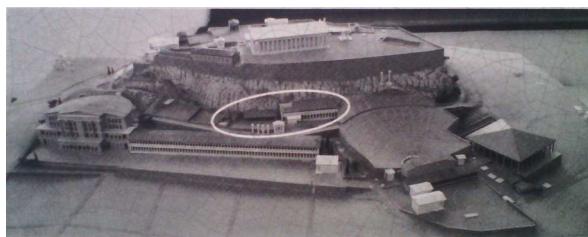


Figure 3: The location of the temple in the area of Athens Parthenon.

surface has a recess for the placement of the statue of Markia Αθηναῖς (full name: Marcia Annia Claudia Alkia Athenais Gavidia Latiaria), the younger daughter of Herodes Atticus. The area was re-organized after the discovery of the Asklepieion temple in 2011 (of course, only the foundations were found); there has been some partial restoration of a few columns (see Figure 4), and the pedestal was put in place recently.

The people of YSEE asked me if I could digitize the lettering as a font. The result of this work is demonstrated above. The font, named "Athenais", is available from the YSEE site (<http://www.ysee.gr>).

4 More about the font

The font was designed so that one can use the idea of the extended Φ and Ψ. Since these letters appear rarely in Greek text, more letters use the extended form in this font. These are

Ι Γ Κ Π Τ Τ Υ Χ Χ

All extended characters have several heights.



Figure 4: The partially restored temple as it stands today.

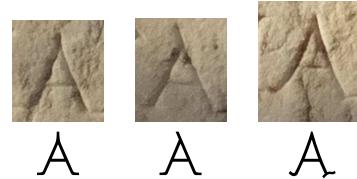
As an example, here is the letter Τ (the first is at regular height):

Τ Τ Τ Τ Τ Τ

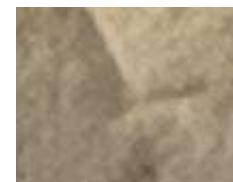
Alternative characters are provided and these are:

Α Α Ζ Λ Ν Ζ Σ Τ Φ

In particular, Alpha is given in three forms as it appears on the pedestal:



The last Alpha is very interesting in that it uses a swash type serif which looks like this magnified:



The serif of Ψ is also very interesting as it reminds us of serifs from Palatino (say of Palatino X):



Another type of serif is used on the top right of Μ. In the next magnification check that the top left and top right serifs are not the same:



Serifed Greek type: Is it "Greek"?

The ligatures provided are:

A NE NH TZ TY TE E YZ.

In the modern world one can not escape the need for Arabic numbers and some punctuation. So these have been added and the numbers 4, 6, 7 and 9 are provided in variable heights:

0123444445666667777778999999

5 Use of the font

In order to make it possible to use the font in a beautiful way one needs to scale parts of the title she¹ is typesetting. And then the need arises to balance the weight of the scaled parts. Thus the font is provided in several weights to make this possible. For example, if we write "University of the Aegean" in Greek, we may do it this way:

ΠΑΝΕΠΙΣΤΗΜΙΟ
ΑΓΑΙΟΥ

The first word is not at the same size as the second word so that the extended letters can interact nicely. To balance the color we used a heavier version for the first word.

6 Comments on the lowercase Greek

The font does not contain lowercase. It is a titling font. In this section we return to the first question we posed. Are serifs a non-Greek characteristic? Lowercase appeared after the 8th century CE as an alteration of the capital letters. They appeared when the monks were trying to rewrite the ancient texts, since they were in great demand from the West (and paid well). They consciously altered the forms of the Greek capital letters so that they could be written with fewer strokes, which saved them time and increased their income. Later, people such as Aldus Manutius and Claude Garamond designed the Greek lowercase letters in fonts such as Grec de Roi. The monks' writing, as well as Grec de Roi, is very hard to read. In my opinion it is not just an alteration of the Greek letters. It is clearly a deterioration of the letters as a result of speedy and bad quality writing of the monks.

Is this Greek? Let us follow this line of thinking: some people, that consider themselves non-English by their own writings, copy texts of the best English calligraphers and really destroy their form. And the result is "English" letters! No reasonable person can accept this.

¹ I dislike slashes in text such as he/she. A coin is flipped, I will write in feminine.

I think it is proper to say the following: the existence of lowercase letters is a fact and we are used to them. To return them to forms that match capital Greek serifed letters is only making them Greek and not Latin.

7 Conclusion

A serifed Greek font may be beautifully designed or ugly. But the existence of serifs or their absence can not justify Greek or Latin characteristics.

A serious designer that respects the Greek culture and with intentions not to cause any harm, but rather to promote type art, can decide to use or not to use serifs without blaming herself for "latinization" or anything else.

ΜΑΡΚΙΑ
ΑΘΗΝΑΙΔΑ

References

- [1] Matthew Carter, *Which came first, the Greeks or the Romans?* Greek Letters, From Tablets to Pixels, ed. Michael Makrakis, Oak Knoll Press, 1996.
- [2] Personal communication.
- [3] Στ. Α. Κουμανούδης, Αθήναιον 5 (1876) 324 αρ. 3.
- [4] *IG II²* 4073.
- [5] Levensohn M., *Inscriptions on the South Slope of the Acropolis*, Hesperia 16, 1947, 73 map number 125, fig. 1.

◊ Antonis Tsolomitis
University of the Aegean
Department of Mathematics
832 00 Karlovassi
Samos, Greece
<http://myria.math.aegean.gr/~atsol>

Addendum. This article uses GFSNeoHellenic at 11 pt for the main text font, designed by the Greek Font Society in 1993–1994 (ctan.org/pkg/gfs), sponsored by the Archæological Society at Athens. The samples are typeset in the new Athenais font.

ConTeXt for beginners

Willi Egger

Abstract

In 2017 we had a joint meeting of BachoTeX and TUG. During this conference a workshop for ConTeXt beginners was requested. The following article comes from this workshop.

As with any typesetting system offering possibilities to handle virtually any project, ConTeXt is a huge system. As with the workshop, this article can only lift the veil a little. The workshop was a hands-on session for playing with basic elements to create a document. Towards the end there was a chance to work on a small project—a single-sided document containing all the elements to build an invoice.

Keywords: ConTeXt, beginner, workshop

1 About ConTeXt

TeX, developed by Donald Knuth in the 1970s and 1980s, is still widely used. There are three principal flavours: Plain TeX, L^AT_EX and ConTeXt.

ConTeXt is an advanced macro package which uses TeX as an engine. ConTeXt is a development of PRAGMA ADE in Hasselt, The Netherlands. The code base was written and continues to be actively maintained by Hans Hagen. It was first developed for typesetting schoolbooks and school math.

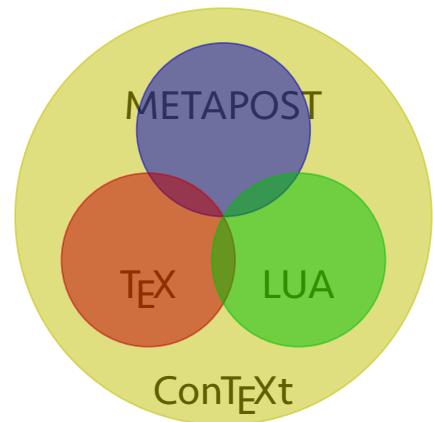
At present two versions of ConTeXt are in use. ConTeXt MkII makes use of the pdfTeX engine. This version is not developed any further and the code is frozen, but bug-fixes are still applied. The current version ConTeXt MkIV makes use of LuaTeX and is under continuous development. For new users of ConTeXt we advise starting with ConTeXt MkIV with LuaTeX.

In contrast to other macro packages ConTeXt is a primarily monolithic system. Essentially all facilities are provided out of the box. Nevertheless, some modules for add-on features, written by third parties, can be installed and invoked as necessary.

ConTeXt makes heavy use of MetaPost. The latter is therefore used as a library which makes it possible to generate graphics at runtime very efficiently.

ConTeXt can be used for processing TeX-coded documents but it also provides a fully developed environment to process XML-coded data. ConTeXt has interface capabilities with SQL databases.

Graphically the ConTeXt-typesetting system can be presented as follows:



2 Availability

ConTeXt is included in TeX Live, so you may well already have it. However, for users who wish to use ConTeXt extensively or exclusively, we recommend the standalone release, which can be obtained from the ConTeXt garden wiki: wiki.contextgarden.net. The garden also has information on installation of the system for current operating systems, notably Windows, Mac OS X, and Linux.

3 Documentation

ConTeXt is a highly complex typesetting system. It comes with many detailed manuals, many of which are included in the distribution, and all can be downloaded from the Pragma website (pragma-ade.com). Along with this, there are a couple of printed books; these are available from H₂O-books (see also section 8).

4 Basic elements

A basic difference compared to other macro packages is that in ConTeXt almost all commands are defined as structured elements, enclosed by `\start ... \stop` commands. This is a prerequisite for working with XML and provides much control over the beginning and end of an element.

Each document starts with `\starttext` and ends with `\stoptext`. Other examples of such structured commands are:

- `\startchapter ... \stopchapter`
- `\startsection ... \stopsection`
- `\startplacefigure ... \stopplacefigure`

The system comes with a reasonable set of presets for many constructs that are needed while building a new document. ConTeXt provides maximum flexibility to adjust almost anything according to the user's wishes. For this purpose most commands

come with a `\setup...` command, where appropriate variables can be customized. A couple of examples, taking options in square brackets (common in ConTeXt):

- `\setupframedtext[...][...]`
- `\setuplayer[...][...]`

It is important to note that, in case it's needed, ConTeXt still understands most of Plain TeX.

5 Export formats

When running a document with ConTeXt the output is by default PDF. Correctly coded documents can also be exported as XML, HTML, XHTML and ePub.

6 A first ConTeXt document

As with other TeX environments, it is best to use a text editor for coding.

For the first document we can open a TeX file, say `myfirstfile.tex` and type e.g.

```
\starttext
Dzień dobry at the \CONTEXT
beginners tutorial!
\stoptext
```

After saving this file, we can compile it from the terminal with `context myfirstfile`. Provided that the installation was successful it will result in a document typeset as A4 with the name `myfirstfile.pdf`.

It is worth mentioning that ConTeXt is UTF-8 aware out of the box, so typesetting accented glyphs, as above, is no problem.

7 Elements of a document

Now that we know that the installation is fine, we can start describing the common elements of our documents. In a second step these elements will be used in a small project for creating a simple invoice.

We will deal in this first step with the paper size, the general layout of the page, fonts, two types of table environments, headers and footers, and the layer mechanism and the basic placement of graphical elements.

7.1 Paper size

By default ConTeXt produces A4 pages, but of course one is in no way bound to this format. The system comes with many predefined paper sizes including the DIN-A series, American paper sizes and oversized paper sizes for the print industry.

Because ConTeXt produces A4 by default we do not have to set up the paper size for that case. However if you are using letter size you will have to tell this to the system in the preamble of each document:

```
\setuppapersize[letter][letter]
```

To set a custom paper size the following two commands are needed:

```
\definepapersize[Mysize] [width=80mm,height=95mm]
\setuppapersize[Mysize] [A4]
```

The `\setuppapersize` command accepts two arguments, used to make up the page and place it on paper. The first argument tells ConTeXt the size of the page, and the second argument tells the system how to put the page on a certain paper size for printing. To both arguments one can add the attributes `portrait` or `landscape`. So the above defines a small layout sized, to be placed on A4 paper.

Of course one often wants the layout size and printing size to be the same, so both arguments are the same:

```
\definepapersize[Mysize] [width=80mm,height=95mm]
\setuppapersize[Mysize] [Mysize]
```

Finally, we do not need to define our own paper sizes when using any of the predefined ones. For instance, to place an A4 page on an A3 in landscape:

```
\setuppapersize[A4] [A3,landscape]
```

7.2 Page layout

Although ConTeXt comes out of the box with a decent set of presets one has all sort of possibilities to adapt a layout of a page to the needs at hand. In order to be able to work with this rather extensive command it is necessary to have a look at the division of the space of a page.

The following drawing shows all regions which we can manipulate individually: the outermost areas indicated by `top` and `bottom` as well as `leftmargin` and `leftedge` and such. The edges are normally only used in interactive documents like presentations. We will not discuss those here.

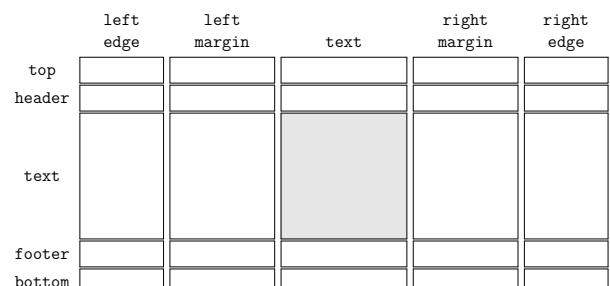


Figure 1: Page layout areas

The one command to set up these different regions of the page is `\setuplayout`. An example:

```
\setuplayout
  [topspace=15mm,
   backspace=15mm,
   header=0pt,
   footer=1.2\bodyfontsize,
   height=middle,
   width=middle]
```

This command has many arguments and needs to be studied in the documentation. Designing a page layout is done always on a right-hand (odd) page. When we need a double-sided document we tell this to ConTeXt and the system will automatically switch the backspace and eventually-determined margins to the right position on the even page.

- The key `topspace` gives the white space above the content, and key `backspace` denotes the white space left of the content.
- Normally ConTeXt has an active header. If we do not want a header we switch the key `header` to zero, using any of the usual TeX dimensions, e.g. pt, mm, cm, in. When the header is zero, automatically the header-distance (not shown above; it's the space below the header and above the text) is set to zero too. The same applies to the footer.
- The key `height` specifies the total height in which any content can occur, i.e. it includes the header, header-distance, text body, footer-distance and footer. If we give `height=middle`, the system will calculate the height of the typesetting area such that there is an even distribution of white on top and bottom of the page.
- The key `width` allows us to set the width of the typesetting area for the header, text-body and footer. `width=middle` causes ConTeXt to calculate the width such that at the right side of the typesetting area is the same amount of white space as the `backspace`.
- If the document has content in the margins, we can set up margins for the left and right, as well as the distance they should get from the text body. Margin content is placed into the backspace and adjacent to the text body at the right side. When using margins we need to adjust the backspace and the text body width in order to keep this content on the paper.

7.2.1 Show me the layout

Specifying your page layout can be difficult in the beginning. In order to facilitate this there is a handy command to show the actual page layout graphically: you can issue `\showframe` inside `\starttext ... \stoptext` or outside at the top of the docu-

ment. For an example of how this looks, see the appendix.

If you want to know all the dimensions set for the page you can put `\showlayout` into the document. It will show all dimensions on one page. The default values of page parameters in ConTeXt are as follows.

<code>\paperheight</code>	29.7000cm
<code>\paperwidth</code>	21.0000cm
<code>\printpaperheight</code>	29.7000cm
<code>\printpaperwidth</code>	21.0000cm
<code>\topspace</code>	2.4998cm
<code>\backspace</code>	2.5000cm
<code>\makeupheight</code>	25.0000cm
<code>\makeupwidth</code>	14.9999cm
<code>\topheight</code>	0.0000cm
<code>\topdistance</code>	0.0000cm
<code>\headerheight</code>	1.9999cm
<code>\headerdistance</code>	0.0000cm
<code>\textheight</code>	21.0002cm
<code>\footerdistance</code>	0.0000cm
<code>\footerheight</code>	1.9999cm
<code>\bottomdistance</code>	0.0000cm
<code>\bottomheight</code>	0.0000cm
<code>\leftedgewidth</code>	0.0000cm
<code>\leftedgedistance</code>	0.0000cm
<code>\leftmarginwidth</code>	2.6564cm
<code>\leftmargindistance</code>	0.4217cm
<code>\textwidth</code>	14.9999cm
<code>\rightmargindistance</code>	0.4217cm
<code>\rightmarginwidth</code>	2.6564cm
<code>\rightedgedistance</code>	0.0000cm
<code>\rightedgewidth</code>	0.0000pt
<code>\bodyfontsize</code>	10.0000pt
<code>\lineheight</code>	12.5720pt
<code>\strutheightfactor</code>	.72
<code>\strutdepthfactor</code>	.28
<code>\topskipfactor</code>	1.0
<code>\maxdepthfactor</code>	0.4

7.2.2 Header and footer

If we have set up the page to carry a header and/or a footer with `\setuplayout`, we can fill those areas with content. The commands to put content into the header and footer are:

```
\setupheadertexts[lo][ro][re][le]
\setupfootertexts[lo][ro][re][le]
```

In double sided documents we can have four different pieces of content, two for the left (odd) and two for the right (even) page. For a single sided document we need only two arguments. The above letters `lo .. re` are placeholders; to explain their meaning:

```

lo  text left on odd pages
ro  text right on odd pages
re  text right on even pages
le  text left on even pages

```

We can give normal text into the fields or add commands, e.g. insert the page number or the current section title. Assuming we have a single sided document, we might do

```

\setupheadertexts
  [{\getmarking[chapter]}]
  [\userpagenumber/\totalnumberofpages]
\setupfootertexts
  [\jobname]
  [\currentdate]

```

We could also put information in the header margin as shown in figure 1. In a double-sided document it could be

```

\setupheadertexts
  [margin] [\userpagenumber] [Document A]

```

In a single-sided document it would be

```
\setupheadertexts[margin] [\userpagenumber]
```

7.2.3 Marginal text

For a document with content in the margin we have to set this up with the `\setuplayout` command. For instance we could set the parameters to the following values

<code>topspace</code>	1.5cm
<code>backspace</code>	2.5cm
<code>leftmargin</code>	2.0cm
<code>leftmargindistance</code>	0.2cm
<code>textwidth</code>	15.0cm
<code>rightmargindistance</code>	0.2cm
<code>rightmargin</code>	2.0cm

In the text, we can then say

```

\inrightmargin {\tfxx text in the margin.}
to get marginal text as shown here. (\tfxx selects a
small typewriter font; we'll briefly discuss fonts later,
but see wiki.contextgarden.net/Font\_Switching
for more.)

```

7.3 Tables

7.3.1 Tabulation

The tabulation environment is specially suited to typeset tables in the text flow. The environment provides many facilities to customize a table, but there is no support for vertical rules. This environment uses a template at the beginning of the table, as shown here:

```

\starttabulate[|l|c|r|p|]
\NC left \NC center \NC right \NC para \NC\NR
\stoptabulate

```

The `|` characters in the template here merely delimit column specifications, and do not indicate

rules to be typeset. Additional option characters can be added in a column as follows.

Options for column width:

<code>w(dim)</code>	fixed column width
<code>p(dim)</code>	fixed paragraph width
<code>p</code>	maximum width paragraph

Options for style:

<code>B,I,R,S,T</code>	bold, italic, roman, slanted, typewriter
<code>m,M</code>	inline math, display math

Then, within the tabulate body, `\NC` indicates the next column and `\NR` the next row, as shown above. Other commands can be included, for example, these can add vertical space between rows:

<code>\TB[halffine]</code>	Vertical space of half a line
<code>\TB[line]</code>	Vertical space of a whole line
<code>\TB[1cm]</code>	Vertical space of 1 cm

7.3.2 Natural tables

Natural tables are an environment for large tables and provide a huge set of possibilities to customize cells, columns and or rows. The coding is generally similar to HTML. A basic table looks like this:

```

\bTABLE
  \bTR \bTD a \eTD \bTD x \eTD \bTD y \eTD \eTR
  \bTR \bTD b \eTD \bTD x \eTD \bTD y \eTD \eTR
  ...
\eTABLE

```

Thus, the environment is started with the command `\bTABLE` and ended with `\eTABLE`; each row starts with `\bTR` and ends with `\eTR`; and each cell of a row starts with `\bTD` and is closed with `\eTD`.

Default behaviour is tight cells, all frames on. For customizing we can use `\setupTABLE`:

```

\setupTABLE[row]
  [number,odd,even,first,last,each][...]
\setupTABLE[column]
  [number,odd,even,first,last,each][...]

```

Options are the same as in the `\framed` environment ([wiki.contextgarden.net/Framed](#)) and influence most aspects of how the table appears. So we have quite a number of `[(option)=(key)]` possibilities for setting up the frame (each side), style, color, rule thickness etc.

If we want a consistent table design throughout the document we might place these setups at the beginning of the document. If we then need a local adaptation we place the `\setupTABLE` command inside the `\bTABLE ... \eTABLE` construct. We can also attach to a given `\bTR` or `\bTD` options for this specific row or cell.

Cells can span multiple columns or rows, as in: `\bTD[nx=2]`, `\bTD[ny=3]`.

Natural tables can break over pages and we can define a table header for the first page and a table header for the following pages. It is also possible to define a table foot.

```
\bTABLEhead
  \bTR \bTD A \eTD \bTD B \eTD \bTD C \eTD \eTR
\cTABLEhead

\bTABLEnext
  \bTR \bTD X \eTD \bTD Y \eTD \bTD Z \eTD \eTR
\cTABLEnext

\bTABLEfoot
  \bTR[bottomframe=on] \eTR
\cTABLEfoot

\bTABLEbody
  \bTR \bTD 1 \bTD 10 \eTD \bTD 100 \eTD \eTR
  \bTR \bTD 1 \bTD 10 \eTD \bTD 100 \eTD \eTR
\cTABLEbody
```

7.4 Layers

In ConTeXt one can place content into specific locations. This is done with layers. The procedure is to define the layer, fill it and place it.

7.4.1 Defining a layer

```
\definelayer[Logo]
  [width=3cm, height=4cm]
```

The layer is by default attached to the left top corner of the typesetting area, if not stated otherwise.

7.4.2 Fill the layer

To fill a defined layer we use `\setlayer`, with the name of the layer as a first bracketed option, and *x* and *y* positioning dimensions relative to the anchor point as a second option. Finally, the actual content is given between braces (not brackets). This can be any content like figures, tabulations, tables ...

```
\setlayer[Logo]
  [x=-15mm,y=-35mm]
  {\externalfigure[cow][width=3cm]}
```

7.4.3 Placing the layer

The layer is placed with `\placelayer[<name>]`. A layer can only be placed (or flushed) if there is already content on the page. If the flushing is done before the page is started, the layer will not appear. If we have to place the layer as the first action we can use the command `\strut` or `\null`, which add the necessary anchor point.

```
\strut
\placelayer[Logo]
```

7.4.4 Dealing with floats

In TeX figures, pictures and tables often are floating objects, meaning that the typesetting system determines where such an object is to be placed.

ConTeXt supports the following image formats: JPEG, PNG, PDF, MetaPost; it will also honor EPS if Ghostscript is installed, as additional converters can kick in.

Normally pictures and other graphics are separate from the TeX sources, so we need to tell ConTeXt where to look for the required files:

```
\setupexternalfigures[location={local,default}]
```

In this case ConTeXt is looking for picture files locally and secondly in the TeX tree.

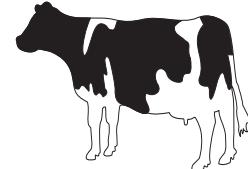
If we want ConTeXt to look for the files in a specific directory, we add the key `global` and give the path after the key `directory`:

```
\setupexternalfigures
  [location={local,default,global},
   directory={/MyDocs/Bachotex2017/tut/figures}]
```

7.4.5 Placing a figure

After the setup is done, we can insert a figure as follows, here to the right of the text:

```
\startplacefigure
  [title=Cow drawing.,
   location=right]
  {\externalfigure[cow]
   [height=.1\textheight]}
\stopplacefigure
```



To omit any caption and numbering we can add the keys `title=`, `number=`.

7.5 Fonts

In today's TeX environments, thanks especially to the great work by the Polish GUST e-foundry, we have an excellent collection of fonts available in modern font formats. The following are included in the ConTeXt standalone distribution: Latin Modern, the TeX Gyre collection, Antykwa Toruńska, Iwona, Kurier, and XITS.¹

Of course a great number of additional fonts are available, included in TeX Live and/or separately downloadable, such as: Gentium (from SIL International), DejaVu (based on an original design from Bitstream), ...

7.5.1 Playing with built-in fonts

If no font is specified, ConTeXt will use Latin Modern by default. The traditional way of specifying a font is as follows:

```
\setupbodyfont[pagella,rm,10pt]
```

¹ Khaled Hosny adapted STIX to TeX needs to make XITS.

Here we give the name of the font (family), followed by the required style and the size.

With the TeX Gyre fonts, we have a nice collection of fonts in the OpenType format, covering the so-called “base 35” PostScript fonts.

serif	rm	termes	(Times)
		pagella	(Palatino)
		schola	(New Century Schoolbook)
		bonum	(Bookman)
sans	ss	heros	(Helvetica)
		adventor	(Avant Garde)
mono	tt	cursor	(Courier)
calligraphic	cg	chorus	(Zapf Chancery)

In order to be able to use serif, sans serif, mono-space and math fonts mixed in one document, ConTeXt has predefined sets (families), which are called “typefaces”. E.g. the `pagella` option offers:

serif	pagella
sans	latin modern
mono	latin modern
math	pagella

7.5.2 Another way of using fonts

As long as you do not use commercial fonts, for which you have to set up the typescripts/font loading instructions yourself, you can easily use the predefined typefaces as shown above.

So especially for the use of fonts, i.e. system fonts or commercial fonts, there is a font selection mechanism (written by Wolfgang Schuster) which is part of the ConTeXt core. For a font that is not supported out of the box you can define a font family like this:

```
\definefontfamily [dejavu] [serif] [DejaVu Serif]
\definefontfamily [dejavu] [sans] [DejaVu Sans]
\definefontfamily [dejavu] [mono] [DejaVu Sans Mono]
\definefontfamily [dejavu] [math] [XITS Math]
[scale=1.1]

\definefontfamily [office] [serif] [Times New Roman]
\definefontfamily [office] [sans] [Arial]
[scale=0.9]
\definefontfamily [office] [mono] [Courier]
\definefontfamily [office] [math]
[TeX Gyre Termes Math]

\definefontfamily [linux] [serif] [Linux Libertine O]
\definefontfamily [linux] [sans] [Linux Biolinum O]
\definefontfamily [linux] [mono] [Latin Modern Mono]
\definefontfamily [linux] [math]
[TeX Gyre Pagella Math] [scale=0.9]

\definefontfamily [myfamily] [mono] [TeX Gyre Cursor]
[features=none]
```

When you want to combine fonts of which the design sizes are not directly compatible you can add

a scaling factor to the definitions, as shown above. [The line breaks above are for *TUGboat*’s narrow columns; normally such definitions are written all on one source line, although this is not required.]

7.5.3 Your own typescripts

The same approach is used for commercial fonts. Either you use Wolfgang’s core module as above, or you write your own typescripts, as briefly outlined here.

In any case, when adding a new font unknown to ConTeXt the file database must be renewed. The way to do this is to open a terminal and issue the command `context --generate`.

Suppose you have the font Seravek. The set of typescripts would read as follows. First the font’s filenames are mapped on a symbolic name inside a typescript, which takes two arguments: a category name, such as `sans` or `serif`, and a symbolic name.

```
\starttypescript [sans] [seravek]
\definefontsynonym [Seravek-Regular]
[file:Seravek-Regular] [features=default]
\definefontsynonym [Seravek-Bold]
[file:Seravek-Bold] [features=default]
\definefontsynonym [Seravek-Italic]
[file:Seravek-RegularItalic] [features=default]
\definefontsynonym [Seravek-BoldItalic]
[file:Seravek-BoldItalic] [features=default]
\stoptypescript
```

In a second step. the symbolic names of the font files are mapped to ConTeXt’s internal names. As with the first typescript, this typescript has the same category name and the symbolic name.

```
\starttypescript [sans] [seravek]
\definefontsynonym [Sans]
[Seravek-Regular] [features=default]
\definefontsynonym [SansItalic]
[Seravek-Italic] [features=default]
\definefontsynonym [SansBold]
[Seravek-Bold] [features=default]
\definefontsynonym [SansBoldItalic]
[Seravek-Bold-Italic] [features=default]
\definefontsynonym [SansCaps]
[Seravek-Regular] [features=smallcaps]
\definefontsynonym [SansBoldCaps]
[Seravek-Bold] [features=smallcaps]
\definefontsynonym [SansItalicCaps]
[Seravek-Italic] [features=smallcaps]
\definefontsynonym [SansBoldItalicCaps]
[Seravek-Bold-Italic] [features=smallcaps]
\stoptypescript
```

Now that the definitions are ready we create a third typescript, which defines the font family as a typeface. This typescript has a symbolic name with which we will use the fonts inside the document.

```
\starttypescript [Seravek]
\definetypeface[Seravek][ss][sans][seravek] [default]
\definetypeface[Seravek][mm][math][palatino] [default]
\stoptypescript
```

The three typescript definitions can be saved in a file with the name `type-imp-seravek.tex`. This file is best placed in the `TeX` tree, e.g. (in the standalone ConTeXt distribution):

```
.../tex/texmf-context/tex/context/user.
```

To use this typeface in the document the font is set up with:

```
\usetypescriptfile[type-seravek]
\usetypescript[seravek]
\setupbodyfont[Seravek,ss,10pt]
```

And now the default text font will be Seravek, in the sans serif style that we've defined.

8 Documentation

What has been shown so far is only a glimpse of what ConTeXt can do. Learning ConTeXt calls for careful study of the documentation. Many manuals written by Hans Hagen are included in the distribution; they can be found in the `TeX` tree, in (standalone ConTeXt) `texmf-context/doc/context/documents` or (`TeX` Live) `texmf-dist/doc/context`.

Also available is a large wiki containing very useful help, including command references with many explanations and examples. In addition, it has both shorter and longer articles contributed by many ConTeXt users. You can find it at wiki.contextgarden.net. For convenience, a summary reference of the main commands in this tutorial are also available at tug.org/TUGboat/tb38-3/tb120egger-cmds.pdf. (They were prepared using the built-in command `\showsetup`, e.g. `\showsetup [setuplayout]`.)

A large set of test files is also available. These are used in development, but also contain very useful information for self-study. On the page pragma-ade.com/download-1.htm you can find a link to them as `context/latest/cont-tst.7z`.

Last but not least there is a mailing list to which you are invited to join. On this list practical issues with the system and questions are posted. The address is `ntg-context@ntg.nl` (with archives, etc., at `ntg.nl/mailman/listinfo/ntg-context`)—don't be shy or afraid; also simple questions are answered promptly!

9 Acknowledgements

`TeX` is still alive after so many years. I would like to thank Hans Hagen cordially for his tremendous work given to us for free. With his system we are still competitive in the modern environment of typesetting from different sources, producing PDF from XML, linking to databases, exporting XML and other formats like ePub. The transition from pdf`TeX` to Lua`TeX` was and is supported by a number of people, notably Taco Hoekwater, Hans Hagen, Helmut

Henkel and Luigi Scarso. We owe them a big thank you! Another member of the community who spends a considerable amount of time and effort in supporting the system is Wolfgang Schuster. I would like to thank him for all that he does. Finally, behind the scenes a lot happens in order to make ConTeXt available for anybody from the ConTeXt garden. I would like to thank Mojca Miklavec for all the energy she puts into the preparation and testing of the ConTeXt distribution.

◊ Willi Egger
Maasstraat 2
5836 BB SAMBEEK
The Netherlands
w dot egger (at) boede dot nl

A Bringing the elements together: An invoice skeleton

The following text shows a basic approach to setting up an invoice. It starts with a preamble setting up the body font and other definitions. After `\starttext` the invoice is built. The result is a one-page document containing all elements for the invoice. It's true that it is not yet the most beautiful document in the world, but it is the base for tuning ...

```
% Setup font to be used
\setupbodyfont[pagella,rm,10pt]

% Setup path to find graphics
\setupexternalfigures[location={local,default}]

% Switch off automatic page numbering
\setuppagenumbering[location=]

% Setup page layout
\setuplayout
  [topspace=15mm,
   backspace=15mm,
   header=40mm,
   footer=2.2cm,
   height=middle,
   width=middle,
   leftmarginwidth=10mm,
   rightmarginwidth=14mm]

% Setup headers and footers
\startsetups[Header]
  \framed
    [height=\headerheight,
     align=lohi,
     frame=off,
     foregroundcolor=green,
     foregroundstyle=\bf]
    {Agricultural Services}
\stopsetups
%
\setupheadertexts[][\setups{Header}][][\setups{Header}]

\startsetups[Footer]
\switchtobodyfont[8pt]
\starttabulate[|p|p|p|p|]
```

```

\NC Bank account      \EQ P10123 23 3445 678      % Receiver address
\NC Address          \EQ Jackowskiego 12 m 3 \NC\NR
\NC Chamber of Commerce \EQ North-14 53 21
\NC VAT number        \NC 61-757 Pozna\n      \NC\NR
\NC Phone             \NC \unknown           \NC\NR
\NC e-Mail            \NC info at boede.nl    \NC\NR
\stoptabulate
\stopsetups
\setupfootertexts[] [\setups{Footer}] [] [\setups{Footer}]

% Setup logo layer
\definelayer
[Logo]
[width=3cm, height=4cm]

% Standard text included on each invoice
\startbuffer[Conditions]
All prices are in EUR unless stated otherwise.

\blank[small]

This invoice is payable within 14 days
after date of issuing.

\blank[small]

Our general terms and conditions apply to all quotes,
contracts and services unless stated otherwise.
The general terms and conditions are deposited
at the record office of the court of justice at
s'-Hertogenbosch, The Netherlands. On request
a copy in the Dutch language is available for free.

\stopbuffer

\startbuffer[Thanks]
Thank you for your order.
\stopbuffer

% for our example
\showframe
\showlayout

% Begin document
\starttext

% Logo
\setlayer
[Logo]
[x=-5mm,
y=-42mm]
{\externalfigure[cow][width=3cm]}

%
\setlayer
[Logo]
[x=148mm,
y=137mm]
{\externalfigure[mill][height=4cm]}

%
\placelayer[Logo]

\starttabulate{lllr}
\startlines
Name
Company
Street
Postal code Town
\stoplines
\blank[1cm]

\stoptabulate
\stopsetups
\setupfootertexts[] [\setups{Footer}] [] [\setups{Footer}]

% Invoice data
\starttabulate{lllr}
\startlines
\NC \bfc Invoice   \NC           \NC\NR
\NC Invoice date   \EQ 26-04-2017 \NC\NR
\NC Invoice number \EQ 01-2017   \NC\NR
\stoplines
\stoptabulate

\blank[2cm]

% Invoice content
\setupTABLE[each][each][frame=off]
\setupTABLE[r][1][style=bold]
\setupTABLE[c][1][width=2.5cm]
\setupTABLE[c][2][width=9cm]
\setupTABLE[c][3][width=2.5cm,align=flushright]
\setupTABLE[c][4][width=1.5cm,align=flushright]
\setupTABLE[c][5][width=2.5cm,align=flushright]

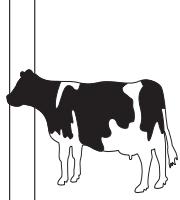
\bTABLE
\bTR[bottomframe=on]
\bTD Date \eTD\bTD Description \eTD
\bTD Quantity \eTD\bTD Price \eTD\bTD Amount \eTD
\eTR
\bTR
\bTD 26-04-2017 \eTD\bTD Delivery of goods \eTD
\bTD 10 \eTD\bTD 25.00 \eTD\bTD 250.00 \eTD
\eTR
\bTR[topframe=on,rulethickness=1.5pt]
\bTD \eTD\bTD \eTD\bTD Subtotal \eTD
\bTD \eTD\bTD 250.00 \eTD
\eTR
\bTR
\bTD \eTD\bTD \eTD\bTD VAT 20\% \eTD
\bTD \eTD\bTD 50.00 \eTD
\eTR
\bTR
\bTD \eTD\bTD \eTD\bTD Total \eTD\bTD \eTD
\bTD[bottomframe=on,rulethickness=1.5pt]
300.00\eTD
\eTR
\etABLE

\vfil

\getbuffer[Thanks]
\blank[2*big]
\getbuffer[Conditions]
\stoptext

```

The output is on the next page [scaled for *TUGboat*, so the absolute dimensions given in the source will only match proportionally; sorry].

	 <h2 style="font-size: 1.2em; font-weight: bold; margin: 0;">Agricultural Services</h2>																										
	<p>Name Company Street Postal code Town</p> <p>Invoice Invoice date : 26-04-2017 Invoice number : 01-2017</p>																										
	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left; padding: 2px;">Date</th> <th style="text-align: left; padding: 2px;">Description</th> <th style="text-align: right; padding: 2px;">Quantity</th> <th style="text-align: right; padding: 2px;">Price</th> <th style="text-align: right; padding: 2px;">Amount</th> </tr> </thead> <tbody> <tr> <td style="padding: 2px;">26-04-2017</td> <td style="padding: 2px;">Delivery of goods</td> <td style="text-align: right; padding: 2px;">10</td> <td style="text-align: right; padding: 2px;">25.00</td> <td style="text-align: right; padding: 2px;">250.00</td> </tr> <tr> <td></td> <td></td> <td style="text-align: right; padding: 2px;">Subtotal</td> <td style="text-align: right; padding: 2px;">250.00</td> <td></td> </tr> <tr> <td></td> <td></td> <td style="text-align: right; padding: 2px;">VAT 20%</td> <td style="text-align: right; padding: 2px;">50.00</td> <td></td> </tr> <tr> <td></td> <td></td> <td style="text-align: right; padding: 2px;">Total</td> <td style="text-align: right; padding: 2px;">300.00</td> <td></td> </tr> </tbody> </table>	Date	Description	Quantity	Price	Amount	26-04-2017	Delivery of goods	10	25.00	250.00			Subtotal	250.00				VAT 20%	50.00				Total	300.00		
Date	Description	Quantity	Price	Amount																							
26-04-2017	Delivery of goods	10	25.00	250.00																							
		Subtotal	250.00																								
		VAT 20%	50.00																								
		Total	300.00																								
	<p>Thank you for your order.</p> <p>All prices are in EUR unless stated otherwise. This invoice is payable within 14 days after date of issuing. Our general terms and conditions apply to all quotes, contracts and services unless stated otherwise. The general terms and conditions are deposited at the record office of the court of justice at s'-Hertogenbosch, The Netherlands. On request a copy in the dutch language is available for free.</p>																										
																											
	<p>Bank account : PL0123 23 3445 678 Chamber of Commerce : North-14 53 21 VAT number : PL12653007 Phone : ...</p> <p>Address : Jackowskiego 12 m 3 61-757 Poznań e-Mail : info at boede.nl</p>																										

Art Concret, Basic Design and meta-design

Marcel Herbst

Abstract

This note links Concrete Art (*l'art concret*), a visual art form originating around 1925, with Basic Design as taught in first-year design courses. It extends this view by approaching the topic from a rule-based, meta-design perspective using METAPOST and Nicola Vitacolonna's engine for TeXShop.

1 Introduction

Art academies of the 19th century had their introductory courses. The Bauhaus (1919–33), the famous design school located first in Weimar, then in Dessau, and eventually in Berlin when the Nazis forced it to close in 1933, introduced a first year introductory course that was to serve as a model for other design schools. This introductory course, the *Vorkurs* or *Grundlehre*, was emulated by schools around the globe in an attempt to initiate students to the basics—the fundamental principles—of design.

In post-World-War-II Germany, the idea came up to found anew a Bauhaus. The plan was formulated by (among others) Inge Scholl, the sister of Hans and Sophie Scholl; Hans and Sophie were, as members of the resistance group *Weisse Rose*, executed by the Nazis. Inge and her husband-to-be, Otl Aicher, a graphic designer, were involved in the post-war *Ulmer Volkshochschule* (Ulm Adult Education Center), and they initially pursued the notion to found a school in Ulm that was to focus on political education. When Max Bill (architect, designer and artist—and former student of the Bauhaus) was invited to join the planning team, the idea took hold to found a “new” Bauhaus. John McCloy, the U.S. High Commissioner in Germany at the time, supported the project, and eventually the *Hochschule für Gestaltung* (HfG) in Ulm opened its doors in 1953 (to be shut, in 1968, by a regressive government) [10].

The HfG had, like the Bauhaus, a formative, propædeutic year designed to initiate students to various design professions (product design, architecture, graphic design, et cetera). Part of the curriculum of this first year was a “basic design” studio, and Max Bill, the first rector of the HfG and an early advocate of Concrete Art, recruited various people of similar orientation to teach this course: Josef Albers (a Bauhaus émigré, former rector of Black Mountain College in Asheville, NC, and, after 1950, faculty member of Yale University and chair of its Department of Design—and, presumably, the person who had coined the term “basic design”),

Otl Aicher (subsequently the corporate identity designer of Lufthansa and the Munich Olympic Games), Friedrich Vordemberge-Gildewart, Johannes Itten, Hermann von Baravalle, Tómas Maldonado, and others. One who enrolled in this studio course was William S. Huff, an American architecture student and, later on, an associate of Louis Kahn who went to Germany on a Fulbright grant to attend the HfG for the 1956/57 academic year. The course which Huff took was taught by Maldonado and, upon returning to the U.S., Huff proceeded to teach this course (even part-time at the HfG during the period of 1962–66) until his retirement (mainly as a member of the architecture faculty of the Carnegie Institute of Technology/Carnegie Mellon University [1960–72] and the State University of New York at Buffalo [1974–98]).

As a student of the HfG myself (1958–62), I had heard of Huff, and had seen some of the works of his students, but I had never met him in Ulm. I came to realize the significance of Huff's work later on, when I read Douglas Hofstadter's essay on Huff [7]. Around 2009 I learned that Huff had handed over the various designs of his students to the archive of HfG, and in the winter of 2013/14 the Ulm Museum curated a show based on his legacy: *Basic Design — Von Ulm in die USA und zurück*. Perhaps two years earlier I had established mail contact, and in 2015 I visited William Huff in Pittsburgh, where he lives.

Huff creatively extended the foci of Basic Design as taught in Ulm, and he implicitly shifted the limiting aesthetic boundaries of Concrete Art. Basic Design, as seen by Huff, has a strong rule-based orientation, and because of this feature it may be approached via meta-design [4] and can be programmed. Huff's students did not use the computer to create their designs, but I found this feature attractive and, hence, started to program designs (using METAPOST, and with the help of our son, Joshua Aaron) which we had been playing with as students, or patterns Huff's students had come up with.

2 Concrete Art and Basic Design

Concrete Art evolved as an offshoot of several versions of abstract art such as Constructivism (with Kasimir Malevitch or El Lissitzky as exponents) and De Stijl (with Theo van Doesburg, Piet Mondrian or Georges Vantongerloo as members). A first manifesto of De Stijl, dating from 1918, lists nine points, none of which bear a direct relation to stylistic elements of design: it is more a call for participation and a statement regarding social conditions.

Concrete Art, on the other hand, with its intersection of membership, is more specific than the

manifesto of De Stijl. The term *art concret* is said to have been coined by Theo van Doesburg, and in a new manifesto, published 1930 [1], six principles were emphasized, four of which (namely the second to the fifth) have direct bearing on the style (and on the particular approach):

- the art work has to be completely pre-conceived (*L'œuvre d'art doit être entièrement conçue et formée avant son exécution*);
- the art work must be constructed with elements which only represent themselves ([...] doit être [...] construit avec des éléments purement plastiques [...] Un élément pictural [et le tableau] n'a pas d'autre signification que 'lui-même');
- the picture and its elements ought to be simple;
- the technical execution [of the picture] ought to be exact, anti-impressionistic.

This manifesto is, implicitly, a rejection of abstraction (as practiced, for instance, by Piet Mondrian). It is to be seen in the context of a new focus on machines and technology, shared by a range of artistic movements (including Futurism), and in relation to a culture which embraced architecture, product design, graphic design, typography, and art, in the sense of a *Gesamtkunstwerk* (synthesis of the arts). This *Neue Sachlichkeit* (new objectivity, new simplicity, functionalism), part also of the Bauhaus, was not restricted only to the visual art and design but affected philosophy as well [5].

After 1933, Concrete Art gained momentum where it could (with people like Hans Arp, Sophie Taeuber-Arp, Friedrich Vordemberge-Gildewart, Richard Paul Lohse, Max Bill, Verena Loewensberg). Many of the works of these artists are not conceived to follow rules, but some are, particularly those of Lohse and Bill, and the ones that are rule-based can be programmed; even those that do not follow specific rules may be “recreated” (or simulated) using random variables.

Many of the rule-based works created by concrete artists are also problem-based and, hence, can easily be used in the context of assignments in a Basic Design course. The advantage of programmed design is obvious: in the old days, students were given at least one month to come up with a particular design; today, program generation can proceed much faster, and the designer (programmer) can play with parameters which may produce unexpected—unforeseen—results which cannot easily be pre-visualized. In this way, one can work like Jackson Pollock (an exponent of abstract expressionism), that is, interactively. Pollock let paint drip onto the canvas to inspect the intermediate result, and proceeded this

way until he was satisfied with the result. In similar fashion, the programmer-designer can interactively play with his program by adjusting parameters, until the design is satisfactory. Because turn-around times of design production become greatly reduced, as we all know (and as font designers working with METAFONT know), design itself, and the didactic approach to design education, can be seen in a new light.

Lastly, I should also mention that Basic Design, approached from a meta-design angle, is not simply computer generated art. Computer generated art (making use, for instance, of fractals) normally lacks the historical link that I have sketched above: it does not follow the ascetic aesthetic of Concrete Art, it is frequently rather baroque (or even tacky) in appearance, and often lacks the implicit link to applications (such as architecture or design).

3 Meta-design

Meta-design, so natural to TeXnicians or users of TeX-related programs, is not frequently used in art or design schools (speculation on why this is the case would require another note). But meta-design was somehow anticipated at the HfG in Ulm, in that topics of operations research were part of the general curriculum (taught by Horst Rittel who subsequently moved to Berkeley). We were introduced to graph theory on the basis of Dénes König [9], and later I acquired the text of Claude Berge [2]. I became conscious of the four-color problem, of the problem on subdivision of a square with squares (of unequal size), and the famous problem of Königsberg which Leonhard Euler had formulated. For architecture students, graph theory was seen as a natural ally because geographic maps or floor plans could be translated, uniquely, into graphs; but the converse, so important in practical applications, was not that easy to find—that is, given a graph (of some relations), how to produce an associated floor plan.

Meta-design of Basic Design patterns, as I mentioned in the Introduction, is a new avocation of mine, spawned by old age (e.g. instead of solving crossword puzzles). I am a lover of Concrete Art, charmed by its frugal aesthetic, and I was taken aback by its seeming stagnation during the past half century. When I realized, after being exposed to Huff's design on the basis of the note written by Hofstadter, that there is a lot of life left in Concrete Art, and after I had decided to use METAPOST to again occupy myself with Basic Design, I spent occasional sessions on this theme, along with my usual work in the field of higher education management (or the writing of essays on cultural matters).

Donald Knuth's book on METAFONT [8] I have had on my shelves for some time, but I cannot claim to have studied it: I had no reason, no motive, to absorb it and was concerned with other tasks. However, I had bought and perused the tome because I wanted to know what it is about. I also acquired, early on, a technical report by Neenie Billawala [3] where she introduces her Pandora font and demonstrates visually the inherent power of METAFONT to generate variations (of fonts or patterns).

Basic Design, in its meta-design version, could not only be used in the propædeutic courses of design schools; it could also be used in high schools (in a cross-disciplinary way spanning mathematics/programming and art education).

4 Examples of Concrete Art

A few examples of Concrete Art should be introduced here to show the uninitiated what Concrete Art is about (and then I shall proceed to examples of Basic Design); I will show three designs. The first is a rendition of Max Bill that he composed for an exhibition poster (Fig. 1). It shows the ingredients of the classic Concrete Art: simple shapes, and basic colors. The second is a composition of Richard Paul Lohse, using rotation and progression as design principles (shown in black and white here; see Fig. 2). And the third is a special one, by Josef Albers, a commentary on perspective and architectural isometric drawings (Fig. 3). Albers had this one engraved in black Formica in the wood shop of the *Hochschule für Gestaltung* that was run by Paul Hildinger, and it is now in the possession of Hildinger's son Peter. There are a range of such engravings on the market.

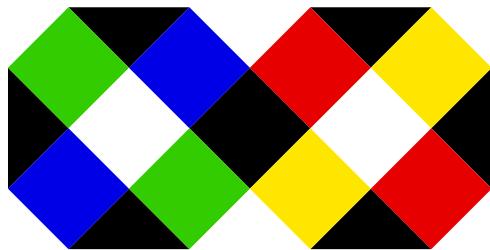


Figure 1: Max Bill, composition for a poster (1967)

5 Examples of Basic Design

Basic Design, because of its didactic focus and because it is (generally) rule-based, is naturally more structured than Concrete Art. Its focus suggests that Basic Design starts out with "assignments" and deals with various problems that affect perception, i.e. visual comprehension: foreground-background, color equivalences, moiré, symmetry (various), repetitions,

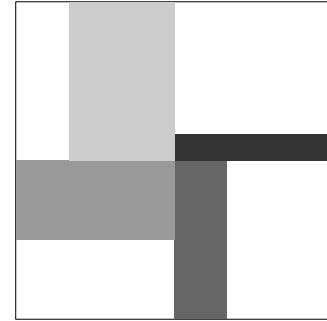


Figure 2: Richard Paul Lohse, "Bewegungen um ein Zentrum" (1982)

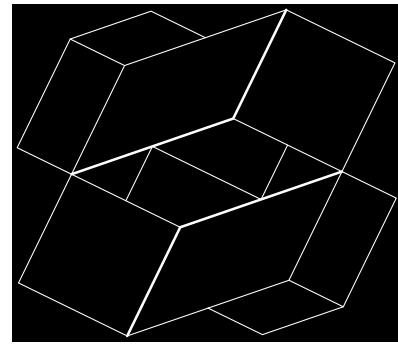


Figure 3: Josef Albers, "Structural Constellation" (circa 1955)

rotations, transformations, et cetera, or combinations thereof.

One of the first patterns that I had created was a simple dot pattern (Fig. 4). The code is the following:

```
beginfig(1)
picture dotimage;
dotimage := image(
u:=15pt;
for i=1 upto 18:
  for j=1 upto 12:
    pickup pencircle scaled ((-1**i+j)+1);
    draw fullcircle scaled (0.8*u) shifted
      ((i*u)+((-1**j)),(j*u)+((-1**i)));
    draw fullcircle scaled (0.8*u) shifted
      ((i*u)+((-1**j)),(j*u)+((-1**i)));
  endfor;
endfor;
);
draw dotimage scaled 1;
endfig;
end
```

On the basis of that first dot pattern I created others (e.g. Fig. 5).

I'll now show a few patterns which were created during the various design studios of Basic Design,

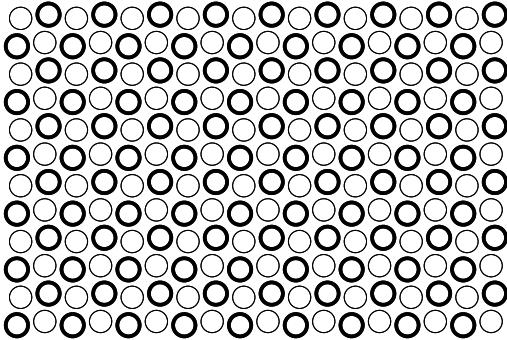


Figure 4: Simple dot pattern

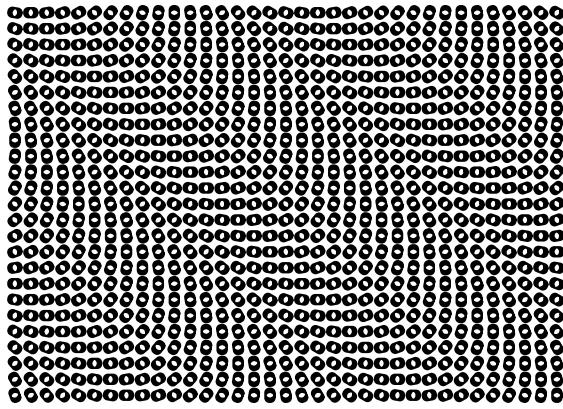


Figure 5: Dot pattern

e.g. the well-known spiral depicted in Fig. 6; or one of the mathematical curves that Tómas Maldonado (at the HfG) loved to use in his assignments (see Fig. 7—I show it here in a color version). From Huff's studio I present two drawings: a refined simple design of two linked squares (Fig. 8), and one of the transformations which Hofstadter had liked so much (Fig. 9) and which prompted me to remark as above, after seeing these designs, “that there is a lot of life left in Concrete Art”.

Finally, I shall include a random pattern designed to emulate the designs of Hans Arp (Fig. 10). The code for that picture is the following:

```
beginfig(1);
picture dotimage;
dotimage := image(
u:=24pt;
pair p,q,r,s,a,b,c,d;
for i = 1 upto 1:
p := (uniformdeviate 200,
uniformdeviate -200);
q := (uniformdeviate 200,
uniformdeviate -200);
r := (uniformdeviate 200,
uniformdeviate -200);
```

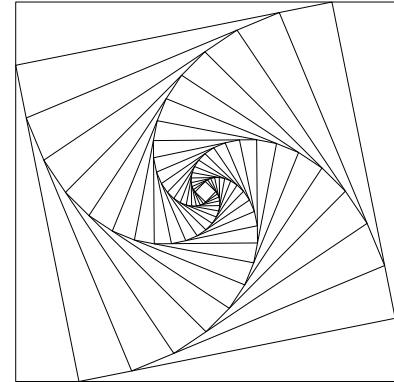


Figure 6: Transformation

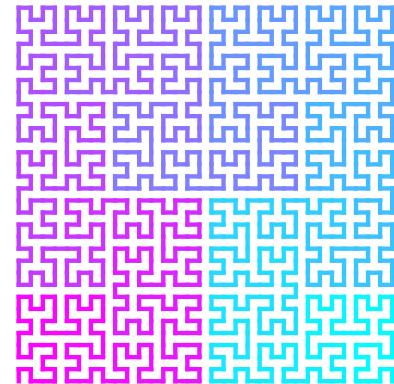
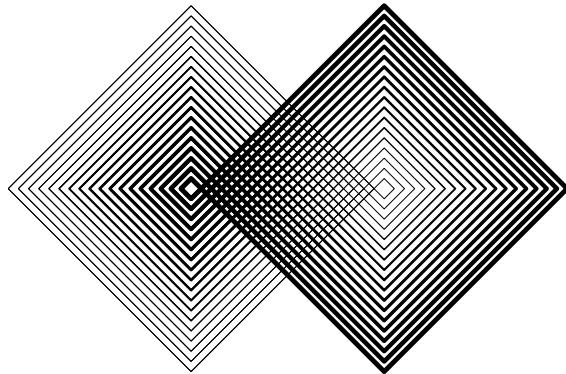
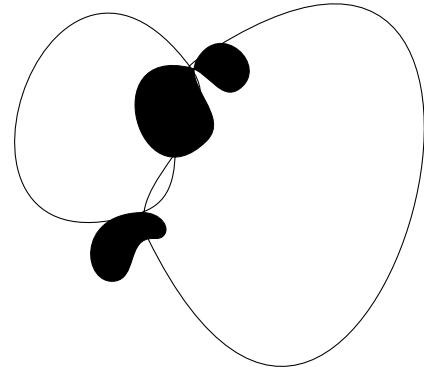
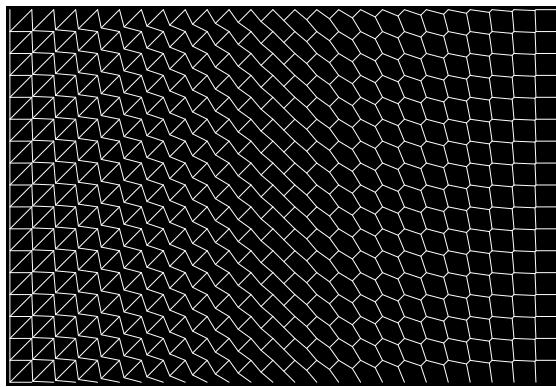


Figure 7: Hilbert curve (programmed by [6], with the exception of the color gradient)

```
s := (uniformdeviate 200,
uniformdeviate -200);
a := (uniformdeviate 300,
uniformdeviate 300);
b := (uniformdeviate 300,
uniformdeviate 300);
c := (uniformdeviate 300,
uniformdeviate 300);
d := (uniformdeviate 300,
uniformdeviate 300);

fill p..q..r..s..cycle withpen pencircle;
fill a..b..c..d..cycle withpen pencircle;
draw a..q..c..s..cycle withpen pencircle;
endfor;
);
draw dotimage scaled 1;
endfig;
end
```

**Figure 8:** Linked squares**Figure 10:** After Hans Arp**Figure 9:** Transformation

References

- [1] Art Concret. Numéro d'introduction. Paris, April 1930.
https://monoskop.org/Art_concret
- [2] Claude Berge. *Théorie des graphes et des applications*. Dunod, 1958.
- [3] Neenie Billawala. Metamarks: Preliminary studies for a Pandora's Box of Shapes. Technical Report STAN-CS-89-1256, Computer Science Department, Stanford University, 1989.
- [4] Frederick P. Brooks. *The Design of Design: Essay from a Computer Scientist*. Addison-Wesley, 2010.
- [5] Rudolf Carnap. Die Überwindung der Metaphysik durch logische Analyse. *Erkenntnis*, 2:219–241, 1931.
- [6] J.G. Griffiths. Table-driven algorithms for generating space-filling curves. *Computer-Aided Design*, 17(1):37–41, January–February 1985.
- [7] Douglas R. Hofstadter. Parquet Deformations: A Subtle, Intricate Art Form. In *Metamathematical Themas: Questing for the Essence of Mind and Pattern*, chapter 10, pages 191–212. Basic Books, 1985.
- [8] Donald E. Knuth. *The METAFONTbook*, volume C of *Computers & Typesetting*. Addison-Wesley, 1986.
- [9] Dénes König. *Theorie der endlichen und unendlichen Graphen: kombinatorische Topologie der Streckenkomplexe*. Chelsea Publishing Company, 1950.
- [10] René Spitz. *The Ulm School of Design: A View Behind the Foreground*. Axel Menges, 2002.

◊ Marcel Herbst
 Ostbühlstrasse 55
 CH-8038 Zürich
 Switzerland
 herbst (at) 4mat dot ch

The current state of the PSTricks project, part II

Herbert Voß

Abstract

PSTricks is an abbreviation for PostScript Tricks, using the enormous graphical capabilities of the programming language PostScript, old as it may be. It is a so-called page description language, created in 1984 by Adobe Systems. In [4] we gave a report of what was possible with the different packages at that time. With this article we show what's new in the last seven years.

1 From PSTricks to PDF

The traditional route to create a PDF document with PostScript specials is still `latex → dvips → ps2pdf`. This sequence of commands can be put into a script or defined as a build-command for a GUI to make it only one mouse click. However, if one wants to use `pdflatex` or `XeLaTeX` instead, this can also be done.

1.1 Using pdflatex

The package `auto-pst-pdf`¹ from Will Robertson works in the same way as `pst-pdf`², but doesn't need a script or the usual four runs by the user. Everything is done in one `pdflatex` run; this requires the `shell-escape` option to be enabled, to allow the running of external programs from within `pdflatex`. MiKTeX users have to enable the option `enable-write18`. This `shell-escape` option can be enabled in any GUI, for example TeXstudio:

LaTeX	<code>latex --shell-escape -src -interaction=nonstopmode %.tex</code>
PdfLaTeX	<code>atex -synctex=1 --shell-escape -interaction=nonstopmode %.tex</code>
XeLaTeX	<code>atex -synctex=1 -shell-escape -interaction=nonstopmode %.tex</code>
LuaLaTeX	<code>lualatex --shell-escape -interaction=nonstopmode %.tex</code>

It is available from the GUI panel via Options → Configure TeXstudio → Commands.

`auto-pst-pdf` converts all `pspicture` environments into single images which replace the environment on the fly, in a second run. If there is no `pspicture` environment then the PSTricks-related code must be enclosed in a `postscript` environment.



For example: the poker card internally uses the `pspicture` environment. With the `postscript` environment it will be converted by `auto-pst-pdf`, otherwise it will be missing in the PDF output. Here is the code for the above:

```
... For example: the poker card
\begin{postscript}
\psset{unit=0.5}\crdAs
\end{postscript}
internally uses ...
```

The `postscript` environment can be used as its own paragraph or within a line.

1.2 Using XeLaTeX

XeLaTeX always creates an `.xdv` (extended DVI) file, which then is automatically converted into a PDF document. However, there are some cases where the program `xdvipdfmx` cannot create the correct PDF, e.g. nearly all examples from the old package `pst-light3d`.

1.3 LuaLaTeX

LuaLaTeX creates PDF directly, like pdfTeX. Therefore the first LuaLaTeX run needs to specify the option `--output-format=dvi`, which is not handled by the package `auto-pst-pdf`. However, using the package `dtk-extern` from <https://ctan.org/pkg/dtk> one can create any kind of TeX document inside a LuaLaTeX document.

2 Old packages with new macros

2.1 pst-barcode

This package has existed for a long time, but now supports dozens of additional barcodes. Please refer to the documentation for the complete list. Here's an example of usage:



```
\begin{pspicture}(2in,0.5in)
\psbarcode{(00)030123456789012340|(02)130
12345678909(37)24(10)1234567ABCDEFG}%
{ccversion=c}{gs1-128composite}
\end{pspicture}
```

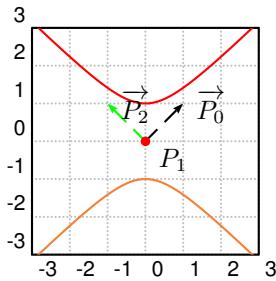
2.2 pst-bezier — Bézier curve with weighted points

A mass point is a weighted point $(P; \omega)$ with $\omega \neq 0$, or a vector $(\vec{P}; 0)$ with a weight equal to 0. A generic mass point is noted $(P; \omega)$. The package `pst-bezier` has a new macro `\psRQBCmasse`, which allows drawing a Bézier curve with such weighted points.

¹ ctan.org/pkg/auto-pst-pdf

² ctan.org/pkg/pst-pdf

Conic	Three weighted points	Points and vectors
Parabola	$(P_0; 1), (P_1; \omega) (P_2; \omega^2)$	$(P_0; 1), (\vec{P}_1; 0) (\vec{P}_2; 0)$
Ellipse	$(P_0; 1), (P_1; \omega_1), (P_2; \omega_2), \omega_2 > \omega_1^2$	$(P_0; 1), (\vec{P}_1; 0) (P_2; 1)$
Hyperbola	$(P_0; 1), (P_1; \omega_1) (P_2; \omega_2), \omega_2 < \omega_1^2$	$(P_0; 1), (\vec{P}_1; 0) (P_2; -1)$ $(\vec{P}_0; 0), (P_1; 1) \text{ and } (\vec{P}_2; 0)$



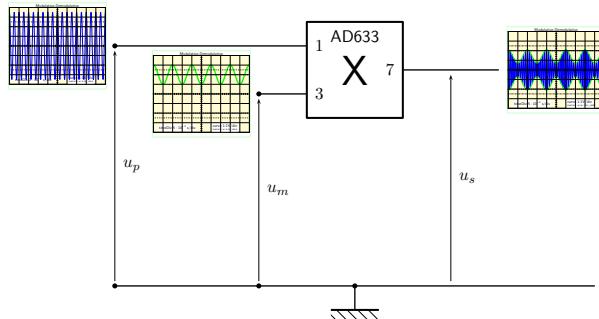
```
\begin{pspicture}[showgrid](-3,-3.4)(3,3)
\psclip{\psframe(-3,-3)(3,3)}
\psRQBCmasse[linecolor=red,
    autoTrace](1,1)(0,0)(-1,1){0,1,0}
\put[r]{(P0)}{$\overrightarrow{P_0}$}
\put[r]{(P1)}{$\overrightarrow{P_1}$}
\put[r]{(P2)}{$\overrightarrow{P_2}$}
\psRQBCmasse[linecolor=orange,
    autoTrace=false](1,1)(0,0)(-1,1){0,-1,0}
\endpsclip
\end{pspicture}
```

3 The new packages

3.1 pst-am

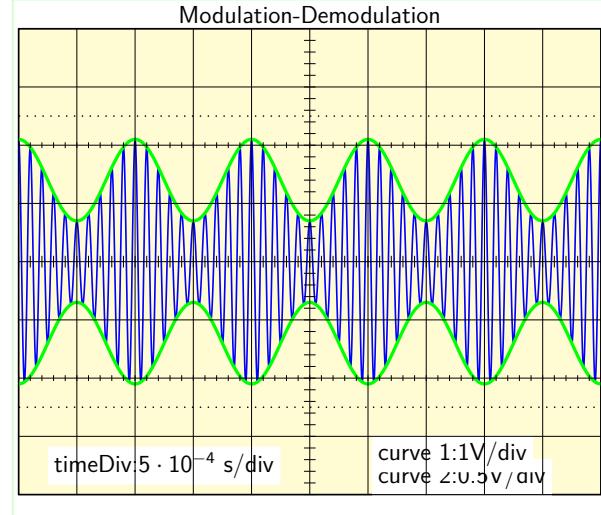
pst-am allows the simulation of modulated and demodulated amplitude of radio waves. You can choose several possible parameters and plot the following curves:

- modulated signals;
- wave carrier;
- signal modulation;
- signal recovering;
- signal demodulation.



```
\def\noeud{\qdisk{#1}{1.5pt}}
\begin{pspicture}(-5,-1)(5,7)
\psline(-5,0)(5,0)\psline(-5,5)(-1,5)
\psline(-2,4)(-1,4)
\pnode(-5,5){E2}\noeud(E2)
```

```
\pnode(-2,4){E1}\noeud(E1)
\psline[arrowinset=0,arrowscale=2](1,4.5)(3,4.5)
\psframe[linewidth=1.5\pslinewidth](-1,3.5)(1,5.5)
\rput(0,4.5){\Huge\sffamily X}
\put[270]{(0,5.5)}{\sffamily AD633}
\pnode(-5,0){M1}\pnode(-2,0){M2}
\pnode(0,0){O}\noeud(O)\noeud(M1)\noeud(M2)
\rput(0){\masse}
\put[0]{(-1,5){1}}
\put[0]{(-1,4){3}}
\put[180]{(1,4.5){7}}
\psset{linewidth=0.5\pslinewidth}
\psline{->}(-5,0.1)(-5,4.9)
\put[0]{(-5,2.5){$u_p$}}
\psline{->}(-2,0.1)(-2,3.9)
\put[0]{(-2,2){$u_m$}}
\psline{->}(2,0.1)(2,4.4)
\put[0]{(2,2.25){$u_s$}}
\psset[pst-am]{values=false}
\put[0]{(3,4.5){\psscalebox{0.2}{\psAM[SignalModule,
    enveloppe,frequencePorteuse=1e4,
    voltDivY2=0.5,timeDiv=5e-4,linewidth=2\pslinewidth]}}}
\put[1]{(-2,4){\psscalebox{0.2}{\psAM[SignalModulant,
    timeDiv=5e-4,linewidth=5\pslinewidth]}}}
\put[1]{(-5,5){\psscalebox{0.2}{\psAM[SignalPorteuse,
    timeDiv=2e-4, frequencePorteuse=1e4,
    linewidth=5\pslinewidth]}}}
\end{pspicture}
```



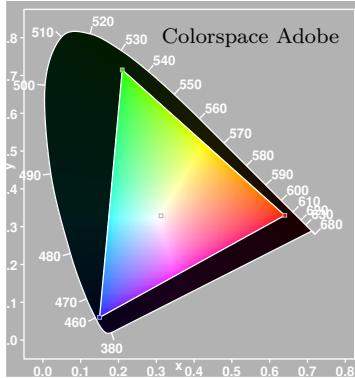
```
\psAM[SignalModule,enveloppe,frequencePorteuse=1e4,
voltDivY2=0.5,timeDiv=5e-4]
```

3.2 pst-cie

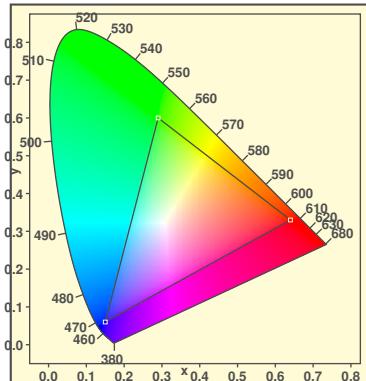
Using data (CIE XYZ 1931 and 1964) from the International Commission on Illumination (Commission internationale de l'éclairage) the package **pst-cie** proposes to represent the color table and/or the chromaticity diagram for different color spaces. The color spaces available are: **Adobe**, **CIE**, **ColorMatch**, **NTSC**, **Pal-Secam**, **ProPhoto**, **SMPTE** and **sRGB**.

It provides just one macro, which supports several optional arguments:

```
\psChromaticityDiagram[<options>]
```



```
\begin{pspicture}(-1,-1)(8.5,9.5)
\psChromaticityDiagram[datas=CIE1964,
  ColorSpace=Adobe,contrast=0.1,
  bgcolor=black!30]
\rput(5.5,8){\footnotesize Colorspace Adobe}
\end{pspicture}
```



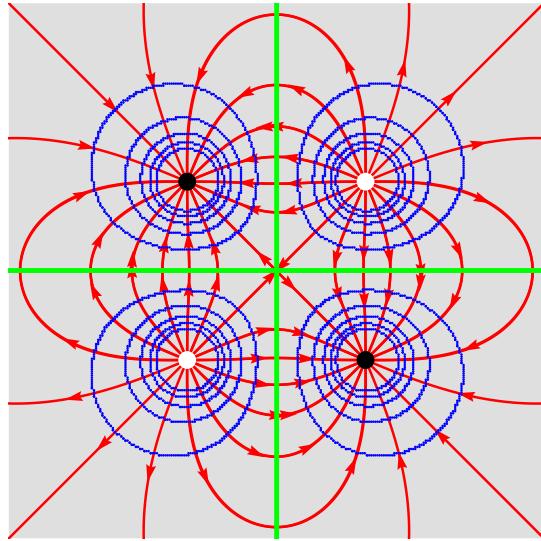
```
\begin{pspicture}(-1,-1)(8.5,9.5)
\psChromaticityDiagram[ColorSpace=Pal-Secam,
  bgcolor=yellow!100!black!20,
  textcolor=black!70]
\end{pspicture}
```

3.3 pst-electricfield

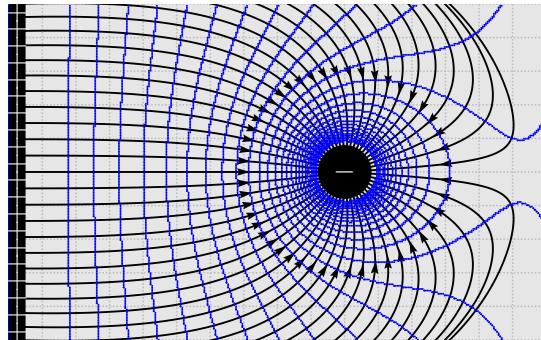
Equipotential surfaces and electric field lines can be drawn by using the package `pst-func` and the command `\psplotImp[options](x1,y1)(x2,y2)`. The Gauss theorem states that the electric flux across a closed surface S , defined by

$$\psi = \iint_S \vec{D} \cdot \vec{u}_n dS = Q \quad (1)$$

is equal to the real charge Q inside S . As a consequence, in places with no charge ($Q = 0$), the electric flux is a conserved quantity.



```
\begin{pspicture*}(-6,-6)(6,6)
\psframe*[linecolor=lightgray!50](-6,-6)(6,6)
\psgrid[subgriddiv=0,gridcolor=gray,griddots=10]
\psElectricField[Q={[-1 -2 2][1 2 2]
  [-1 2 -2][1 -2 -2]},linecolor=red]
\psEquipotential[Q={[-1 -2 2][1 2 2]
  [-1 2 -2][1 -2 -2]},linecolor=blue]
\psEquipotential[Q={[-1 -2 2][1 2 2]
  [-1 2 -2][1 -2 -2]},linecolor=green,
  linewidth=2\pslinewidth,
  Vmax=0,Vmin=0]
\end{pspicture*}
```



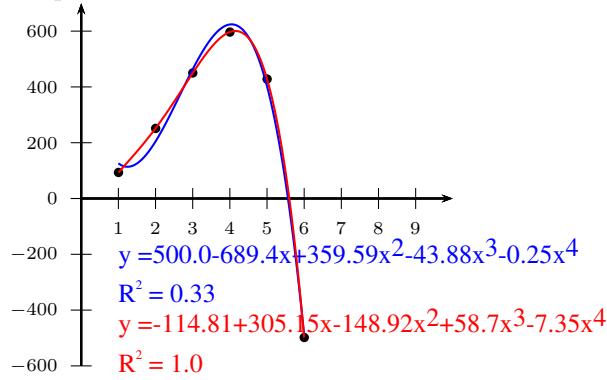
```
\begin{pspicture*}(-10,-5)(6,5)
\psframe*[linecolor=lightgray!40](-10,-5)(6,5)
\psgrid[subgriddiv=0,gridcolor=lightgray,griddots=10]
\psElectricField[
  Q={[600 -60 0 false][-4 0 0]},
  N=50,points=500,runit=0.8]
\psEquipotential[
  Q={[600 -60 0 false][-4 0 0]},
  linecolor=blue,Vmax=100,Vmin=50,
  stepV=2](-10,-5)(6,5)
\psframe*(-10,-5)(-9.5,5)
\rput(0,0){\textcolor{white}{\large$-$}}
\multido{\rA=4.75+-0.5}{20}{%
  \rput(-9.75,\rA){\textcolor{white}{\large$+$}}}
\end{pspicture*}
```

3.4 `pst-fit`

Curve fitting is the process of constructing a curve, or mathematical function, that has the best fit to a series of data points, possibly subject to constraints. The package `pst-fit` has many optional arguments to help achieve the desired interpolated curve. The following example shows six points of the polynomial

$$-109 + 294.53x - 142.94x^2 + 57.4x^3 - 7.26x^4$$

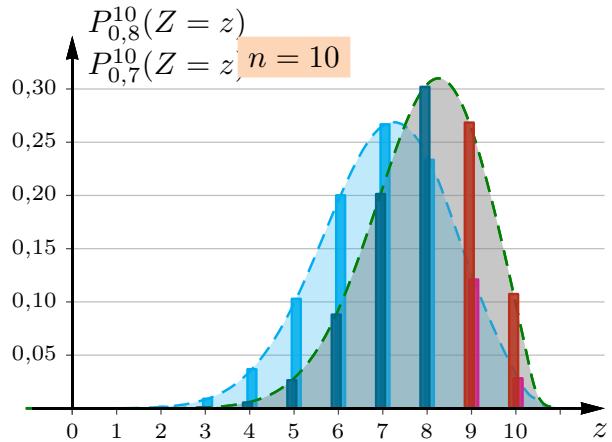
which are marked in the example as dots. The red and blue lines are two different solutions for a polynomial of 4th order. The internally calculated equations are plotted.



```
%Poly(-7.26*x^4+57.4*x^3-142.94*x^2+294.53*x-109)
\def\poly{1 93 2 251 3 450 4 597 5 428 6 -498}
\begin{pspicture}(-0.5,-6.5)(13,7.5)
\psset{yunit=0.0075}
\psaxes[arrows=->,Dx=1,Dy=200,
    labelFontSize=\scriptstyle,
    xsubticks=1,ysubticks=1](0,0)(0,-600)(10,700)
\listplot[plotstyle=dots]{\poly}
\listplot[valuewidth=20,
    decimals=2,EqPos=1 -200,
    plotstyle=GLSR,PolyOrder=4,
    plotpoints=400,Yint=500,
    linecolor=blue]{\poly}
\listplot[linecolor=red,
    decimals=2,EqPos=1 -400,
    plotstyle=GLSR,PolyOrder=4,
    plotpoints=400]{\poly}
\end{pspicture}
```

3.5 `pst-func`

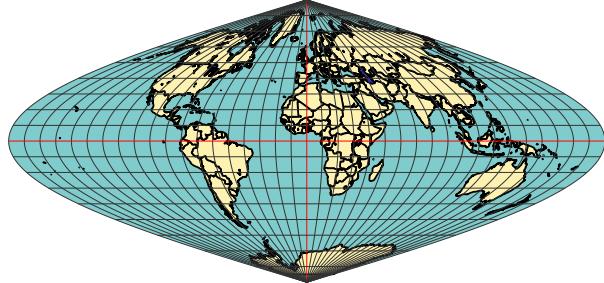
This package has some new macros for distributions, especially binomial distributions.



```
\begin{pspicture}(-.75,-1.8)(13.2,4.7)%
\psset{yunit=12cm}%
\psset{plotpoints=500,arrowscale=1.3,
    arrowinset=0.05,arrowlength=1.9,comma}
\psaxes[labelFontSize=\scriptstyle,xticksize=0 0,
    yticksize=0 12,tickcolor=gray!50,Dy=0.05,dy=0.05
]{->}(0,0)(-0.9,0)(10.8,0.34)
\uput[-90](11.9,0){$z$}
\uput[0](0,0.36){$P_{0,8}(Z=z)$}
\uput[0](0,0.32){$P_{0,7}(Z=z)$}
\rput(-0.05,0){%
\psBinomialC[linecolor=Green,
    fillstyle=solid,fillcolor=gray,opacity=0.25,
    plotstyle=curve,linestyle=dashed]{10}{0.8}}
\rput(0.05,0){\psBinomialC[linecolor=cyan,
    fillstyle=solid,fillcolor=cyan,opacity=0.25,
    plotstyle=curve,linestyle=dashed]{10}{0.7}}
\psBinomial[markZeros,linecolor=cyan,
    fillstyle=solid,fillcolor=cyan,barwidth=0.2,
    opacity=0.85]{0,8,10}{0.7}
\psBinomial[markZeros,linecolor=magenta,
    fillstyle=solid,fillcolor=magenta,barwidth=0.2,
    opacity=0.85]{9,10,10}{0.7}
\rput(-0.05,0){%
\psBinomialC[linecolor=Green,fillstyle=solid,
    fillcolor=gray,opacity=0.25,plotstyle=curve,
    linestyle=dashed]{10}{0.8}}
\psBinomial[markZeros,linecolor=DeepSkyBlue4,
    fillstyle=solid,fillcolor=DeepSkyBlue4,
    barwidth=0.2,opacity=0.85]{0,8,10}{0.8}
\psBinomial[markZeros,linecolor=BrickRed,
    fillstyle=solid,fillcolor=BrickRed,barwidth=0.2,
    opacity=0.85]{9,10,10}{0.8}
\psaxes[labels=none,xticksize=-2pt 0,yticksize=-2pt 0,
    tickcolor=black!70,Dy=0.05,dy=0.05\psyunit,Dx=1,
    dx=1\psxunit]{->}(0,0)(-0.9,0)(12,0.35)
\rput(5,0.33){\psframebox[fillstyle=solid,
    fillcolor=orange!30,
    linestyle=none]{$n=10$}}
\end{pspicture}
```

3.6 pst-geo

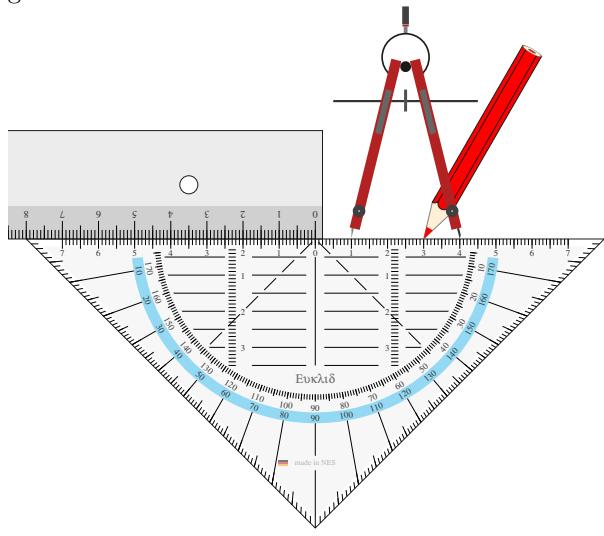
In the past the user had to load four different packages for the different geographical macros. After rearranging the code, there is now only one package: **pst-geo** with only one .sty file and one .pro file (PostScript code). The Sanson-Flamsted projection of the world is a *sinusoidal* projection which is a pseudo-cylindrical equal area-map:



```
\begin{pspicture}(-5,-5)(8,5)
\WorldMap[type=4] % Sanson-Flamsted
\end{pspicture}
```

3.7 pst-geometrictools

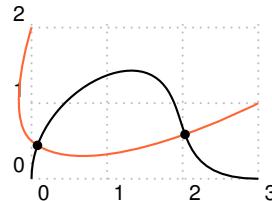
For mathematical worksheets in schools this package provides four macros for the tools which are used for geometrical constructions.



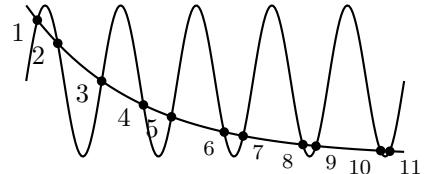
```
\begin{pspicture*}(-17,-17)(17,17)
\psProtractor{0}(0,0)% origin of the protractor
\psRuler{0}(0,0)% origin of the ruler
\psPencil{-30}(6,0)% origin of the pencil
\psCompass{3}(2,0)% origin of the compass
\end{pspicture*}
```

3.8 pst-intersec

This package calculates the intersection points of Bézier curves and/or arbitrary PostScript paths.



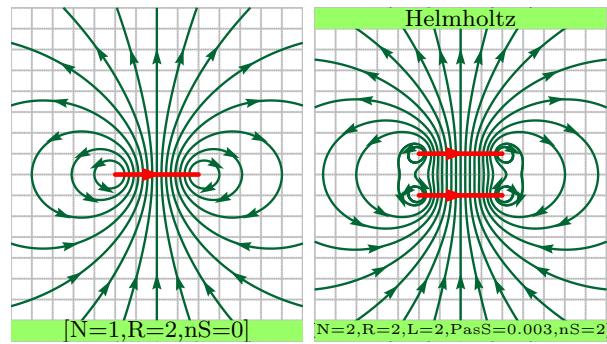
```
\begin{pspicture}(3,2)
\pssavepath[linecolor=DOrange]{MyPath}{\pscurve(0,2)(0,0.5)(3,1)}
\pssavebezier{MyBez}{(0,0)(0,1)(1,2)(3,2)(1,0)(3,0)}
\psintersect[showpoints]{MyPath}{MyBez}
\end{pspicture}
```



```
\begin{pspicture}(10,4.4)
\pssavepath{A}{%
\psplot[plotpoints=200]{0}{10}{x 180 mul sin 1 add 2 mul}}
\pssavepath{B}{%
\psplot[plotpoints=50]{0}{10}{2 x neg 0.5 mul exp 4 mul}}
\psintersect[name=C, showpoints]{A}{B}
\multido{\i=1+1}{5}{\uput[210](C\i){\i}}
\multido{\i=6+2,\ii=7+2}{3}{%
\uput[225](C\i){\footnotesize\i}
\uput[-45](C\ii){\footnotesize\ii}}
\end{pspicture}
```

3.9 pst-magneticfield

This package is similar to **pst-electricfield**. It supports the default two-dimensional magnetic field, density plots and a three-dimensional view of the field.

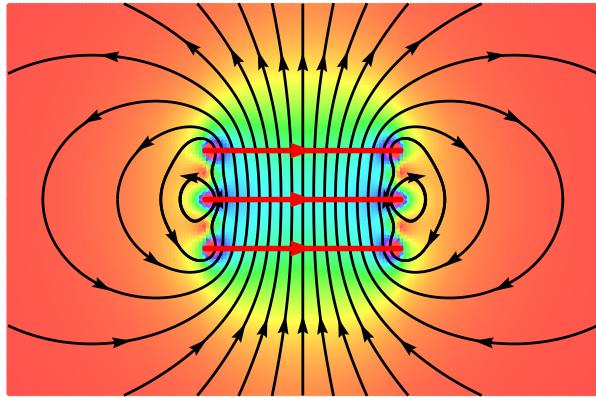


```
\psset{unit=0.5}
\begin{pspicture*}(-7,-8)(7,8)
\psmagneticfield[linecolor={[HTML]{006633}}],
```

```

N=1,R=2,nS=0] (-7,-8)(7,8)
\psframe*[linecolor={[HTML]{99FF66}}](-7,-8)(7,-7)
\rput(0,-7.5){\footnotesize[N=1,R=2,nS=0]}
\end{pspicture}
\begin{pspicture*}[showgrid](-7,-8)(7,8)
\psmagneticfield[linecolor={[HTML]{006633}}, N=2,R=2,L=2,PasS=0.003,nS=2](-7,-8)(7,8)
\psframe*[linecolor={[HTML]{99FF66}}](-7,7)(7,8)
\rput(0,7.5){\footnotesize Helmholtz}
\psframe*[linecolor={[HTML]{99FF66}}](-7,-8)(7,-7)
\rput(0,-7.5){\tiny[N=2,R=2,L=2,PasS=0.003,nS=2]}
\end{pspicture}

```



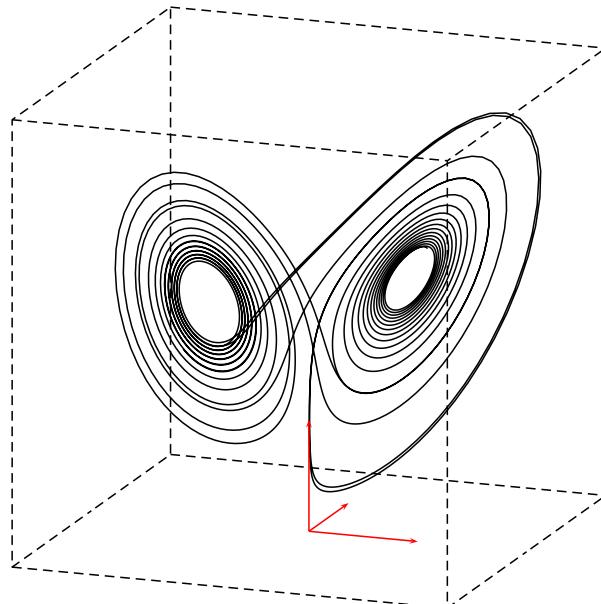
```

\begin{pspicture}(-6,-4)(6,4)
\psmagneticfield[N=3,R=2,L=2, StreamDensityPlot](-6,-4)(6,4)
\end{pspicture}

```

3.10 pst-ode

This package integrates differential equations using the Runge-Kutta-Fehlberg (RKF45) method with automatic step size control. Thus, the precision of the result does not depend on the number of plot points specified, as would be the case with the classical Runge-Kutta (RK4) method.

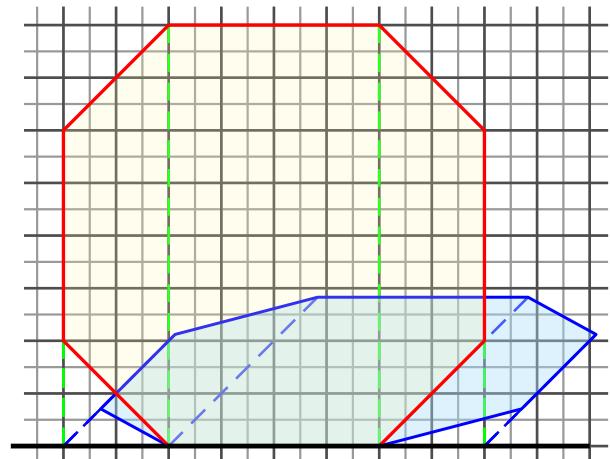


```

\begin{pspicture}(-8,-4)(6,12)
\pstVerb{ /alpha 10 def /beta 28 def
/gamma 8 3 div def }%
\pstODEsolve[algebraic]{lorenzXYZ}%
{0 1 2}{0}{25}{2501}{10 10 30}%
{ alpha*(x[1]-x[0])           |% x
x[0]*(beta-x[2]) - x[1] |% y
x[0]*x[1] - gamma*x[2]    % z
}
\psset{unit=0.17cm,Alpha=160,Beta=15}
\listplotThreeD{lorenzXYZ}%
\psset{unit=0.425cm,linestyle=dashed}
\pstThreeDNode(0,0,0){O}\pstThreeDNode(0,0,5){Z}
\pstThreeDNode(5,0,0){X}\pstThreeDNode(0,5,0){Y}
\pstThreeDNode(-10,-10,0){A}\pstThreeDNode(-10,-10,20){B}
\pstThreeDNode(-10,10,20){C}\pstThreeDNode(-10,10,0){D}
\pstThreeDNode(10,-10,0){E}\pstThreeDNode(10,-10,20){F}
\pstThreeDNode(10,10,20){G}\pstThreeDNode(10,10,0){H}
\pspolygon(A)(B)(C)(D)\pspolygon(E)(F)(G)(H)
\psline(A)(E)\psline(B)(F)\psline(D)(H)\psline(C)(G)
\psset{linestyle=solid,linecolor=red}
\psline{->}(0)(X)\psline{->}(0)(Y)\psline{->}(0)(Z)
\end{pspicture}

```

3.11 pst-perspective



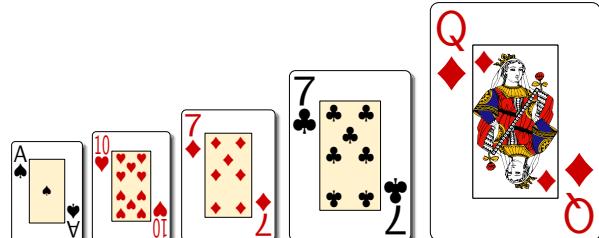
```

\begin{pspicture}(0.5,-0.5)(11.5,8.5)
\begin{psclip}
\psframe*[linestyle=none](0.25,-0.25)(11.35,8.35)
\psgrid[subgriddiv=2,gridlabels=0,gridwidth=0.7pt,
gridcolor=black!70,subgridwidth=0.6pt,
subgridcolor=black!40]{-1,-1}{13,10}
\end{psclip}
\psset{translineA=true,translineB=true,
linestyle=dashed,dash=5pt 3pt,linecolor=blue,
linejoin=2}
%----- create octagon -----
\pstransTS(3,0){A'}{A'}\pstransTS(7,0){B'}{B'}
\pstransTS(9,2){C'}{C'}\pstransTS(9,6){D'}{D'}
\pstransTS(7,8){E'}{E'}\pstransTS(3,8){F'}{F'}
\pstransTS(1,6){G'}{G'}\pstransTS(1,2){H'}{H'}
\pspolygon[fillstyle=solid,fillcolor=cyan!30,
opacity=0.4, linecolor=blue]%
(A')(B')(C')(D')(E')(F')(G')(H')
\pspolygon[fillstyle=solid,fillcolor=yellow!40,
opacity=0.2,linewidth=0.9pt,
linecolor=red](A)(B)(C)(D)(E)(F)(G)(H)
\pcline[linewidth=1.3pt](0,0|0)(11,0|0)
\end{pspicture}

```

3.12 pst-poker

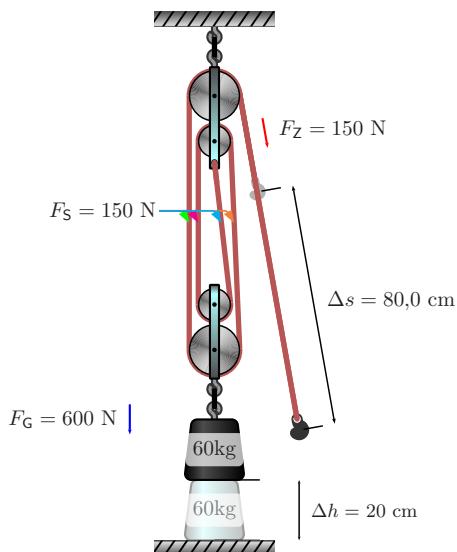
This is mostly a package for fun: it draws single or a group of poker cards. It can be displayed inline, like this:  or displayed as big cards:



```
\crdAs  
\psset{unit=1.1}\crdtenh \psset{unit=1.2}\crdsevd  
\psset{unit=1.3}\crdsevc \psset{unit=1.4}\crdqD
```

3.13 pst-pulley

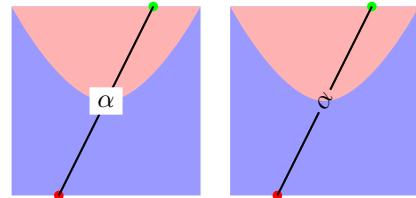
This package draws a nice view of a pulley, which may be of help to physics teachers in schools. There is only one macro which takes up to four optional parameters: $N=1\dots6$ gives the number of wheels of the pulley; $M=\dots$ gives the mass of the weight in kg ; $h=\dots$ gives the height of the weight in cm from the bottom.



```
\pspulleys [pulleyGrid=false ,N=4 ,M=60 ,h=20]
```

3.14 pst-rputover

The macro `\ncput*` places an object at the middle of two given nodes. It is a general method to mark lines. With a background color, it doesn't look especially good, as can be seen in the left example with the default behaviour. The package `pst-rputover` has the same effect but without using its own background color.



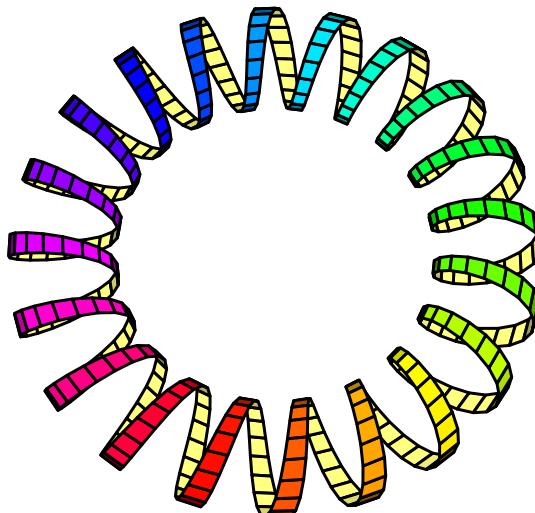
```

\begin{pspicture}(2,2)
\psframe*[linecolor=blue!40](0,0)(2,2)
\pscurve*[linecolor=red!30](0,2)(1,1)(2,2)
\pnode(.5,0){A}\psdot[linecolor=red](A)
\pnode(1.5,2){B}\psdot[linecolor=green](B)
\pcline(A)(B)\ncput*{$\alpha$}
\end{pspicture}\quad
\begin{pspicture}(2,2)
\psframe*[linecolor=blue!40](0,0)(2,2)
\pscurve*[linecolor=red!30](0,2)(1,1)(2,2)
\pnode(.5,0){A}\psdot[linecolor=red](A)
\pnode(1.5,2){B}\psdot[linecolor=green](B)
\pclineover(A)(B){$\alpha$}
\end{pspicture}

```

3.15 pst-ruban

This package draws ribbons (instead of lines) on three dimensional objects. It is an extension of the package `pst-solides3d` allowing you to draw ribbons on certain solids of revolution: cylinder, torus, sphere, paraboloid and cone. The width of the ribbon, the number of turns, the color of the external face as well as that of the inner face can be optionally specified.



```

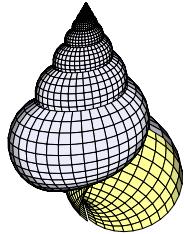
\psset{viewpoint=20 20 70 rtp2xyz,Decran=20,
       lightsrc=viewpoint,
       resolution=360,unit=0.6}
\begin{pspicture}(-5,-5)(5,5)
\psSpiralRing[incolor=yellow!50,r1=4,r0=1,hue=0 1]
\end{pspicture}

```

3.16 pst-shell

Geometric modeling of shellfish was carried out by Michael B. Cortie. In the “Digital Seashells” document he gives the parametric equations which are a

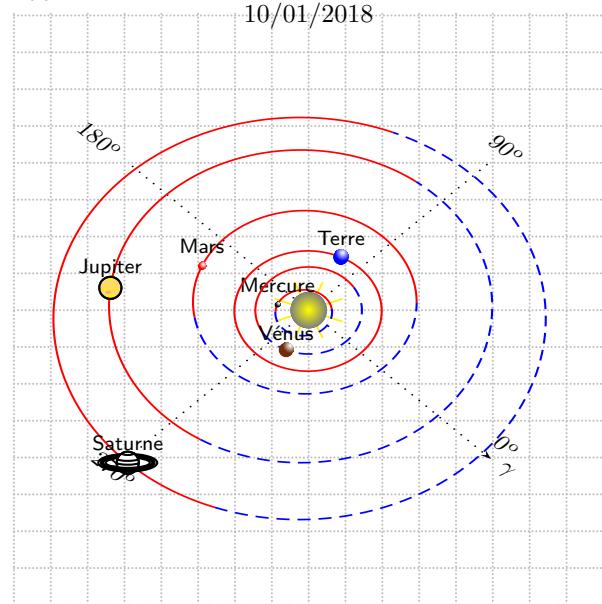
function of 14 parameters, in order to allow modeling of a very large number of shells (researchgate.net/publication/223141547_Digital_seashells).



```
\begin{pspicture}(-3,-7)(3,0)
\psset{lightsrc=viewpoint,
       viewpoint=800 -90 20 rtp2xyz,Decran=50}
\psShell[style=Escalaria,base=0 -7200 -180 180,
         ngrid=720 30,incolor=yellow!40,
         fillcolor=yellow!20!blue!10,linewidth=0.01pt]
\end{pspicture}
```

3.17 pst-solarsystem

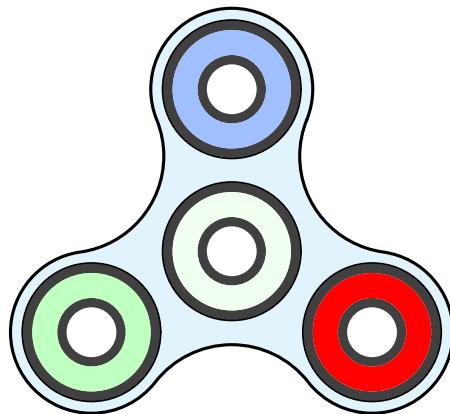
Position of the visible planets, projected on the plane of the ecliptic. The following example shows the solar system on Don Knuth's next magic birthday, at high noon.



```
\SolarSystem[Day=10,Month=01,Year=2018,Hour=12,
Minute=0,Second=0,viewpoint=1 -1 2,solarValues=false]
```

3.18 pst-spinner

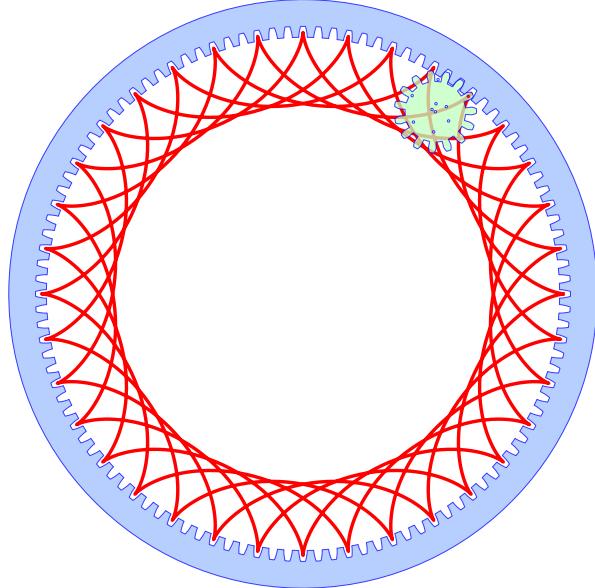
This package is just for fun. A fidget spinner is a type of stress-relieving toy.



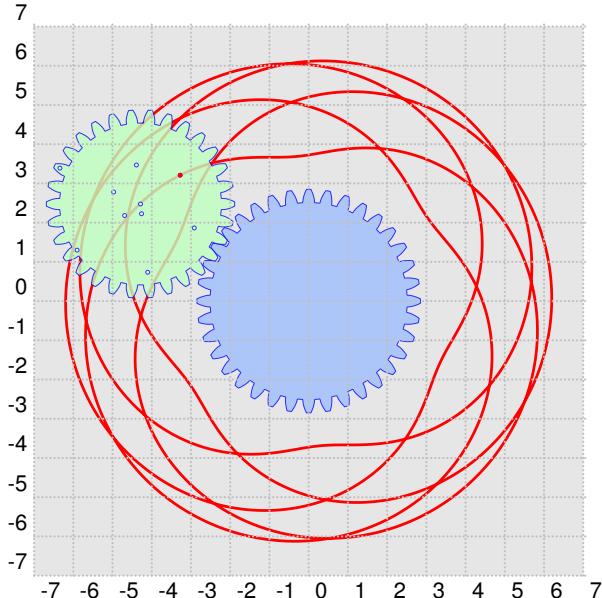
```
\begin{pspicture}(-4,-4)(5,4)
\psFidgetSpinner[fillcolor=cyan!10,linewidth=0.05,
                 mask=false](0,0)
\end{pspicture}
```

3.19 pst-spirograph

A spirograph is a geometric drawing toy that produces mathematical roulette curves that are technically known as hypotrochoids and epitrochoids. It is possible to draw inner and outer curves.



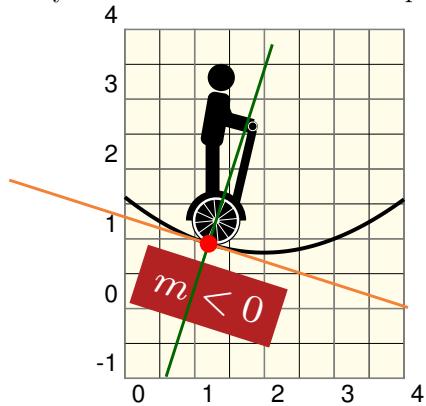
```
\begin{pspicture}(-7,-7)(7,7)
\psset{unit=0.5}
\psSpirograph[thetamax=-1800,Z1=108,Z2=15,
              m=0.2,linewidth=0.025,ap=10,
              fillstyle=solid,polarangle=54,
              linecolor=blue,holenumber=0,
              opacity=0.75]
\end{pspicture}
```



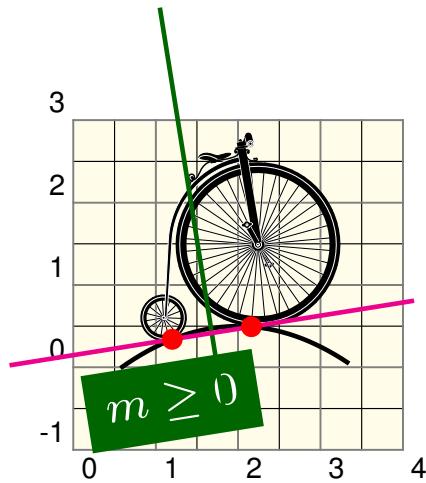
```
\begin{pspicture}[showgrid=top](-7,-7)(7,7)
\psframe*[linecolor=gray!20](-7,-7)(7,7)
\psSpirograph[thetamax=1800,Z1=36,Z2=30,m=0.15,
 linewidth=0.025,ap=20,inner=false,
 fillstyle=solid,polarangle=150,
 linecolor=blue,holenumber=4,opacity=0.8]
\end{pspicture}
```

3.20 pst-vehicle

This package provides slipping/rolling vehicles on curves of any kind of mathematical function, especially for animations in math and physics.



```
\def\FuncA{(x-3)*sin(0.2*(x-1))+1}
\begin{pspicture}(0,-1)(4,4)
\psframe*[linecolor=yellow!10](0,-1)(4,4)
\psgrid[style=quadrillage](0,-1)(4,4)
\psplot{0}{4}{\FuncA}
\psVehicle[vehicle=\Segway,
 style=segway]{0.25}{1.2}{\FuncA}
\end{pspicture}
```



```
\def\FuncA{-0.25*(x-2)^2+0.5}
\begin{pspicture}(0,-1)(4,3)
\psframe*[linecolor=yellow!10](0,-1)(4,3)
\psgrid[style=quadrillage](0,-1)(4,3)
\psplot[yMinValue=0]{0}{4}{\FuncA}
\psVehicle[vehicle=\HighWheeler]%
 {0.25}{1.2}{\FuncA}
\end{pspicture}
```

4 Summary

The Turing-complete PostScript programming language is old in computer terms, but provides many nice and useful graphical features. More information and many examples of PSTRicks can be found on the following websites:

- <http://pstricks.tug.org>
- <http://pstricks.blogspot.de>

References

- [1] Bill Casselman. *Mathematical Illustrations — A manual of geometry and PostScript*. Cambridge University Press, Cambridge, first edition, 2005.
- [2] Denis Girou. Présentation de PSTRicks. *Cahier GUTenberg*, 16:21–70, February 1994.
- [3] Frank Mittelbach, Michel Goossens, Sebastian Rahtz, Denis Roegel, and Herbert Voß. *The LATEX Graphics Companion*. Addison-Wesley, Boston, 2nd edition, 2006.
- [4] Herbert Voß. The current state of the PSTRicks project. *TUGboat*, 31(1):36–49, 2010. tug.org/TUGboat/tb31-1/tb97voss.pdf.
- [5] Timothy Van Zandt and Denis Girou. Inside PSTRicks. *TUGboat*, 15(3):239–248, September 1994. tug.org/TUGboat/tb15-3/tb44tvz.pdf.

◊ Herbert Voß
 Herbert.Voss (at) fu-berlin dot de

Glisternings

Peter Wilson

Our stars must glisten with new fire, or be
To daie extinct;

The Two Noble Kinsmen, JOHN
FLETCHER (and WILLIAM SHAKESPEARE?)

The aim of this column has been (see last sections) 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.

And scribbled lines like fallen hopes
On backs of tattered envelopes.

Instead of a Poet, FRANCIS HOPE

1 Reading lines

The `\input` macro reads a complete file into TeX as an atomic action. This was not what Lars Madsen needed when he posted to `cct` wanting to be able to read a file that consisted of blocks of lines of text, where a block was ended by a blank line, and then do something with the last non-blank of the block(s). The impetus for the following was Dan Luecking's posting [6], which was one of several responses.

The basis of a solution to Lars' problem is the TeX construct

`\read {stream} to \mymacro`

which reads one line from the file associated with `{stream}` and defines `\mymacro` to be the contents of that line.

Let's start with a file of the kind that Lars is concerned with. Using the `filecontents` environment, putting the following in the preamble will, if it does not already exist, create the file `glines16.txt` which will start with four TeX comment lines written by `filecontents`, stating how and when the file was created; if the `filecontents*` environment is used instead then the initial four comment lines are not output, just the body of the environment as given [7].

`\begin{filecontents}{glines16.txt}`

This is the file glines16.txt
containing some text lines.

They come in blocks
with blank lines between.

This is the third block
consisting of
three lines.

`\end{filecontents}`

Peter Wilson

We need to set up a `\read` stream and associate it with a file to be read, making sure that the file does exist, along the lines of:

```
\newread\instream \openin\instream= qwr!?.tex
\ifeof\instream
  \message{No file `qwr!?.tex'!^^J}
  \textbf{File `qwr!?.tex' not found!}
\else
  \message{File `qwr!?.tex' exists.^^J}
  \textbf{File `qwr!?.tex' exists.}
  % do something with qwr!?.tex
\fi
\closein\instream
```

File 'qwr!?.tex' not found!

Dan's statement was that:

If you
`\def\ispar{\par}`
and then
`\read <handle> to \myline`
you will find that
`\ifx\myline\ispar`
will be true for a blank line and also true for a `\read` taken after that last line of a file (when `\ifeof` is also true).

Putting all this together, the next piece of code produces the result shown afterwards.

```
\newcommand*{\ispar}{\par}
\newcommand*{\processfile}[1]{%
  \openin\instream=#1\relax
  \ifeof\instream
    \message{No file '#1'!^^J}%
    \textbf{File '#1' not found!}%
  \else
    \message{File '#1' exists.^^J}%
    \textbf{File '#1' exists!}%
    \par\noindent
    \loop
      \let\lastline\aline
      \read\instream to \aline
    \ifeof\instream\else
      \ifx\aline\empty (commentline) \\ \else
        \ifx\aline\ispar
          \ifx\lastline\ispar
            (blankline) \\
          \else
            (lastline) \lastline (followed by)\\
            (blankline) \\
        \fi
      \else
        (aline) \aline\\
      \fi
    \repeat
  \closein\instream
}\processfile{glines16.txt}
```

```
File 'glines16.txt' exists.
(commentline)
(commentline)
(commentline)
(commentline)
(aline) This is the file glines16.txt
(aline) containing some text lines.
(lastline) containing some text lines. (followed by)
(blankline)
(aline) They come in blocks
(aline) with blank lines between.
(lastline) with blank lines between. (followed by)
(blankline)
(blankline)
(aline) This is the third block
(aline) consisting of
(aline) three lines.
(lastline) three lines. (followed by)
(blankline)
```

When to the sessions of sweet silent thought
 I summon up remembrance of things past,
 I sigh the lack of many a thing I sought,
 And with old woes new wail my dear times' waste.

Sonnet 30, WILLIAM SHAKESPEARE

2 Paragraph endings

In earlier columns I described several aspects related to the typesetting of paragraphs [9, 10] and here are some additions to those.

2.1 Singletons

Andrei Alexandrescu wrote to *c_tt* [1] that:

My publisher has the rule that a single word on a line should not end a paragraph, as long as reflowing wouldn't make things really ugly otherwise. So I defined this macro:

```
\newcommand\lastwords[2]{%
  #1\leavevmode\penalty500\ \mbox{#2}}
```

and used it like this:

Lorem ipsum yadda \lastwords{amet}{dolor}.

The macro forces the last word never to be hyphenated, and imposes a penalty of 500 for inserting a line break between the first-to-last and the last word. My understanding is that 500 is the same penalty as that of a hyphen (by default).

Things work pretty well, but it turns out quite a lot of paragraphs need \lastwords — a whole 188 for a 500 page book ...

Is there an automated means to enact the rule above?

Suggestions ranged from ignoring the rule, to using existing code to make the last line at least *<some length>* long (see [9]), to code based on a further suggestion by Andrei and using *\everypar* that, subject to many caveats, implements the requirement.

Peter Flynn [2] suggested the macro

```
\def\E #1 #2.{ \mbox{#1}\~\mbox{#2}.}
```

which would be used like:

Lorem ipsum yadda \E amet dolor.

He observed that it was faster to type and easier to edit in than *\lastword* but noted that it might not handle arguments with embedded commands, spaces, curly braces, math, etc.

Dan Luecking [5] came up with corrections to Andrei's second suggestion, together with an extension to handle single-word paragraphs.

```
\usepackage{ifthen}
% handle (one word) paragraph, pass others on
\def\controlorphanword #1 #2\par{%
  \ifthenelse{\equal{#2}{}}%
    {#1\par}% one word para
    {\controlorphanwordtwo #1 #2\par}}
% to handle multi-word paragraph
\def\controlorphanwordtwo #1 #2 #3\par{%
  \ifthenelse{\equal{#3}{}}%
    {#1\leavevmode\penalty500\ \mbox{#2}\par}
    {#1 \controlorphanwordtwo #2 #3\par}}
```

Dan noted that these macros will not handle words separated by ‘\space’ or ‘\’, nor will it work with *\obeyspaces* in effect.¹ He also commented that the process seemed very inefficient.² The code should be called by using *\everypar* like:

```
\begin{document}
Normal paragraph. The macro
\cs{cs}\texttt{\{arg\}} will print \cs{arg}.
```

Another one, and introducing *\cs{everypar}*.

```
\everypar{\controlorphanword}
Almost every place I have ever read about
\cs{everypar} (or redefinition of
\cs{par} or changes to paragraph
parameters) there is this or similar caveat:
```

(La)TeX may have strange ideas what is counted as a paragraph. Use at your own risk, or turn it off in complicated circumstances.

Turn off orphan word control by putting *\everypar{}* here.

¹ For instance, using *\verb* when *\controlorphanword* is in effect will cause L^AT_EX to hiccup violently.

² The macros would be called for each word in a paragraph.

Turn it back on:

```
\everypar{\controlorphanword}
Sentence.
```

There was a general consensus among the respondents that the publisher's requirement was not particularly sensible, one going so far as to call it 'crazy'.

(Although not a solution to the problem as stated, too-short last lines can be mostly avoided in an entirely different way: `\parfillskip=.75\hsize plus .06\hsize minus .75\hsize`, with the numbers tweaked as desired, and with the usual caveats about packages resetting this primitive, etc.)

2.2 All is not what it seems

On rare occasions it may be desirable to either fake the end of a paragraph or to insert an invisible end of paragraph.

Faking the end is simple:

```
\newcommand*\{\\fakepar}{\\[\parskip]%
  \hspace*{\parindent}}
```

and it can be used as:

```
\ldots the end of a sentence.\fakepar
A new sentence looking as though it starts
a new paragraph\ldots
```

which will be typeset as:

```
...the end of a sentence.
```

A new sentence looking as though it starts a new paragraph...

Sometimes it is useful to nudge *TeX* into breaking a page, which it is inclined to do at the end of a paragraph while keeping the appearance of unbroken text. From *The TeXbook* [4, Ex. 14.15] and [12] the `\parnopar` macro accomplishes this:

```
\newcommand*\{\\parnopar}{%
  \parfillskip=0pt\par\parskip=0pt\noindent}}
```

TeX typesets paragraph by paragraph, initially taking no account of any page break. Only after the text has been set in lines does *TeX* consider if there should be a page break within the paragraph. If you need something different about the setting on the two pages, then the original paragraph must be split at the page break.

One application is when using the `changepage` package [11] to temporarily change the width or location of the textblock (e.g., like the `quote` environment). If you are trying to extend the `textwidth` into, say, the outer margin, which in two-sided documents is the left margin on even pages and the right margin on odd pages and there is a page break in the

shifted text then the results are not what you hoped for. This can be manually fixed using `\parnopar`, and splitting the adjustment into two.

```
\usepackage{changepage}
...
% move text 4em into outer margin
\begin{adjustwidth*}{0em}{-4em}
  ... first part of paragraph with the natural
  page break at this point\parnopar
\end{adjustwidth*}%
\begin{adjustwidth*}{0em}{-4em}
  but the sentence continues on the
  following page ...
\end{adjustwidth*}
```

2.3 Paraddendum

Selon Stan posted to `texhax`, asking [8]:

Is there a way to fill the last line of a paragraph with leaders that extend a fixed width beyond the edge of the paragraph, with right-aligned numbers on the right? I am trying ...

Paul Isambert [3] replied with code that I have cast into the following form:

```
\def\parend#1{%
  \leaders\hbox{\.,.}\hfill #1\par}
Here's a sentence.\parend{1}
Here's a sentence \\
on two lines.\parend{291}
```

Here's a sentence.	1
Here's a sentence	
on two lines.	291

The shades of night were falling fast,
 The rain was falling faster
 When through an Alpine village passed
 An Alpine village pastor;
 A youth who bore mid snow and ice
 With nary a sign of fluster
 A banner with a strange device—
 ‘Glingerings glister with lustre’.

The Shades of Night, A.E.
HOUSMAN & PETER WILSON

3 In conclusion

Some years ago, at Barbara Beeton's suggestion, I agreed to take over Jeremy Gibbons' *Hey — It works!* column which was published between 1993 and 2000 in, firstly, *TeX and TUG News*, and then later in *TUGboat*. He provided many useful tips for solving *LATEX* typesetting problems. Between 2000 and 2011 I managed to write some 15 columns, titled *Glingerings* as in 'All that glisters is not gold' carrying on Jeremy's work but then found that my circumstances had changed and I could no longer

produce a column on a regular basis. Also, the `comp.text.tex` newsgroup from which I got most of my inspiration seemed to be fading away, being replaced by `tex.stackexchange.com` which appealed to the younger generation but not to a GOM³ like me where many questions were directed towards problems with `tikz` graphics, the `beamer` package and ‘How do I produce this’.

I wrote a final 16th column trying to wrap everything up, but the wrapping ended up being so extensive that it would have taken up most of a *TUGboat* issue, so Karl decided that it would be best to split it up into several pieces and publish these over the coming years.⁴

You load sixteen tons and what do you get?
 Another day older and deeper in debt.
 Say brother, don’ you call me ’cause I can’t go
 I owe my soul to the company store.

Sixteen Tons, MERLE TRAVIS

4 Sixteen

For what I thought would be that final 16th *Glisternings* column I wrote the following, which perhaps might still be a suitable closing.

Sixteen is a rather remarkable number in that it can be expressed in many striking ways.

- In binary sixteen is: 10000
- In octal sixteen is: 20
- In decimal sixteen is: 16
- In hexadecimal sixteen is: 10

In decimal notation, which is the one most people are familiar with, there are quite a few ways in which sixteen can be represented. Among the more eye-catching ones are:

Powers

- $4^2 = 16$
- $2^4 = 16$
- $2^{2^2} = 16$

Additions

- sum of the first $\sqrt{16}$ odd numbers:
 $1 + 3 + 5 + 7 = 16$
- sum of adjacent numbers:
 $1 + 2 + 3 + 4 + 3 + 2 + 1 = 16$
 which can also be expressed as:
 $1 + 4 + 6 + 4 + 1 = 16$

Among other properties sixteen is the smallest number with exactly 5 divisors—1, 2, 4, 8 and 16. It is also the only number that is expressible as both m^n and n^m , with $m \neq n$.

³ Grumpy Old Man

⁴ I don’t think that either of us thought that ‘coming’ would turn out to be ‘next six’. [Editor’s note: So true.]

5 Acknowledgements

Glisternings would not have been possible without the support and input of many others. In particular I thank Jeremy Gibbons for his *Hey — It works!* and Barbara Beeton and Karl Berry for their enthusiasm and editorial improvements to the column. There are many others who also contributed, often unknowingly, by asking questions on the various *TeX* related mailing lists and to those who answered. With my grateful thanks to all of you.

References

- [1] Andrei Alexandrescu. Single word on a line at end of paragraph. `comp.text.tex`, 26 April 2010.
- [2] Peter Flynn. Re: Single word on a line at end of paragraph. `comp.text.tex`, 1 May 2010.
- [3] Paul Isambert. Re: [texhax] leaders protruding a fixed width from end of paragraph? `texhax` mailing list, 17 March 2011.
- [4] Donald E. Knuth. *The TeXbook*. Addison-Wesley, 1984. ISBN 0-201-13448-9.
- [5] Dan Luecking. Re: Single word on a line at end of paragraph. `comp.text.tex`, 28 April 2010.
- [6] Dan Luecking. Re: package for processing text from external files. `comp.text.tex`, 23 May 2011.
- [7] Scott Pakin. The `filecontents` package, 2009. ctan.org/pkg/filecontents.
- [8] Selon Stan. [texhax] leaders protruding a fixed width from end of paragraph? `texhax` mailing list, 17 March 2011.
- [9] Peter Wilson. Glisternings: Paragraphs regular, paragraphs particular, paragraphs Russian. *TUGboat*, 28(2):229–232, 2007. tug.org/TUGboat/tb28-2/tb89glister.pdf.
- [10] Peter Wilson. Glisternings: More on paragraphs regular, L^AT_EX’s defining triumvirate, *TeX*’s dictator. *TUGboat*, 29(2):324–327, 2008. tug.org/TUGboat/tb29-2/tb92glister.pdf.
- [11] Peter Wilson. The `changepage` package, 2009. ctan.org/pkg/changepage.
- [12] Peter Wilson. The `memoir` class for configurable typesetting, 2016. ctan.org/pkg/memoir.

◊ Peter Wilson
 12 Sovereign Close
 Kenilworth, CV8 1SQ, UK
`herries dot press (at)`
`earthlink dot net`

DocVar: Manage and use document variables

Zunbeltz Izaola and Paulo Ney de Souza

1 Introduction

In book production, we are frequently faced with the problem of using and reusing the same information in various locations of the products. Text strings like the title, author, ISBN, ... may appear in the book cover (in the front cover or the spine) and also in the colophon, as well in the text itself.

It is also desirable to be able to have “place holders” for this kind of information in templates that are used to produce each book of a collection. In the production of each book, ideally, the document will read these data from a database (or from an intermediate file derived from a database).

Figure 1 shows two books, members of the “Coleção Professor de Matemática” collection. You can see how they share the same design, while the first one shows one more piece of information: a subtitle. These two covers are generated from the same L^AT_EX file, but loading different metadata files.

Figure 2 shows a full cover of a book in the “Coleção Projecto Euclides” collection. It shows other kinds of metadata handled in a similar way (ISBN, title in the spine, ...).

2 Usage

The aim of our package is to facilitate the use of such “place holders” in a document, by making it easier to create, validate and use a document variable that is loaded from an external file.

At the beginning, we planned to call this package `metadata` because we are dealing with information that varies from book to book in a collection (author(s), title, date, subtitles, ...) and formatting variations (font size of title, subtitle, author(s), ...). But the name `metadata` has a rather specific meaning in the world of documents and there is already a package called `metadata` on CTAN. Therefore, the name Document Variable or DocVar for short, was selected.

The usage of document variables is done in three steps:

Define document variable Typically a class will define several docvars that individual documents, applying the class, will set and use.

Set value of document variable The value of all docvars used by a document is set; typically in a separate file that is included in the document.

Use value of document variable The docvar is replaced by the value to which it has been set.

2.1 Define document variables

Each `docvar` is defined by a unique `<key>`. This `<key>` is the mandatory argument of `\definedocvar`. There are optional arguments that control the behaviour of the `docvar`.

The syntax for the `\definedocvar` macro is: `\definedocvar[<option>]{<key>}`. Table 1 lists all the options planned for the `\definedocvar` macro. At the time of writing, not all the options are yet implemented.

The `docvar` can be of different types: integer, float, string, length. Some variables can have multiple values and they will be treated as list-type variables (for example, a book may have multiple authors). It is possible to transform the value by applying a macro; e.g., in some book designs the title is uppercase. The value of a `docvar` may be defined by “inheritance” from another variable: In most cases the name of authors printed on the spine of the book will be the same as on the cover, but sometimes the names should be modified (space limitations, design). It may be useful to define different “error levels” if the variable is empty. The error levels are the same as described for the variable validation. The DocVar package defines a mechanism to validate the values give to the each key. See section 2.4 for more details.

2.2 Set document variables

The macro sets the value of a previously defined `docvar`. The intent is to set the value of the `docvars` in a file loaded by a document, or alternatively to use the macro in the document `.tex` file itself.

The `\setdocvar` macro has two mandatory arguments, the `docvar` key and the value:

`\setdocvar{<key>}{<value>}`.

2.3 Use document variables

The macro `\getdocvar[<option>]{<key>}` retrieves the value of the `docvar`. In general, it will mean to *print* the value of the `docvar`, but `docvars` can also be used to set arguments of other macros.

`\getdocvar` accepts one option, `transform`; its value is a macro to be applied to the value of the `<key>`. This transformation is applied after any other transformation defined by `\setdocvar`.

2.4 Data validation

The process of validating data may be complex. On one hand we can validate the value of isolated `<key>`s, while on the other hand, we can validate the correctness of a value related to the value of other `<key>`s. For example, if a `docvar` represents a zip code, we can validate its format (as a single key value), but



Figure 1: Two examples of covers from the CPM collection.



Figure 2: A full cover produced using DocVar

Option	value	Implemented?	Definition
<code>type</code>	<code>integer, float, string, length</code>	No	Type of the variable
<code>multiple</code>	<code>true, false</code>	No	Set to true if the <code>docvar</code> has multiple values
<code>empty</code>		No	Behaviour of <code>\usedocvar</code> when value is empty
<code>inherit</code>	<code><key></code>	Yes	<code>docvar</code> from which value may be inherited
<code>transform</code>	<code>macro</code>	Yes	Always transform value before using it

Table 1: Options of the `\definedocvar` macro.

may also be validated in relation to a “state” `docvar`, to which it is related.

Each validation has an “error level”. There are five levels:¹

none Nothing happens.

info Information is logged in the log file.

warning A warning message is logged to the terminal and the log file.

error An error message is logged to the terminal and the log file, and an error mark is shown in the document.

critical After issuing a critical error, `TEX` will stop reading the current input file. This may halt the `TEX` run (if the current file is the main file) or may abort reading a sub-file.

fatal After issuing a fatal error the `TEX` run halts.

The validation is defined with the macro:

`\setvalidation[<error>]{<keys>}{{<validation>}}`

The macro’s first required argument `<keys>` is a comma-separated list; the list can have just one element. This is the list of related `<keys>` which will be validated together. The second mandatory argument is the macro to do the actual test. The optional argument is the *error level* associated to the test. The default value is `error`.

The package will provide several basic validation tests. The user-defined `<validation>` macro should accept as many arguments as `<keys>` are listed and it should return a boolean; true if the validation is passed and false if it fails. The `<validation>` macro will receive its arguments in the order given in `<keys>`.

Some validations may be too complex to be programmed efficiently in `LATEX`. Examples of using external scripting languages (Lua, Bash, Perl, Python) will be provided.

When a `docvar` is used with the `\getdocvar` command, the package will execute all the validations containing the corresponding `<key>`.²

3 Availability

The `DocVar` package is licensed under the LPPL, and copyrighted by Books in Bytes. The first public version should appear soon on CTAN; for development, see the repository at <https://gitlab.com/booksinbytes/docvar>.

- ◊ Zunbeltz Izaola
Durango
Spain
`zunbeltz (at) gmail dot com`
- ◊ Paulo Ney de Souza
Berkeley, CA
USA
`pauloney (at) gmail dot com,`
`paulo (at) berkeley.edu`
<http://booksinbytes.com/>

¹ These error levels are modelled after the `13msg` package error levels.

² This may not be efficient because the same validation will be run several times. But it seems to be the only way to show error messages close to the point where the `docvar` is used.

Set my (pdf)pages free

David Walden

Experienced (L^AT_EX users know that they can do many things with these systems. However, new users, for instance only having learned enough to typeset a thesis, may not think of some of the other possibilities. Below I describe one (admittedly trivial) use of L^AT_EX for something other than typesetting a document.

With fair frequency I receive PDF files from which I wish to extract pages or images but cannot (my collaborators may not know their word processors are creating protected files). Maybe if I knew more about such security settings, I could undo the protection in other ways. However, I do know that the following tiny L^AT_EX program has always “set my PDF pages free” in the way I wanted. The file for the following program is named `select-pages.tex`.

```
\documentclass{article}
\usepackage{pdfpages}
\begin{document}
\includepdf[pages=1-8]{name-of-file-to-be-freed.pdf}
\end{document}
```

I put a copy of the file in the directory with the PDF I want to set free, and then change the file name in the `\includepdf` command to the name of the file I want to unlock. I compile this little L^AT_EX program, and rename the result (which initially is `select-pages.pdf`) to be whatever I want it to be. Now I have a file which is no longer protected. (I don’t know why this works, but it does.)

All the work is done by the `pdfpages` package (ctan.org/pkg/pdfpages). In the above example, pages 1 to 8 of the original document are processed into the output file. If the optional argument in square brackets is left out, the entire input document is processed into the output file. Other options for the `pages` parameter are available, and the `pdfpages` package has lots of other options; read about it at the above noted url.

Once the desired “free” pages are in the new file, I have found I can now extract pages and copy images which Acrobat and other applications on my Windows computer previously would not let me touch except to read.

This is one minuscule example of how (L^AT_EX can do miscellaneous things for you. *TUGboat* has published many articles on using (L^AT_EX as a more general purpose computing tool than typesetting alone, and no doubt would welcome more.

◊ David Walden
walden-family.com/texland

Automatic generation of herbarium labels from spreadsheet data using L^AT_EX

R. Sean Thackurdeen and Boris Veytsman

Abstract

L^AT_EX, being a programmable language, has advanced capabilities for automatic generation of documents. While these capabilities are often considered the realm of advanced users, they are also attractive for entry-level users. The latter can use them to learn about L^AT_EX while performing a typesetting task. The goal of this tutorial is to describe a method to typeset herbarium labels using data stored in a `.csv` file. This example is especially relevant for the botanical research community, where labels must be generated from standardized data sets to annotate physical plant collections.

1 Botanical primer

Botanical vouchers are the foundation of the study of the evolutionary history of plants, known as systematics, and the study of their classification, known as taxonomy. They are the ontological basis on which botanical theories and hypotheses of evolution are made. Additionally, the study of specii and their niches (ecology), and the study of their distributions across temporal and spatial scales (biogeography) are allied sciences which draw from these instances of recorded plant life.

Botanical vouchers are composed of two components: 1) a specimen, and 2) a label. The specimen commonly features fertile plant parts as well as other distinguishing characteristics, such as leaf arrangement, developmental variation, etc. When combined with DNA evidence, it is used for classification and identification of a plant. The label presents information grounding a specimen in physical space. It is the written manifestation of the specimen’s identification, and includes collection information, geolocality data, and information about the habitat where the specimen was collected, in addition to other information about the specimen not apparent on the sheet.

Although not directly related to the present topic, readers may also be interested in the two articles by Joseph Hogg previously published in *TUGboat* (vol. 26, no. 1 and vol. 35, no. 2) on botanical typesetting: <http://tug.org/TUGboat/Contents/listauthor.html#Hogg>, Joseph.

2 Workflow

While botanists generally proceed by the adage, “by their fruits ye shall know them”, it can be more apt to say that “botanists make labels”.

```
Mj Gp,Scientific Name,Family,Genus,Specific Epithet,Taxon Rank,Infraspecific epithet,Scientific Name
Authorship,,IdentifiedBy,Date Identified,Identification Remarks,,Identification Qualifier,,Event Date,
Collector,Associated Collectors,collectorNumberPrefix,collectorNumber,collectorNumberSuffix,habitat,habit
,country,stateProvince,island,locality,localitySecurity,localitySecurityReason,geodeticDatum,
decimalLatitude,decimalLongitude,elevation (m.),duplicates,numberLabels,preparations,,tripNumber,
shippingPermit,shippingBox,shippingNote,,vernacularName1,languageName1,notesVernacularName1,
vernacularName2,vernacularLanguage2,notesVernacularName2,plantUse1,plantUseCategory1,plantUse2,
plantUseCategory2,informationWithheld,sourceNames,interviewers,interviewDate,,enteredBy
Angiosperm,Stachytarpheta jamaicensis (L.) Vahl,Verbenaceae,Stachytarpheta,jamaicensis,sp.,,(L.) Vahl,,,,,,,,,"June 7, 2014",Gregory M. Plunkett,"Michael Balick, Kate Armstrong, Sean Thackurdeen, Jean-Pascal Wahe, Presley Dovo & Joshua Andrew",,2783,,Growing in open area along roadside of disturbed secondary forest.,"Herb to subshrub, 0.5 m tall, flowers purple.",Vanuatu,Tafea ,Tanna,"West Tanna, just east of Lenakel, along track to Letakran Village, along creek.",,WGS84,-19.52803,169.2813,44,,6,6,"DNA, digital image
",,,,,,,,,,,,
Angiosperm,Ophioglossum reticulatum var. reticulatum L.,Ophioglossaceae,Ophioglossum,reticulatum,var.,
reticulatum,L.,,Gregory M. Plunkett,6/25/2014,,,!,"June 25, 2014",Gregory M. Plunkett,"Tom Ranker, Chanel Sam, Jean-Pascal Wahe, Sean Thackurdeen, Kate Armstrong, Laurence Ramon, Frazer Alo, Alexis Tupun, David Kapwia & Joseph Dabauh.",,2910,,Terrestrial fern growing in dense forest.,Vanuatu,Tafea ,Tanna,"Southwest Tanna, along trail from Yenhuip to Mount Tukosmera.",,WGS84,-19.588028,169.366611,559,,6,6,"DNA , digital image",,,,,,,,,,,
```

Figure 1: A three-line botanical .csv file (indented line breaks are editorial).

In the field, notes of a plant collection are made on weather resistant paper. At a moment's rest in the field, or back at home, data is provisionally transferred to a spreadsheet. The columns of the spreadsheet usually correspond to the database schema of the archival repository. Most often the schema adheres to the biodiversity standard known as Darwin Core (DwC). From this spreadsheet labels can be made using Microsoft Word's mail merge capabilities. Alternately, data can be uploaded to an intermediary (e.g., Filemaker) or to an archival repository that features reporting capabilities.

Producing labels directly from a field sheet allows a greater flexibility, since the labels can be generated anywhere a user has access to a computer with the requisite software. Unfortunately, the common work flow described above is unreliable. Once a spreadsheet is merged in Microsoft Word, any additional edits produce cascading effects which drastically alter the formatting of the document. The changes require an unnecessary amount of time and tedious effort. This issue can be alleviated through a reporting template used in intermediary repositories, but these systems are less flexible. A portable, field-ready and reliable solution is required to help botanists to avoid loss of time and to help them to make labels. Additionally, a system that is free and open source may be important for the botanists and collections managers in countries where herbaria lack extensive resources.

3 Tutorial

The tutorial which follows is a sequential, step by step, explanation of the L^AT_EX code which structures the document. As the tutorial proceeds, lines of code are grouped according to similarity of function. They are presented as blocks. While snippets of code are explained in relation to their function in the given example, possible alternatives are rarely explained. More detailed explanations of the options are better found L^AT_EX tutorials, of which there are many. We use the .csv file shown in Figure 1 for our examples.

First, a standard article class is called specifying the font size. The `geometry` package is used to specify the margins, and the `graphicx` and `datatool` packages are called to import images and spreadsheet values, respectively. The `datatool` package, authored by Nicola Talbot, is the key to this tutorial. It allows us to manipulate and typeset data stored in .csv files using L^AT_EX commands. Other approaches to automated document generation often rely on multiple programming languages to generate L^AT_EX code.

```
\documentclass[12pt]{article}
\usepackage{datatool,graphicx}
\usepackage[right=.2in, left=.2in,
top=.2in,bottom=.2in,
columnsep=.5in]{geometry}
```

The next portion of the preamble is a function designed to convert latitudes and longitudes in decimal degrees to degrees-minutes-seconds representation, for example, -19.588028 latitude to S 19°31'40".

The Darwin Core data standard for biodiversity data specifies decimal-degrees as the accepted standard for geographic data. However, this format is less reader friendly and thus less aesthetically pleasing. The following function transforms and typesets the GPS data so as to satisfy readers and data handlers alike, using datatool (`\DTL...`) commands.

```
% #1- negative suffix,
% #2 - positive suffix,
% #3 - lat/lon
\newcommand{\latlontodeg}[3]{%
\DTLifnumlt{#3}{0}{#1}{#2}%
\DTLabs{\TMPlatlon}{#3}%
\DTLtrunc{\TMPdeg}{\TMPlatlon}{0}%
\DTLsub{\TMPlatlon}{\TMPlatlon}{\TMPdeg}%
\DTLmul{\TMPlatlon}{\TMPlatlon}{60}%
\DTLtrunc{\TMPmin}{\TMPlatlon}{0}%
\DTLsub{\TMPlatlon}{\TMPlatlon}{\TMPmin}%
\DTLmul{\TMPlatlon}{\TMPlatlon}{60}%
\DTLtrunc{\TMPsec}{\TMPlatlon}{0}%
$ \TMPdeg` \circ \TMPmin' \TMPsec' $}
```

The next code block begins the document environment. Immediately, a datatool command is used to load the spreadsheet data, at which point the working database is named and the file, residing in the same directory, is specified.

```
\begin{document}
\DTLloadrawdb{labels}{labelExample.csv}
```

Herbarium labels are customarily 4" in width, and approximately 4" in length, varying with the amount of data recorded. One often prints 4 labels to a US letter page. To typeset multiple labels on a commonly available US letter sheet, we switch to two column layout. Two columns of a portrait US letter, with appropriate margins, produces the desired 4 inches width of herbarium labels.

```
\twocolumn
```

There are two parts to the datatool formula which will generate the labels: assignments and commands. The first will designate identifiers for each row in the spreadsheet. Once the database is defined, a working name is assigned to the .csv column name that is to be typeset. Below is an abbreviated version of the code. Note the spaces and capitalization on the right side of the equation, which refer to column names in your .csv file.

```
\DTLforeach{labels}{%
  \Family=Family,
  \Genus=Genus,
  \Species=Specific Epithet,
  \Authorship=Scientific Name Authorship}
```

Thus far we have defined the document size, its margins, created a multi-column environment, called a function and set assignments. Now, we begin the task of organizing the static and dynamic elements of the label.

While the two column typesetting allows text to flow from the base of one column to the beginning of the next one, we do not want an individual label to be continued on the next column or page. The text of a given label must be manipulated as a block. To create this environment we put the label inside a `minipage`:

```
{\noindent
\begin{minipage}{1.0\linewidth}%
\raggedright
\setlength{\parskip}{.5\baselineskip}%
\raisebox{-.5\height}
```

Next is a set of instructions to typeset a header. Many herbarium labels simply include a title which indicates flora of which the specimen is a part. In this case, an additional header with logos and herbarium codes is used. This can easily be customized as needed.

```
{\includegraphics[height=.8cm]{nybgLogo}}%
\hfill
\parbox[t]{5cm}{\centering\scshape\tiny
  New York Botanical Garden: NY \\
  Vanuatu National Herbarium: PVNH}%
\hfill
\raisebox{-.5\height}{%
\includegraphics[height=1cm]{pvnhLogo}}%
\par
{\centering \bfseries\itshape\large
  The Flora of Vanuatu\par}
```

Now we typeset the previously assigned elements. This is the heart of the approach. Here is an abbreviated example:

```
\DTLifnullorempty{\Family}{\hfill(\Family)}%
\DTLifnullorempty{\Genus}{\textit{\bfseries\Genus}}%
\DTLifnullorempty{\Species}{\textit{\bfseries\Species}}%
\DTLifnullorempty{\Authorship}{\Authorship}
```

We use `\DTLifnullorempty` to typeset the data. The program reads the assignment in the first set of braces. If the column does not have data, we skip it, hence the second set of blank braces. If it does have some data, we typeset it as instructed in the third set of braces with the additional formatting instructions.

The benefit of using the `\DTLifnullorempty` command is that static elements of the text can be nested in the third set of braces. This text will then be typeset, but only if the column is present. Thus we do not include ugly placeholders for missing information, as in some other approaches.

The last part of the label is an acknowledgment. This is currently set to be included on every label. Alternatively, it could be easily incorporated into the `\DTLifnullorempty` command and therefore be included only on specimen labels so designated. After the acknowledgment there is a command ending the label block, and a command to include space between each label. Then we close the document:

```
\centering\itshape\small
A collaboration of NYBG and PVNH,
funded by The~Christensen~Fund,
The~National~Geographic~Society,
and
The~Critical~Ecosystem~Partnership~Fund.

\end{minipage}%
\hspace{1cm}\par}
\end{document}
```

A page of example labels is given in Figure 2.

4 Notes on implementation

Many of the users adapting this code for use will stem from the natural history research community. Thus, it is worth mentioning a caveat about compiling documents from code. Computer programming languages, such as L^AT_EX, are exact and precise. If there are invalid characters in your data set, and you are unaware, you are sure to find out when you receive an error message (likely inscrutable) upon compiling. Similarly, if there are incorrect characters in your column mapping, an error message will result. While there are numerous possible errors, here are few tips to help you along.

- Compile in batches:
 - 100 rows returns a quick and sufficiently large output, while creating a smaller data-set to troubleshoot.
 - If your data set is greater than 100 rows, use the `split` command line tool to generate parts. The resulting files have no headers and these headers can be specified as an option to the `\DTLloaddb` command.
- Consider both `\DTLloaddb` & `\DTLloadrawdb`:
 - `\DTLloaddb` requires L^AT_EX special characters to be treated as such in your `.csv` file. A positive benefit to this is that you

can format specific text within cells using L^AT_EX syntax (e.g., taxon names in habitat descriptions).

- `\DTLloadrawdb` is needed when L^AT_EX special characters are present in the file. This command will automatically convert those special characters to the required L^AT_EX format.

- Clean code & clean data:

Issues compiling are probably due to either a syntax error in the code or invalid characters in your data set.

- Be mindful of stray spaces, generally, and capitalization when defining the column mapping.
- Be mindful of the input encoding and the text encoding.

5 Compile your own

A package including an example `.csv` file, L^AT_EX template, and output PDF will be posted to CTAN. This template will additionally be published to the online L^AT_EX compilers *Overleaf* and *ShareL^AT_EX*. Should you have questions regarding the template, feel free to reach out to the authors. Contact information is provided below.

- ◊ R. Sean Thackurdeen
Institute of Economic Botany
New York Botanical Garden
Bronx, NY 10458 USA
`sthackurdeen` (at) `nybg` dot `org`
<http://thackur.org/>
- ◊ Boris Veytsman
Systems Biology School and
Computational Materials
Science Center
MS 6A2
George Mason University
Fairfax, VA 22030 USA
`borisv` (at) `lk` dot `net`
<http://borisv.lk.net/>



NEW YORK BOTANICAL GARDEN: NY
VANUATU NATIONAL HERBARIUM: PVNH



The Flora of Vanuatu

(Verbenaceae)

Stachytarpheta jamaicensis (L.) Vahl

Vanuatu: Tafea. Tanna Island. West Tanna, just east of Lenakel, along track to Letakran Village, along creek. Growing in open area along roadside of disturbed secondary forest.

S 19°31'40"; E 169°16'52"; 44 m elev.

Herb to subshrub, 0.5 m tall, flowers purple. DNA, digital image. Duplicates: 6.

Gregory M. Plunkett, #2783 June 7, 2014

Michael Balick, Kate Armstrong, Sean Thackurdeen, Jean-Pascal Wahe, Presley Dovo & Joshua Andrew

A collaboration of NYBG and PVNH, funded by The Christensen Fund, The National Geographic Society, and The Critical Ecosystem Partnership Fund.



NEW YORK BOTANICAL GARDEN: NY
VANUATU NATIONAL HERBARIUM: PVNH



The Flora of Vanuatu

(Moraceae)

Ficus adenisperma Miq.

Vanuatu: Tafea. Tanna Island. West Tanna, just east of Lenakel, along track to Letakran Village, along creek. Growing along roadside of disturbed secondary forest.

S 19°31'40"; E 169°16'52"; 44 m elev.

Well branched tree, 12 m tall, 0.5 m dbh, fruits green turning yellow. DNA, digital image. Duplicates: 6.

Gregory M. Plunkett, #2784 June 7, 2014

Michael Balick, Kate Armstrong, Sean Thackurdeen, Jean-Pascal Wahe, Presley Dovo & Joshua Andrew.

A collaboration of NYBG and PVNH, funded by The Christensen Fund, The National Geographic Society, and The Critical Ecosystem Partnership Fund.



NEW YORK BOTANICAL GARDEN: NY
VANUATU NATIONAL HERBARIUM: PVNH



The Flora of Vanuatu

(Ophioglossaceae)

Ophioglossum reticulatum L.

Vanuatu: Tafea. Tanna Island. Southwest Tanna, along trail from Yenhup to Mount Tukosmera.

S 19°35'16"; E 169°21'59"; 559 m elev.

Terrestrial fern growing in dense forest. DNA, digital image. Duplicates: 6.

Gregory M. Plunkett, #2910 June 25, 2014

Tom Ranker, Chanel Sam, Jean-Pascal Wahe, Sean Thackurdeen, Kate Armstrong, Laurence Ramon, Frazer Alo, Alexis Tupun, David Kapwia & Joseph Dabauh.

A collaboration of NYBG and PVNH, funded by The Christensen Fund, The National Geographic Society, and The Critical Ecosystem Partnership Fund.



NEW YORK BOTANICAL GARDEN: NY
VANUATU NATIONAL HERBARIUM: PVNH



The Flora of Vanuatu

(Pteridaceae)

Antrophyum alatum Brack.

Vanuatu: Tafea. Tanna Island. West Tanna, just east of Lenakel, along track to Letakran Village, along creek.

S 19°31'40"; E 169°16'52"; 44 m elev.

Epipetric fern growing on boulder in dry stream bed. DNA, digital image. Duplicates: 6.

Gregory M. Plunkett, #2785 June 7, 2014

Michael Balick, Kate Armstrong, Sean Thackurdeen, Jean-Pascal Wahe, Presley Dovo & Joshua Andrew.

A collaboration of NYBG and PVNH, funded by The Christensen Fund, The National Geographic Society, and The Critical Ecosystem Partnership Fund.

Figure 2: A page of example labels

Typesetting actuarial symbols easily and consistently with `actuarialsymbol` and `actuarialangle`

David Beauchemin and Vincent Goulet

Abstract

Actuarial notation is characterized by subscripts and superscripts on both sides of a principal symbol, numbers positioned above or below subscripts, and some otherwise unusual symbols. The pair of packages `actuarialsymbol` and `actuarialangle` provides all the facilities to compose actuarial symbols of life contingencies and financial mathematics, easily and consistently.

1 Introduction

Actuaries, the “engineers of insurance”, denote various quantities of life contingencies using a whole array of symbols. The highly descriptive, yet compact, notation was standardized as far back as in 1898 [10]. Figure 1 shows a creative use of the notation by the graduating class of 1972 in Actuarial Science at Université Laval.

As most readers of *TUGboat* are probably unfamiliar with actuarial notation, let us start with the following examples:

1. the net single premium for an n -year term insurance payable at the end of year of death issued to a person aged x is $A_{x:\overline{n}}^1$;
2. the monthly premium for an annual life annuity payable at the beginning of the year, starting n years from now is $P^{(12)}(n|\ddot{a}_x)$;
3. the net reserve at time t for a whole life insurance payable at death is $_t\bar{V}(\bar{A}_x)$.

All symbols are for nominal benefits of 1.

Actuarial notation is characterized by auxiliary symbols positioned in subscript and superscript on both sides of a principal symbol, something notoriously difficult to achieve consistently in L^AT_EX. It also requires some unusual symbols not found in standard mathematics packages, like the “angle” denoting a duration n , as in \overline{n} , or the overhead angle bracket \overline{xy} used to emphasize the joint status of lives x and y when ambiguity is possible.

The package `actuarialsymbol` [1] provides a generic command to position all subscripts and superscripts easily and consistently around a principal symbol, four commands to position precedence numbers above and below statuses, and a number of shortcuts to ease entry of the most common actuarial functions of financial mathematics and life contingencies. The companion package `actuarialangle` [3], separate from `actuarialsymbol` for historical rea-

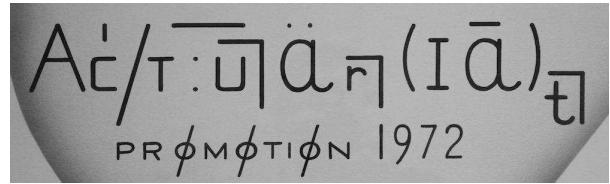


Figure 1: “Actuariat” (French for Actuarial Science) written using actuarial symbols on the 1972 graduating class mosaic at Université Laval

sons but imported by the latter, provides the angle and overhead angle bracket symbols.

2 Existing alternatives

Authors often use ad hoc constructions like `\{ _tA_x` to put subscripts and superscripts in front of a symbol. This notation quickly becomes a nightmare to parse mentally and the source code has little relationship to the actual significance of the symbol. That said, the worst practical drawback to this approach is probably that there is no way to ensure that subscripts and superscripts on either side of the principal symbol are aligned vertically.

The package `mathtools` [5] provides a command `\prescript` to put a subscript or superscript to the left of an argument. This works well when the argument (or principal symbol) has sub- and superscripts on all four corners, but otherwise the auxiliary symbols may end up at different heights.

Finally, various packages tailored for specific disciplines offer the possibility to position sub- and superscripts on the left, for example `tensor` [7] for tensors or `mhchem` [4] for isotopes. There was a previous attempt at a L^AT_EX package for actuarial notation [9], but `lifecon` does not seem to be officially distributed, either from CTAN or from anywhere else.

3 Actuarial notation

Appendix 4 of [2] offers an excellent overview of the composition rules for symbols of actuarial functions. In a nutshell, a principal symbol, say S , is combined with auxiliary symbols positioned in subscript or in superscript, to the left or to the right. Schematically, we thus have:



The principal symbol is in general a single letter. The letter may be “accented” with a bar (\bar{A}), double dots (\ddot{a}) or a circle (\dot{e}). Most commonly, there are alphanumeric statuses in the lower-right position III . Numerals can be placed above or below the individual statuses to show the order of failure; we will refer to these numerals as *precedence numbers*.

Otherwise, auxiliary symbols appear lower-left [I], upper-left [II] and upper-right [IV], in that order of frequency.

Symbols for benefit premiums (P), reserves (V) and amount of reduced paid-up insurance (W), are combined with benefit symbols unless the benefit is a level unit insurance payable at the end of the year of death. In such cases, we have the following symbol structure (replace P by V or W as needed):

$$\begin{array}{c} \text{II} \\ | \\ \text{I} \end{array} \quad P \quad \begin{array}{c} \text{IV} \\ | \\ \text{S} \end{array} \quad \begin{array}{c} \text{III} \\ | \\ \text{m} \end{array} \quad (2)$$

4 Additional special symbols

The package `actuarialangle` defines commands to draw two special symbols used in actuarial and financial notation. In math mode, the command

```
\angl{\duration}
```

composes an angle symbol around $\langle duration \rangle$ with some space (thin by default) between $\langle duration \rangle$ and the right descender. The symbol scales gracefully if the command is used outside of a first-level subscript.

```
\angl{n} \quad a_{\angl{n}} \quad \bar{n} \quad a_{\bar{n}}
```

Commands `\angln`, `\anglr` and `\anglk` are shortcuts for the common cases `\angl{n}`, `\angl{r}` and `\angl{k}`, respectively.

The code for `\angl` and the underlying macro were given to the second author by a colleague many years ago. The original author is unknown.

The command

```
\overanglebracket{\statuses}
```

composes an angle bracket (“roof”) above $\langle statuses \rangle$. The rule thickness and spacing relative to the statuses match those of the angle symbol. The command `\group` is a convenient alias for `\overanglebracket`.

```
\group{xy} \quad a_{\group{xy}:\angln} \quad \overline{xy} \quad A_{\overline{xy}:\bar{n}}
```

5 Construction of actuarial symbols

The package `actuarialsymbol` provides the generic command `\act symb` to typeset a principal symbol with surrounding subscripts and superscripts. Its syntax is somewhat unusual for L^AT_EX, but it serves well the natural order of the building blocks of a symbol and their relative prevalence:

```
\act symb[ll][ul]{symbol}{lr}[ur]
```

Above, $\langle ll \rangle$ identifies the auxiliary symbol in the lower left subscript position [I] (following the notation in the schematic representation (1)); $\langle ul \rangle$ is the upper left superscript [II]; $\langle symbol \rangle$ is the principal symbol S ; $\langle lr \rangle$ is the lower right subscript [III]; and

$\langle ur \rangle$ is the upper right superscript [IV]. The principal symbol and the right subscript are required, the other arguments are optional.

<code>\act symb{A}{x}</code>	A_x
<code>\act symb[n]{A}{x}</code>	nA_x
<code>\act symb[n][2]{A}{x}</code>	${}^2 nA_x$
<code>\act symb[n][2]{A}{x}[(m)]</code>	${}^2 nA_x^{(m)}$

The command `\act symb` supports one more optional argument, for composing symbols for premiums, reserves and paid-up insurance. The extended command

```
\act symb[ll][ul]{P}{symbol}{lr}[ur]
```

puts the symbol $\langle P \rangle$ outside the parentheses in the schematic representation (2).

<code>\act symb[] [] {P}{\bar{A}{x:\bar{n}}}</code>	$P(\bar{A}_{x:\bar{n}})$
<code>\act symb[k] [] [V]{\bar{A}{x:[\{1\}]}}</code>	${}_k V^{\{1\}}(\bar{A}_x)$
<code>\act symb[k] [] [\bar{W}]{\bar{A}{x}}</code>	${}_k \bar{W}(\bar{A}_x)$

Composing actuarial symbols from scratch using `\act symb` can easily get quite involved. For this reason, the package defines a large number of shortcut macros to ease entry of the most common symbols. Table 1 offers a glimpse of the available shortcuts; the package documentation has the complete list.

The definition of `\act symb` is heavily inspired by the code of `\prescript` from package `mathtools` which, as reported by the author, is itself based on a posting to `comp.text.tex` by Michael J. Downes.

6 Positioning of subscripts

T_EX adjusts the position of a subscript downward when a superscript is present:

$$A_x \quad A_x^2.$$

Command `\act symb` maintains this behavior, something we believe to be a desirable feature. Therefore, entering the symbols above using the standard operators `^` and `_` or with `\act symb` yields the same result.

<code>A_x \quad A_x^2</code>	$A_x \quad A_x^2$
<code>\act symb{A}{x} \quad \act symb{A}{x}^2</code>	$A_x \quad A_x^2$

Furthermore, the command ensures that the left and right subscripts, when both present, are at the same level, something common ad hoc constructions do not provide.

<code>{}_t A_x \quad {}_t A_x^2</code>	$t A_x \quad t A_x^2$
<code>\act symb[t]{A}{x} \quad \act symb[t]{A}{x}^2</code>	$t A_x \quad t A_x^2$

Authors who would prefer a uniform subscript position *throughout their document* can load the package `subdepth` [8].

Table 1: Sample of shortcuts for life table, insurance and annuity symbols.
All commands accept the optional arguments $\langle ll \rangle$, $\langle ul \rangle$ and $\langle ur \rangle$ of `\actssymb`.

Definition	Example	Output
<code>\lx{\langle age \rangle}</code>	<code>\lx{x}</code>	ℓ_x
<code>\dx{\langle age \rangle}</code>	<code>\dx[n]{x}</code>	${}_n d_x$
<code>\px{\langle age \rangle}</code>	<code>\px[t]{x}</code>	$t p_x$
<code>\qx{\langle age \rangle}</code>	<code>\qx[t]{x}</code>	$t q_x$
<code>\erinx{\langle lr \rangle}</code>	<code>\erinx{x:\angln}</code>	$\dot{e}_{x:\bar{n}}$
<code>\Ax{\langle lr \rangle}</code>	<code>\Ax{x:\angln}</code>	$A_{x:\bar{n}}$
<code>\Ax*\{\langle lr \rangle\}</code>	<code>\Ax*[x:\angln]</code>	$\bar{A}_{x:\bar{n}}$
<code>\Ex{\langle lr \rangle}</code>	<code>\Ex[n]{x}</code>	${}_n E_x$
<code>\ax{\langle lr \rangle}</code>	<code>\ax{x:\angln}</code>	$a_{x:\bar{n}}$
<code>\ax*\{\langle lr \rangle\}</code>	<code>\ax*[x:\angln]</code>	$\bar{a}_{x:\bar{n}}$
<code>\ax**{\langle lr \rangle}</code>	<code>\ax**{x:\angln}</code>	$\ddot{a}_{x:\bar{n}}$
<code>\aringx{\langle lr \rangle}</code>	<code>\aringx{x:\angln}</code>	$\dot{\ddot{a}}_{x:\bar{n}}$

7 Precedence numbers

Precedence numbers appear above or below individual statuses in the right subscript \boxed{III} of a symbol. The commands

```
\nthtop[⟨length⟩]{⟨number⟩}{⟨status⟩}
\nthbottom[⟨length⟩]{⟨number⟩}{⟨status⟩}
```

put a precedence $\langle number \rangle$ above (resp. below) a $\langle status \rangle$, smashed so that the apparent height of the status is its normal height.

<pre>\actssymb{A}{\nthtop{1}{x}:\angln} \actssymb{A}{x:\nthtop{1}{\angln}} \actssymb{A}{\nthtop{1}{x}y:% \nthtop{2}{\angln}} \actssymb{A}{\nthtop{3}{x}% \nthbottom{1}{y}\nthbottom{2}{z}}</pre>	$A_{x:\bar{n}}^1$ $A_{x:\bar{n}}^1$ $A_{xy:\bar{n}}^2$ $A_{xyz:\bar{n}}^{12}$
--	--

As can be seen in the third and fourth examples above, the constant spacing between the precedence number and the status can result in numbers placed at different heights if one status contains a horizontal rule or a descender. To cope with this situation, we provide * variants of the commands that always align precedence numbers vertically.

<pre>\actssymb{A}{\nthtop*{1}{x}y:% \nthtop*{2}{\angln}} \actssymb{A}{\nthtop*{3}{x}% \nthbottom*{1}{y}\nthbottom*{2}{z}}</pre>	$A_{xy:\bar{n}}^{1,2}$ $A_{xyz:\bar{n}}^{12}$
---	--

The fact that top precedence numbers have zero height means they will clash with a right superscript \boxed{IV} .

<pre>\actssymb{A}{\nthtop{1}{x}:\angln}[(m)]</pre>	$A_{x:\bar{n}}^{(m)}$
--	-----------------------

For such rare circumstances, we left to the user to insert a strut in the subscript to push it downward as needed.

<pre>\actssymb{A}{\rule{0pt}{2.3ex}% \nthtop{1}{x}:\angln}[(m)]</pre>	$A_{x:\bar{n}}^{(m)}$
---	-----------------------

This remark also applies to bottom precedence numbers in inline formulas or multiline equations.

The optional argument $\langle length \rangle$ of `\nthtop` and `\nthbottom` changes the default spacing between the number and the status for one symbol. This can also be changed globally by redefining lengths mentioned in the documentation of `actuarialsymbol`.

The package defines shortcuts `\itop`, `\iitop` and `\iiitop` for first, second and third top precedence (and their analogues for bottom precedence).

The system of precedence numbers builds on a macro that used to be part of `actuarialangle`. As with the code for `\angl`, the original author is unknown.

8 Other functionalities

For brevity, we have omitted some additional features of `actuarialsymbol`, including macros to typeset two-letter symbols such as (IA) , numerous shortcut macros and quite fancy utilities to define new ones. The package documentation provides all the details.

Following [9], the package documentation also contains a *Comprehensive list of life contingencies symbols*. The wording used here should be taken for its intended purpose, namely to acknowledge Scott Pakin's immensely useful *Comprehensive L^AT_EX Symbol List* [6].

References

- [1] David Beauchemin and Vincent Goulet. *Actuarial symbols of life contingencies and financial mathematics*, 2017. ctan.org/pkg/actuarialsymbol.
- [2] Newton L. Bowers, Hans U. Gerber, James C. Hickman, Donald A. Jones, and Cecil J. Nesbitt. *Actuarial Mathematics*. Society of Actuaries, Schaumburg, IL, second edition, 1997.
- [3] Vincent Goulet. *Actuarial angle symbol for life contingencies and financial mathematics*, 2017. ctan.org/pkg/actuarialangle.
- [4] Martin Hensel. *The mhchem Bundle*, 2017. ctan.org/pkg/mhchem.
- [5] Morten Høgholm and Lars Madsen. *The mathtools package*, 2015. ctan.org/pkg/mathtools.
- [6] Scott Pakin. *The Comprehensive L^AT_EX Symbol List*, 2015. ctan.org/pkg/comprehensive.
- [7] Philip G. Ratcliffe. *The tensor package for L^AT_EX2_e*, 2004. ctan.org/pkg/tensor.
- [8] Will Robertson. *Unify subscript depths*, 2007. ctan.org/pkg/subdepth.
- [9] Eddy Trivedi. *Life Contingencies' Symbols*, 2004. *lifecon 2.1 User Guide*.
- [10] Henk Wolthuis. International actuarial notation. In Jozef Teugels and Bjørn Sundt, editors, *Encyclopedia of Actuarial Science*. Wiley, 2004. onlinelibrary.wiley.com/book/10.1002/9780470012505.
 - ◊ David Beauchemin
[david.beauchemin.5\(at\)ulaval.ca](mailto:david.beauchemin.5(at)ulaval.ca)
 - ◊ Vincent Goulet
 École d'actuariat
 Université Laval
 Pavillon Paul-Comtois
 2425, rue de l'Agriculture, Bureau
 4153
 Québec (QC) G1V 0A6
 Canada
[vincent.goulet\(at\)act.ulaval.ca](mailto:vincent.goulet(at)act.ulaval.ca)
<https://vgoulet.act.ulaval.ca>

Converting T_EX from WEB to cweb

Martin Ruckert

Why translate T_EX from WEB to cweb?

A long term goal brought me to construct the program `web2w` that translates T_EX from WEB to cweb: I plan to derive from the T_EX sources a new kind of T_EX that is influenced by the means and necessities of current software and hardware.

The major change in that new kind of T_EX will be the separation of the T_EX frontend: the processing of `.tex` files, from the T_EX backend: the rendering of paragraphs and pages.

Let's look, for example, at ebooks: Current ebooks are of rather modest typographic quality. Just compiling T_EX documents to a standard ebook format, for example `epub`, does not work because a lot of information that is used by T_EX to produce good looking pages is not available in these formats. So I need to cut T_EX in two pieces: a frontend that reads T_EX input, and a backend that renders pixels on a page. The frontend will not know about the final page size because the size of the output medium may change while we read—for example by turning a mobile device from landscape to portrait mode. On the other hand, the computational resources of the backend are usually limited because a mobile device has a limited supply of electrical energy. So we should do as much as we can in the frontend and postpone only what needs to be postponed to the backend. In between front and back, we need a nice new file format that is compact and efficient and transports the necessary information between both parts.

For the work described above, I will need to work with the T_EX source code and make substantial changes. The common tool chain from the T_EX Live project uses `tangle` to convert `tex.web` into Pascal code (`tex.pas`) which is then translated by `web2c` into C code. In the course of this process also other features of a modern T_EX distribution are added. Hence the translation process is not just a syntactic transformation but also introduces semantic changes. So it seemed not the best solution for my project. Instead, I wanted to have cweb [7] source code for T_EX, which I could modify and translate to C simply by running `ctangle`.

The result of my conversion effort was surprisingly good, so I decided to make it available on ctan.org [10] and to present it here, in the hope others may find it useful when tinkering with T_EX.

How the program web2w was written

On December 9, 2016, I started to implement `web2w` with the overall goal to generate a `tex.w` file that is as close as possible to the `tex.web` input file, and can be used to produce `tex.tex` and `tex.c` simply by running the standard tools `ctangle` and `cweave`.

`web2w` was not written following an established software engineering workflow as we teach it in our software engineering classes. Instead the development was driven by an ongoing exploration of the problem at hand where the daily dose of success or failure would determine the direction I would go on the next day.

This description of my program development approach sounds a bit like “rapid prototyping”. But “prototype” implies the future existence of a “final version” and I do not intend to produce such a “final version”. Actually I have no intention to finish the prototype either, and I might change it in the future in unpredictable ways. Instead I have documented the development process as a literate program [6]. So in terms of literature, this is not an epic novel with a carefully designed plot, but more like the diary of an explorer who sets out to travel through yet uncharted territories.

The territory ahead of me was the program `TEX` written by Donald E. Knuth using the `WEB` language [4] as a literate program. As such, it contains snippets of code in the programming language Pascal—Pascal-H to be precise. Pascal-H is Charles Hedrick’s modification of a compiler for the DEC-system-10 that was originally developed at the University of Hamburg (cf. [1], see [5]). So I could not expect to find a pure “Standard Pascal”. But then the implementation of `TEX` deliberately does not use the full set of features that the Pascal language offers. Hence at the beginning, it was unclear to me what problems I would encounter with the subset of Pascal that is actually used in `TEX`.

Further, the problem was not the translation of Pascal to C. A program that does this is available as part of the `TEX` Live project: `web2c` [11] translates the Pascal code that is produced using `tangle` from `tex.web` into C code. The C code that is generated this way cannot, however, be regarded as human readable source. The following example might illustrate this: figure 1 shows the `WEB` code for the function `new_null_box`. The result of translating it to C by `web2c` can be seen in figure 3. In contrast, figure 2 shows what `web2w` will achieve.

`web2c` has desugared the sweet code written by Knuth to make it unpalatable to human beings; the

136. The `new_null_box` function returns a pointer to an `hlist_node` in which all subfields have the values corresponding to ‘`\hbox{}`’. The `subtype` field is set to `min_quarterword`, since that’s the desired `span_count` value if this `hlist_node` is changed to an `unset_node`.

```
function new_null_box: pointer;
  { creates a new box node }
  var p: pointer;  { the new node }
  begin p ← get_node(box_node_size);
    type(p) ← hlist_node; subtype(p) ← min_quarterword;
    width(p) ← 0; depth(p) ← 0; height(p) ← 0;
    shift_amount(p) ← 0; list_ptr(p) ← null;
    glue_sign(p) ← normal; glue_order(p) ← normal;
    set_glue_ratio_zero(glue_set(p)); new_null_box ← p;
  end;
```

Fig. 1: WEB code of `new_null_box`

136. The `new_null_box` function returns a pointer to an `hlist_node` in which all subfields have the values corresponding to ‘`\hbox{}`’. The `subtype` field is set to `min_quarterword`, since that’s the desired `span_count` value if this `hlist_node` is changed to an `unset_node`.

```
pointer new_null_box(void)
  /* creates a new box node */
  { pointer p;  /* the new node */
    p = get_node(box_node_size); type(p) = hlist_node;
    subtype(p) = min_quarterword; width(p) = 0;
    depth(p) = 0; height(p) = 0; shift_amount(p) = 0;
    list_ptr(p) = null; glue_sign(p) = normal;
    glue_order(p) = normal;
    set_glue_ratio_zero(glue_set(p)); return p;
  }
```

Fig. 2: cweb code of `new_null_box`

only use you can make of it is feeding it to a C compiler. In contrast, `web2w` tries to create source code that is as close to the original as possible but still translates Pascal to C. For example, see the last statement in the `new_null_box` function: where C has a `return` statement, Pascal assigns the return value to the function name. A simple translation, sufficient for a C compiler, can just replace the function name by “`Result`” (an identifier that is not used in the implementation of `TEX`) and add “`return Result;`” at the end of the function (see figure 3). A translation that strives to produce nice code should, however, avoid such ugly code.

The structure of `web2w`

The program `web2w` works in three phases: First I run the input file `tex.web` through a scanner producing tokens. The pattern matching is done using `flex`. During scanning, information about macros,

```

halfword
newnullbox ( void )
{
  register halfword Result; newnullbox_regmem
  halfword p ;
  p = getnode ( 7 ) ;
  mem [p ].hh.b0 = 0 ;
  mem [p ].hh.b1 = 0 ;
  mem [p + 1 ].cint = 0 ;
  mem [p + 2 ].cint = 0 ;
  mem [p + 3 ].cint = 0 ;
  mem [p + 4 ].cint = 0 ;
  mem [p + 5 ].hh .v.RH = -268435455L ;
  mem [p + 5 ].hh.b0 = 0 ;
  mem [p + 5 ].hh.b1 = 0 ;
  mem [p + 6 ].gr = 0.0 ;
  Result = p ;
  return Result ;
}

```

Fig. 3: web2c code of *new_null_box*

identifiers, and modules is gathered and stored. The tokens then form a doubly linked list, so that later I can traverse the source file forward and backward. Further, every token has a `link` field which is used to connect related tokens. For example, I link an opening parenthesis to the matching closing parenthesis, and the start of a comment to the end of the comment.

After scanning comes parsing. The parser is generated using `bison` from a modified Pascal grammar [3]. To run the parser, I feed it with tokens, rearranged to the order that `tangle` would produce, expanding macros and modules as I go. While parsing, I gather information about the Pascal code. At the beginning, I tended to use this information immediately to rearrange the token sequence just parsed. Later, I learned the hard way (modules that were modified on the first encounter would later be fed to the parser in the modified form) that it is better to leave the token sequence untouched and just annotate it with information needed to transform it in the next stage.

A technique that proved to be very useful is connecting the key tokens of a Pascal structure using the `link` field. For example, connecting the “`case`” token with its “`do`” token makes it easy to print the expression that is between these tokens without knowing anything about its actual structure and placing it between “`switch` (” and “`)`”.

The final stage is the generation of cweb output. Here the token sequence is traversed again in input file order. This time the traversal will stop at the warning signs put up during the first two passes,

```

function new_character(f : internal_font_number;
                      c : eight_bits): pointer;
label exit;
var p: pointer; { newly allocated node }
begin if font_bc[f] ≤ c then
  if font_ec[f] ≥ c then
    if char_exists(char_info(f)(qi(c))) then
      begin p ← get_avail; font(p) ← f;
      character(p) ← qi(c); new_character ← p;
      return;
      end;
    char_warning(f,c); new_character ← null;
exit: end;

```

Fig. 4: WEB code of *new_character*

```

pointer new_character(internal_font_number
                      f,eight_bits c)
{ pointer p; /*newly allocated node*/
  if (font_bc[f] ≤ c)
    if (font_ec[f] ≥ c)
      if (char_exists(char_info(f)(qi(c)))) {
        p = get_avail(); font(p) = f;
        character(p) = qi(c); return p;
      }
    char_warning(f,c); return null;
}

```

Fig. 5: cweb code of *new_character*

use the information gathered so far, and rewrite the token sequence as gently and respectfully as possible from Pascal to C.

Et voilà! `tex.w` is ready — well, almost. I have to apply a last patch file, for instance to adapt documentation relying on `webmac.tex` so that it will work with `cwebmac.tex`, and make changes that do not deserve a more general treatment. The final file is then called `ctex.w` from which I obtain `ctex.c` and `ctex.tex` merely by applying `ctangle` and `cweave`. Using “`cc ctex.c -lm -o ctex`”, I get a running `ctex` that passes the trip test.

Major challenges

I have already illustrated the different treatment of function return values in Pascal and C with figures 1 and 2. Of course, replacing “`new_null_box ← p;`” by “`return p;`” is a valid transformation only if the assignment is in a tail position. A tail position is a position where the control flow directly leads to the end of the function body as illustrated by figure 4 and 5. It is possible to detect tail positions by traversing the Pascal parse tree constructed during phase 2.

The `return` statement inside the `if` in figure 5 is correct because the Pascal code in figure 4 contains

a “**return**”. This “**return**”, however, is a macro defined as “**goto exit**”, and “**exit**” is a numeric macro defined as “10”. In C, “**return**” is a reserved word and “**exit**” is a function in the C standard library, so something has to be done. Fortunately, if all goto-statements that lead to a given label can be eliminated, as is the case in figure 5, the label can be eliminated as well. So you see no “**exit:**” preceding the final “}”.

Another seemingly small problem is the different use of semicolons in C and Pascal. While in C a semicolon follows an expression to make it into a statement, in Pascal the semicolon connects two statements into a statement sequence. For example, if an assignment precedes an “**else**”, in Pascal you write “**x:=0 else**” whereas in C you write “**x=0; else**”; but no additional semicolon is needed if a compound statement precedes the “**else**”. When converting **tex.web**, a total of 1119 semicolons need to be inserted at the right places. Speaking of the right place: Consider the following **WEB** code:

```
if s ≥ str_ptr then s ← "???"  
    { this can't happen }  
else if s < 256 then
```

Where should the semicolon go? Directly preceding the “**else**”? Probably not! I should insert the semicolon after the last token of the assignment. But this turns out to be rather difficult: assignments can be spread over several macros or modules which can be used multiple times; so the right place to insert a semicolon in one instance can be the wrong place in another instance. **web2w** starts at the **else**, searches backward, skips the comment and the newlines, and then places the semicolon like this:

```
if (s ≥ str_ptr) s = < "???" 1381>;  
    /* this can't happen */  
else if (s < 256)
```

But look at what happened to the string “**???**”. Strings enclosed in C-like double quotes receive a special treatment by **tangle**: the strings are collected in a string pool file and are replaced by string numbers in the Pascal output. No such mechanism is available in **ctangle**. My first attempt was to replace the string handling of **TeX** and keep the C-like strings in the source code. The string pool is, however, hardwired into the program and it is used not only for static strings but also for strings created at runtime, for example to hold the names of control sequences. So I tried a hybrid approach: keeping strings that are used only for output (error messages for example) and translating other strings to string numbers using the module expansion mechanism of **ctangle**, like this:

1381.

```
#define str_256 "???"
```

```
< "???" 1381> ≡ 256
```

This code is used in section 59.

I generate for each string a module, that will expand to the correct number, here 256; and I define a macro **str_256** that I use to initialize the string pool variables.

In retrospect, seeing how nicely this method works, I ponder if I should have decided to avoid the hybrid approach and used modules for all strings. It would have reduced the number of changes to the source file considerably.

Another major difference between Pascal and C is the use of subrange types. In Pascal subrange types are used to specify the range of valid indices when defining arrays. While most arrays used in **TeX** start with index zero, not all do. In the first case they can be implemented as C arrays which always start at index zero; in the latter case, I define a zero based array having the right size, and add a “0” to the name. Then I define a constant pointer initialized by the address of the zero based array plus/minus a suitable offset so that I can use this pointer as a replacement for the Pascal array.

When subrange types are used to define variables, I replace subrange types by the next largest C standard integer type as defined in **stdint.h**—which works most of the time. But consider the code

```
var p: 0 .. nest_size; { index into nest }  
:  
for p ← nest_ptr downto 0 do  
where nest_size = 40. Translating this to  
uint8_t p; /* index into nest */  
:  
for (p = nest_ptr; p ≥ 0; p--)
```

would result in an infinite loop because **p** would never become less than zero; instead it would wrap around. So in this (and 21 similar cases), variables used in **for** loops must be declared to be of type **int**.

Related work

As described by Taco Hoekwater in “LuaTeX says goodbye to Pascal” [2], the source code of **TeX** was rewritten as a part of **LuaTeX** project as a collection of **cweb** files. This conversion proceeded in two steps: first **TEX.WEB** was converted into separate plain C files while keeping the comments; at a much later date, those separate files were converted back

341. Now we're ready to take the plunge into *get_next* itself. Parts of this routine are executed more often than any other instructions of \TeX .

```

define switch = 25 { a label in get_next }
define start_cs = 26 { another }

procedure get_next;
    { sets cur_cmd, cur_chr, cur_cs to next token }
label restart, { go here to get the next input token }
    switch,
        { go here to eat the next character from a file }
    reswitch, { go here to digest it again }
    start_cs, { go here to start looking for a control
        sequence }
    found, { go here when a control sequence has been
        found }
    exit; { go here when the next input token has
        been got }
var k: 0 .. buf_size; { an index into buffer }
t: halfword; { a token }
cat: 0 .. max_char_code;
    { cat_code(cur_chr), usually }
c, cc: ASCII_code;
    { constituents of a possible expanded code }
d: 2 .. 3; { number of excess characters in an
    expanded code }
begin restart: cur_cs ← 0;
if state ≠ token_list then { Input from external file,
    goto restart if no input found 343 }
else { Input from token list, goto restart if end of list
    or if a parameter needs to be expanded 357 };
{ If an alignment entry has just ended, take
    appropriate action 342 };
exit: end;

```

Fig. 6: WEB code of *get_next*

to cweb format. In this process, not only the specific extensions of the Lua \TeX project were added, but \TeX was also enhanced by adding features using the ε - \TeX , pdf \TeX , and Aleph/Omega change files. These extensions are required for L \TeX and modern, convenient \TeX distributions. The conversion was done manually except for a few global regular expression replacements. I have included three versions of the *get_next* function to illustrate the differences between the traditional *TEX.WEB* by Don Knuth (Fig. 6), my code (Fig. 7), the code found as part of Lua \TeX (Fig. 8).

One can see that *web2w* has eliminated the label declarations, but left the comments in the code. Certainly this is something that could be improved in a later version of *web2w*, by moving such comments to the line where the label is defined in C.

341. Now we're ready to take the plunge into *get_next* itself. Parts of this routine are executed more often than any other instructions of \TeX .

```

void get_next(void)
    /* sets cur_cmd, cur_chr, cur_cs to next token */
{
    /* go here to get the next input token */
    /* go here to eat the next character from a file */
    /* go here to digest it again */ /* go here to
        start looking for a control sequence */
    /* go here when a control sequence has been
        found */ /* go here when the next input
        token has been got */
    uint16_t k; /* an index into buffer */
    halfword t; /* a token */
    uint8_t cat; /* cat_code(cur_chr), usually */
    ASCII_code c, cc;
    /* constituents of a possible expanded code */
    uint8_t d; /* number of excess characters in an
        expanded code */
restart: cur_cs = 0;
if (state ≠ token_list) { Input from external file,
    goto restart if no input found 343 }
else { Input from token list, goto restart if
    end of list or if a parameter needs to be
    expanded 357 };
{ If an alignment entry has just ended, take
    appropriate action 342 };
}

```

Fig. 7: *web2w* code of *get_next*

In the Lua \TeX version, these comments have disappeared together with the labels, while the comment that follows after the procedure header was converted into the text of a new section.

web2w retained the definitions of local variables, converting the subrange types to the closest possible type from *stdint.h*. For example, “*k*: 0 .. *buf_size*” was converted to “**uint16_t** *k*”. *buf_size* is defined earlier in *ctex.w* as *buf_size* = 500. Note that changing this to *buf_size* = 70000 would not force a corresponding change to **uint32_t** in the definition of *k*. Only changing the definition in *TEX.WEB* and rerunning *web2w* would propagate this change. This is an inherent difficulty of the translation from Pascal to C. In the Lua \TeX version, the local variables have disappeared and were moved to the subroutines called by *get_next*.

The module references present in the original WEB code (namely {Input from external file ...}, {Input from token list ...}, {If an alignment entry ...}), are not retained in the Lua \TeX version. Instead, Lua \TeX converts them either to function calls or expands the modules turning the module name

34. Now we're ready to take the plunge into *get_next* itself. Parts of this routine are executed more often than any other instructions of T_EX.

35. sets *cur_cmd*, *cur_chr*, *cur_cs* to next token

```
void get_next(void)
{
    RESTART: cur_cs = 0;
    if (istate ≠ token_list) { /* Input from external
        file, goto restart if no input found */
        if (!get_next_file()) goto RESTART;
    }
    else {
        if (iloc == null) {
            end_token_list();
            goto RESTART;
            /* list exhausted, resume previous level */
        }
        else if (!get_next_tokenlist()) {
            goto RESTART;
            /* parameter needs to be expanded */
        }
        /* If an alignment entry has just ended, take
         appropriate action */
        if ((cur_cmd == tab_mark_cmd ∨ cur_cmd ==
            car_ret_cmd) ∧ align_state == 0) {
            insert_vj_template();
            goto RESTART;
        }
    }
}
```

Fig. 8: LuaT_EX code of *get_next*

into a comment. Part of the problem of turning modules into subroutines is the translation of the **goto restart** statements without creating non-local **gotos**. LuaT_EX solves the problem by using boolean functions that tell the calling routine through their return values whether a **goto restart** is called for. In contrast, the automatic translation by **web2w** stays close to the original code, avoiding this problem by retaining the modules.

Conclusion

Using the **web2w** program, the T_EX source code can be converted to the cweb language, designed for the generation of C code and pretty documentation. The resulting code is very close to the original code by Knuth; its readability is surprisingly good. While manual translation is considerably more work, it offers the possibility (and temptation) of changing the code more drastically. Automatic translation can be achieved with limited effort but is less flexible and its result is by necessity closer to the original code.

The **web2w.w** program itself and the converted T_EX source code, **ctex.w**, are available on CTAN for download [8, 10]. Since **web2w** is a literate program, you can also buy it as a book [9].

References

- [1] C. O. Grosse-Lindemann and H. H. Nagel. Postlude to a PASCAL-compiler bootstrap on a DECsystem-10. *Software: Practice and Experience*, 6(1):29–42, 1976.
- [2] Taco Hoekwater. LuaT_EX says goodbye to Pascal. *TUGboat*, 30(3):136–140, 2009. <https://tug.org/TUGboat/tb30-3/tb96hoekwater-pascal.pdf>.
- [3] Kathleen Jensen and Niklaus Wirth. *PASCAL: User Manual and Report*. Springer Verlag, New York, 1975.
- [4] Donald E. Knuth. *The WEB system of structured documentation*. Stanford University, Computer Science Dept., Stanford, CA, 1983. STAN-CS-83-980. <https://ctan.org/pkg/cweb>.
- [5] Donald E. Knuth. *T_EX: The Program*. Computers & Typesetting, Volume B. Addison-Wesley, 1986.
- [6] Donald E. Knuth. *Literate Programming*. CSLI Lecture Notes Number 27. Center for the Study of Language and Information, Stanford, CA, 1992.
- [7] Donald E. Knuth and Silvio Levy. *The CWEB System of Structured Documentation*. Addison Wesley, 1994. <https://ctan.org/pkg/cweb>.
- [8] Martin Ruckert. *ctex.w: A T_EX implementation*. <http://mirrors.ctan.org/web/web2w/ctex.w>, 2017.
- [9] Martin Ruckert. *WEB to cweb*. CreateSpace, 2017. ISBN 1-548-58234-4. <https://amazon.com/dp/1548582344>.
- [10] Martin Ruckert. *web2w: Converting T_EX from WEB to cweb*. <https://ctan.org/pkg/web2w>, 2017.
- [11] *Web2C: A T_EX implementation*. <https://tug.org/web2c>.

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

dvisvgm: Generating scalable vector graphics from DVI and EPS files

Martin Gieseking

Abstract

dvisvgm is a command-line utility that converts DVI and EPS files to the XML-based vector graphics format SVG, an open standard developed by the W3C. Today, SVG is supported by many applications including text processors, graphics editors, and web browsers. Therefore, it's a convenient format with which to enrich websites and non-T_EX documents with self-contained, arbitrarily scalable T_EX output. This article gives an overview of selected features of dvisvgm and addresses some challenges faced in its development.

1 How it all started

In 2005 I was working on a wiki-based cross media publishing system called *media2mult* [3], which was supposed to produce documents in various output formats from a single source without the need to force the authors to scatter format and layout specific settings throughout the input document. Since the conversion back-end was built on the XML formatting technology XSL-FO, which at that time didn't provide sufficient math support through MathML, I needed a way to embed scalable T_EX output in the XSL-FO files. The preferred format for this task was SVG, because it is XML-based and therefore fit nicely into the other involved XML technologies; for example, the files could be post-processed easily by applying XSLT and XQuery scripts. Furthermore, SVG was decently supported by Apache's open-source XSL-FO processor FOP and the related Batik SVG toolkit.

Fortunately, two DVI to SVG converters were already available, *dvisvg* [7] by Rudolf Sabo and *dvi2svg* [2] by Adrian Frischauf. Both utilities looked promising, and created nice results from my initial test files. *dvi2svg* even supported color and hyperref specials, which was another advanced requirement for my needs. However, the main drawback for the planned document conversion engine was that both tools relied on pre-converted SVG font files derived from a selection of common T_EX fonts, notably the *Computer Modern* family. There was no simple way to process DVI files referencing arbitrary fonts supported by the T_EX ecosystem. The latter had to be generated in advance by some kind of DVI preprocessing and by extracting the glyph data from PostScript or TrueType fonts, e.g. as described in [4, pp. 272–274].

Sadly, around that time, the development of

both utilities apparently stalled, and the website of *dvi2svg* disappeared several months later. The alternative approach of creating SVG files from PDF didn't work satisfactorily either, due to the missing conversion of hyperlinks across pages inside the document, and the weak support of METAFONT-based fonts, which were embedded as bitmap fonts if no vector versions were available.

Since I had already written a couple of small DVI utilities before and therefore had working DVI and TFM readers available, I started to build a simple SVG converter on top of them. The first public release of dvisvgm was in August 2005. Since then, it's been a private free-time project and has evolved a lot over the years, largely because of wonderful feedback, detailed bug reports and interesting feature suggestions. dvisvgm is included in T_EX Live and MiK_TE_X, and is also available through MacPorts.

2 About dvisvgm and basic usage

dvisvgm is a command-line utility written in C++. It supports standard DVI files with a version identifier of 2, as well as DVI files created by pT_EX in vertical mode (version 3) and X_TE_X (versions 5 to 7).¹ The latter are also known as XDV files and are created if X_TE_X is called with option `-no-pdf`.

The basic usage of dvisvgm is straightforward and similar to other DVI drivers. If no other options are specified, it converts the first page of the given DVI file to an SVG file with the same name. If the DVI file has more than one page, the page number is appended to the base name. For example,

```
dvisvgm myfile.dvi
```

creates the file `myfile.svg` if `myfile.dvi` consists of a single page only, else `myfile-01.svg`. This was originally because the initial releases of dvisvgm could process only single DVI pages in one run; the behavior is still retained for compatibility. To select a different page or a sequence of page ranges, the option `--page` is required. It accepts a single page number or a comma-separated list of ranges. Regardless of whether any page numbers are specified multiple times, e.g. by overlapping range specification, all selected pages are converted to separate SVG files only once. The command

```
dvisvgm --page=1,3,5-9,8-10 myfile.dvi
```

is identical to

```
dvisvgm --page=1,3,5-10 myfile.dvi
```

and converts the pages 1, 3, and 5 through 10. The file names get the corresponding number suffixes as

¹ When an incompatible change in X_TE_X's XDV format is made, the DVI version identifier is increased. The recent X_TE_X revision 0.99998 creates DVI files of version 7.

--output=<pattern>	SVG file name of page 1
%f	myfile.svg
%f-%p	myfile-01.svg
newfile-%p	newfile-01.svg
%f-%4p-of-%P	myfile-0001-of-20.svg
%f-%4(p-1)	myfile-0000.svg
%f-%(P-p+1)	myfile-20.svg
..%f/svg/%3p	../myfile/svg/001.svg

Table 1: Effect of several output patterns applied to `myfile.dvi` consisting of 20 pages.

above. It's also possible to give open page ranges by omitting the start or end number:

```
dvisvgm --page=-5,10-
```

converts all pages from the beginning up to page 5, as well as page 10 and all following ones. Regardless of the number of pages converted, dvisvgm always prescans the entire DVI file in advance to collect global data, like font definitions, PostScript headers and hyperlink targets. In this way it is possible to convert selected pages correctly even if required information is located on excluded pages.

In order to change the names of the generated SVG files, `--output` can be used. It supports patterns containing the placeholders `%f`, `%p`, and `%P` which expand to the base name of the DVI file, the current physical page number, and the total number of pages in the DVI file, respectively. The command

```
dvisvgm --output=%f-%p-%P myfile
```

converts the first page of `myfile.dvi` to the SVG file `myfile-01-20.svg`, given that the DVI file contains 20 pages. The number of digits used for `%p` is adapted to that of `%P` but can be explicitly determined by a prepended number, e.g., `%4p`. Table 1 shows some further examples of specifying the naming scheme of the generated files. More details can be found in the `dvisvgm` manual page.²

Because of the lengthy text-based nature of XML documents, SVG files tend to be bigger than other vector graphics formats. To reduce the file size, the SVG standard specifies gzip- and deflate-compressed SVG files which normally use the extension `.svgz`. To create compressed files on the fly during a DVI conversion, the option `--zip` is available.

3 Font support

In contrast to PostScript and PDF, DVI provides no means to embed fonts into the file. Fonts are specified merely through their name, size, and a couple of additional parameters allowing the DVI driver to retrieve further data from the user's TeX environ-

ment. While this approach keeps DVI files compact, it also reduces their cross-platform portability and delegates significant processing to the driver. On the other hand, the requirement for a working TeX system enables full access to all font data, including those usually not embedded into PDF files. This is especially important regarding METAFONT-based fonts not available in other formats.

Since I was confronted with a wide variety of documents using a wide variety of fonts, it was important to provide dvisvgm with comprehensive font support including virtual fonts, various font encodings, CMaps, sub-font definitions, font maps, handling of glyph names and Japanese fonts which often use an extended TFM format called JFM. The proper mapping of PostScript character names, as used in Type 1 fonts, to corresponding Unicode points, required the inclusion of the Adobe Glyph List (AGL).³

Furthermore, the generated SVG files needed to be compatible with XSL-FO converters and SVG renderers, like web browsers. It turned out that each type of applications evidently focused on different aspects of the SVG standard, as some elements are not evaluated completely, leading to incorrect or incomplete visual results. To work around this, I added command-line options to alter the representation of glyphs and other graphic components in the generated SVG files as needed. The following sections cover some of the challenges involved in this area.

3.1 Vectorization of bitmap fonts

Today, many popular fonts used in (I)TeX documents are available in OpenType, TrueType or PostScript Type 1 format, greatly simplifying their conversion to SVG as they are already vectorized. This also includes many fonts originally developed with METAFONT, such as the beautiful Old German decorative initials by Yannis Haralambous. However, during the first releases of dvisvgm, I regularly stumbled over documents that could not be converted completely because they relied on fonts only available as METAFONT source. Some of these were designed by the document authors to provide special characters or little drawings.

The problem is that although METAFONT allows detailed, high-level vectorial descriptions of glyphs, it doesn't create vector but bitmap output, in the form of GF (Generic Font) files. While it's possible to utilize bitmap fonts with SVG, the results are not satisfying, and this approach would not meet our objectives. Thus, it was necessary to find a way to

² dvisvgm.sf.net/Manpage#specials

³ github.com/adobe-type-tools/agl-specification

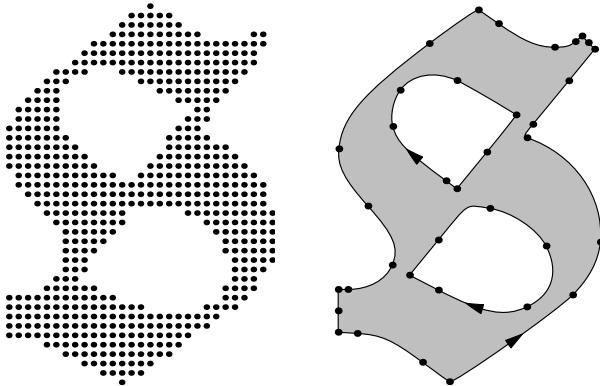


Figure 1: The Schwabacher “round s” extracted from GF font `yswab.600gf` (600 ppi) and the vectorization of the same glyph based on `yswab.2400gf`. It’s composed of three closed oriented paths, with enclosed regions filled according to the non-zero rule, i.e. areas with a winding number $\neq 0$ are considered “inside”.

vectorize the GF fonts during a DVI to SVG conversion without the need to perform this task separately in advance. Fortunately, there were already some open-source tracers available that could be incorporated into dvisvgm to do the hard work. Especially, the free *potrace* library [8] by Peter Selinger produces amazing results from monochromatic bitmaps, like the glyphs of GF fonts (see figure 1).

To avoid unintentional distortions of the generated paths, a high-resolution bitmap of the glyph is required. By default, dvisvgm calls METAFONT to create a GF font with a resolution of 2400 pixels per inch, which turned out to be a suitable choice, and runs potrace on the needed glyphs afterwards. The computed vector descriptions are then converted to SVG `glyph` or `path` elements and inserted into the SVG document tree. Furthermore, dvisvgm stores the vector data in a font cache located in the user’s home directory to avoid repeated vectorizations of the same glyphs. When subsequently converting the same DVI file again, the glyph outlines are read from the cache, drastically increasing processing speed. Information on the data currently stored in the cache can be retrieved with the option `--cache`.

By default, dvisvgm vectorizes only the glyphs actually used on the processed DVI pages. Hence, only these are added to the cache. If the document is modified so that further, currently uncached glyphs are required, METAFONT and the vectorizer are run again to create the missing data. If desired, it’s also possible to vectorize the entire GF font at once so that all its glyphs are cached instantly, with the option `--trace-all`.

While the automatic vectorization of GF fonts works pretty well, the results can’t beat the manually

optimized glyphs provided by native vector fonts. Thus, it has been implemented as a fallback routine only triggered if no vector version of a required font is available in the user’s TeX environment.

3.2 Font elements vs. graphics paths

The initial release of dvisvgm was designed to embed font data only in terms of SVG `font`, `font-face`, and `glyph` elements, as the SVG standard provided them for this very purpose. Once defined, the glyphs can be referenced by selecting the font and using the corresponding UTF-8-encoded characters inside a `text` element, as shown in the following SVG excerpt. Due to the elaborate nature of the XML syntax, only a single, relatively short `glyph` element defining a period is shown here, abridged:

```
<font id="yswab" horiz-adv-x="0">
  <font-face ascent="751" descent="249"
    font-family="yswab" units-per-em="1000"/>
  ...
  <glyph d="m149 1511-59 -59c-7 -7 -15 -14 -20
    -2315 -8174 -74h1159 59c7 7 15 14 20 231-5
    81-75 74z" glyph-name="period" unicode="."
    horiz-adv-x="306" vert-adv-y="306"/>
  ...
</font>
...
<text font-family="yswab" font-size="14.35"
  x="50" y="100">Hello.</text>
```

The advantage of this method is that the SVG file contains both the textual information and the appearance of the characters. This enables SVG viewers to provide features such as text search and copying, which works nicely with Apache’s *Squiggle* viewer, for example. Conversion tools, like the XSL-FO processor FOP, are written to maintain the text properties and can propagate them to other file formats. On the other hand, the big disadvantage is that few SVG renderers actually support `font` elements. In particular, all popular web browsers come with partial SVG support — and none of them evaluate fonts defined as shown above. Therefore, the displayed text is selectable and searchable but very likely does not look as expected (see figure 2). As stated by Daan Leijen [6], this is an irritating problem for applications like his authoring system *Madoko*, which would like to embed math formulas in terms of SVG files into HTML documents.

A workaround for this issue is to forgo font and character information and to convert the glyphs to plain graphic objects in the form of `path` elements. These are correctly processed by all SVG renderers and lead to the desired visual results. If dvisvgm is called with the option `--no-fonts`, the above



Figure 2: Screenshots of two SVG files opened in Firefox. Both were generated from the same DVI file. The left image uses `font`, the right `path` elements.

example is transformed to the following sequence of SVG elements:

```
<defs>
  ...
  <path d="m2.14 -2.171-0.85 0.85c-0.1 0.1
         -0.22 0.2 -0.29 0.3310.07 0.1111.06 1.06
         h0.0110.85 -0.85c0.1 -0.1 0.22 -0.2 0.29
         -0.331-0.07 -0.11-1.08 -1.06z" id="g2-46"/>
  ...
</defs>
...
<use x="50" y="100" xlink:href="#g2-72"/>
<use x="59.8" y="100" xlink:href="#g2-101"/>
<use x="64.3" y="100" xlink:href="#g2-108"/>
<use x="70.5" y="100" xlink:href="#g2-108"/>
<use x="71.1" y="100" xlink:href="#g2-111"/>
<use x="77.8" y="100" xlink:href="#g2-46"/>
```

Based on font parameters like the partition of the em square and the font size, all glyph descriptions are now condensed to isolated graphics path objects tagged with a unique identifier. The latter is utilized to reference the object through `use` elements in order to place instances of it at the appropriate positions. Although the resulting SVG files no longer contain textual information, the visual outcome is indistinguishable from the correctly rendered font data, while simultaneously maintaining high portability across SVG renderers.

3.3 Generating WOFF fonts

Although the conversion of glyphs to graphics paths leads to satisfying visual results, the lack of access to the text is a considerable disadvantage. Fortunately, all main web browsers come with full-featured support of WOFF, WOFF2, and TrueType fonts. The CSS rule `@font-face` allows linking the name of a font family with a font file, which may be either referenced by its name, or completely embedded into the CSS code in terms of base64-encoded data.

As of version 2, dvisvgm provides the option `--font-format` to select between several different formats. Currently, it accepts the arguments `woff`, `woff2`, `ttf`, and (the default) `svg`. Similar to the

treatment of SVG fonts, all data of the newly supported font formats is embedded into the SVG files in order to maximize portability. The alternative approach, to reference local font files already present on the user's system, would clearly significantly decrease the size of the generated files, but is avoided at present due to a couple of drawbacks. Especially, the fact that SVG relies on Unicode tables provided by the font files which don't necessarily have to cover all glyphs present in the font is a problem. If the Unicode table doesn't define a mapping for a certain glyph, it is inaccessible from the SVG document. This turns out to be the case for many displaystyle math operators or character variants defined by several fonts, like the XITS math font, for example. Thus, dvisvgm derives a new font from the original one and assigns random code points in the Unicode Private Use Area to the "hidden" glyphs. The resulting file is then embedded into the corresponding SVG file. For a future version, it might be a nice feature to create external font files containing all glyphs required for the entire DVI document and then reference them inside the various SVG files.

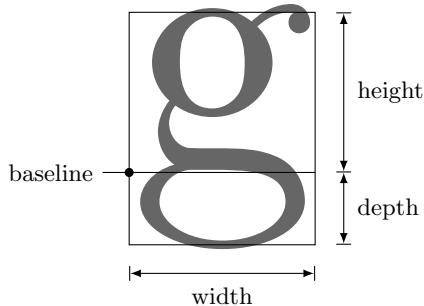
4 Bounding boxes

In contrast to DVI converters like `dvips` or `dvipdfm(x)` which are usually utilized to create self-contained final documents, the main application scenario of dvisvgm is to generate graphics files to be embedded into other documents, like web sites, EPUB or XSL-FO files. A typical example is the alignment of mathematical formulas typeset by `TEX` with the text of an HTML page. Therefore, the generated SVG graphics normally should get a minimal bounding rectangle that tightly encloses all visible parts without surrounding space so that the spacing and positioning of the graphics can be easily controlled inside the main document. Additional static margins present in the SVG file would make this more difficult. For this reason, dvisvgm computes tight bounding boxes for all converted pages. If a different bounding box is needed, though, the option `--bbox` can be used to add additional space around graphics (e.g. `--bbox=5pt`), to set an arbitrary box by specifying the coordinates of two diagonal corners, or to assign a common paper format, e.g. A4 or letter (e.g. `--bbox=letter`).

4.1 Tight text boxes

In order to compute tight bounding boxes, the converter requires information on the extents of each glyph present on the current page. The easiest way to get them is either to read the corresponding values directly from the font file or to use the width, height,

and depth values stored in a font's TFM (TeX Font Metrics) file. dvisvgm always prefers the latter if possible, because the TFM data tends to be more precise and usually leads to better results. This approach isn't perfect either, though. TFM files are primarily designed to provide TeX's algorithms with the font metrics needed to determine the optimal character positions of the processed document. The actual shapes of the characters don't matter for these computations and are in fact never seen by TeX. Furthermore, the character boxes defined by the width, height, and depth values don't have to enclose the characters' glyphs tightly. The boxes are especially allowed to be smaller so that parts of the glyphs can exceed their box as the top and bottom areas of the letter in the following example:



Obviously, this box is also somewhat wider than the enclosed glyph. Depending on the amount of divergence, the computation of the global SVG bounding box based on these values may eventually lead to visibly cropped characters and/or unwanted space at the borders of the generated graphics. That's why dvisvgm's option `--exact` was implemented some time ago. It tells the converter to trace the outlines of each glyph present on the page and to calculate their exact bounds. The path descriptions required for this task are taken from one of the available vector font files or, as a fallback, from the results of the above mentioned vectorization of METAFONT's bitmap output. While slightly more time-consuming, this approach works pretty well and helps to avoid the described text-related bounding box issues.

4.2 Aligning baselines

Another common problem that needs attention when embedding graphical L^AT_EX snippets in e.g. HTML documents is the alignment of the baselines. Since the total height of the generated graphics comprises the height and depth of the shown text, the graphics must be shifted down by the depth value in order to properly line up with the surrounding HTML text. The vertical position can be changed with CSS property `vertical-align`, but how do we get the

required depth values? Unfortunately, plain DVI files don't provide any high-level information such as typographic data. They essentially contain only the positions of single characters and rules. Thus, it's not easily possible to extract the baseline position in a reliable way.⁴ Especially, two-dimensional math formulas with characters at several vertical positions, as shown in the figure below, are difficult to analyze at the DVI level without further assistance from TeX.

A diagram of a mathematical formula $\tilde{y}_k = \frac{x^{2k}}{x+1}$. A horizontal line labeled 'baseline' has a dot at the start of the formula. A vertical double-headed arrow from the baseline to the top of the formula is labeled 'height'. A vertical double-headed arrow from the baseline to the bottom of the formula is labeled 'depth'. A horizontal double-headed arrow across the width of the formula is labeled 'width'.

One helpful tool to work around this limitation is the `preview` package by David Kastrup. Particularly, its package option `tightpage` [5, p. 4] enriches the DVI file with additional data regarding height and depth which allows computing the vertical coordinate of the baseline. dvisvgm uses this information to calculate height and depth of the previously determined tight bounding box, which usually differs from the box extents provided by the preview data due to further preview settings, such as the length `\PreviewBorder`. The resulting box values are printed to the console and can be read by third-party applications afterwards to adjust the embedded SVG graphics accordingly. This happens automatically without the need to request this information explicitly. For instance, the conversion of the unscaled above formula typeset in 10 pt size leads to the following additional output:

`width=39.02pt, height=10.43pt, depth=4.28pt,`
where the unit pt denotes TeX points ($72.27\text{pt} = 1\text{in}$). If dvisvgm should apply the original, unmodified `tightpage` extents present in the DVI file, the command-line option `--bbox=preview` can be specified. Of course, the length values reported to the console then change appropriately as well.

It's important to consider that the extraction of the `tightpage` data requires a dvisvgm binary with enabled PostScript support (see section 6) because the preview package adds the box extents in terms of PostScript specials to the DVI file. If PostScript support is disabled for some reason, dvisvgm prints a

⁴ There are some tricks to detect line breaks and the probable locations of the new starting baselines. One approach is to check the height of the DVI stack every time the virtual DVI cursor is moved by a positional operation. If the stack height underruns a certain threshold, a line break most likely occurred. While this technique works well for splitting hyperlink markings for example, it doesn't work reliably enough to derive the true baseline positions.

corresponding warning message and silently ignores the preview information and behaves as if no preview data were present.

A limitation of the current baseline computation is the restriction to unrotated single-line graphics. Graphics showing multiple lines of text are usually difficult to align with surrounding text and need special treatment not presently covered. The depth of such graphics is currently set to the depth of the lowest line, whereas everything above extends into the height part of the box.

4.3 `papersize` specials

Another way to define the size of the bounding rectangle is to add `papersize` specials to the `TeX` file, e.g. `\special{papersize=5cm,2.5cm}`, where the two comma-separated lengths denote width and height of the page. Since it's not very practical to manually enrich the documents with these commands, a couple of packages like *standalone* are available that compute the extents according to the page content and insert the specials transparently. Once present in the DVI file, `dvisvgm` can be told to evaluate the `papersize` specials and to apply the given extents as bounding box to the generated SVG files. Due to compatibility reasons with previous releases, this doesn't happen automatically but must be enabled with the option `--bbox=papersize`.

While the meaning of the `papersize` special itself is almost unambiguous and documented in the `dvips` manual, the semantics of multiple instances of the special present on the same page is not explicitly specified. Indeed, as recently discussed on the `TeX` Live mailing list, different DVI processors handle sequences of these specials differently. For example, `dvips` used to pick the first one on the page and ignore the rest, whereas `dvipdfmx` and others apply the last one—which, unsurprisingly, leads to different results. Since several popular packages, notably `hyperref` and `geometry`, insert `papersize` specials, it's likely that DVI pages often contain more than one and the user might stumble over this inconsistency at times. As of version 5.997 (2017), `dvips` got the new option `-L` to tweak this behavior. By default, it now also uses the last special, corresponding to `-L1`, whereas `-L0` restores the old behavior. `dvisvgm`'s `papersize` support became available only after this unification effort and could therefore respect the preferred semantics without breaking previous behavior. So, it also always uses the last special present on a DVI page.

A further property of `papersize` specials is their global scope. Once applied, the size settings affect not only the current but also all subsequent pages until another `papersize` is seen. Thus, if all pages

should have the same size, it's sufficient to specify the special only once at the beginning of the document.

5 Evaluation of specials

Although DVI is a very compact binary format to describe the visual layout of a typeset document, it is rather limited regarding the types of objects that can be placed on a page—only characters and solid rectangles are supported natively. Color, rotated text, graphics, hyperlinks and other features to enrich the documents are not covered by the format specification. To handle this, the DVI standard provides an operation called *xxx* which corresponds to `TeX`'s `\special` command. It has no inherent semantics but merely holds the expanded, usually textual, argument of a `\special` command passed from the `TeX` document to the DVI file. Since it also doesn't affect the state of the DVI engine, each DVI driver is allowed to decide whether to evaluate any of the *xxx* operations or to ignore them altogether. Based on this mechanism, authors of `LATeX` packages and DVI processors can specify various special commands with defined syntax and semantics to enhance the capabilities of plain DVI documents, as already seen in the previous section about `papersize` specials.

Over the decades of `TeX` use, many sets of specials have been introduced. Some are well established and used by various packages. These include, among others, the color, `hyperref`, and PostScript specials. The recent version of `dvisvgm` supports these, as well as PDF font map specials, `tpic` specials, and the line drawing statements of the `emTeX` specials. To check the availability of a certain special handler in the current version of `dvisvgm`, the option `--list-specials` can be used. It prints a short summary of the supported special sets. It's also possible to ignore some or all specials during a DVI conversion with option `--no-specials`; this accepts an optional list of comma-separated handler names, which are identical to those listed by `--list-specials`, in order to disable only selected specials. For example, `--no-specials=color,html` disables the processing of all color and `hyperref` specials.

While a detailed description of all supported special commands is beyond the scope of this article, the following sections give some brief information on the `hyperref` and `dvisvgm` specials which might be helpful to know. Aspects of the PostScript handler are addressed in the subsequent section 6.

5.1 `hyperref` specials

The `hyperref` package provides commands to add hyperlinks to a `LATeX` document. Depending on the selected driver, it produces code for `dvips`, `dvipdfmx`,

--linkmark=<style>	visual result
box (default)	linked text
box:blue	linked text
line	linked text
line:#00ff00	linked text
yellow	linked text
yellow:violet	linked text
none	linked text

Table 2: Examples showing the visual effect of --linkmark on hyperlinked texts.

X_ET_EX, or any of the many other supported targets. In order to create hyperlink specials understood by dvisvgm, hyperref must be told to emit “HyperT_EX” specials, with the package option `hypertex`.

By default, a linked area in the SVG file is highlighted by a box drawn around it in the currently selected color. On the request of several users, the option `--linkmark` was added to allow changing this behavior. It requires an argument determining the style of the marking. While `box` is the default, argument `line` underlines the clickable area rather than framing it, and `none` suppresses any visual highlighting of hyperlinks completely. A dvips color name or hexadecimal RGB value appended to these styles and separated by a colon, assigns a static color to the box or line. Finally, a style argument of the form `color1:color2` leads to a box filled with `color1` and framed with `color2`. Table 2 shows some examples to give an idea of the effect of the style arguments.

5.2 dvisvgm specials

Besides the mentioned sets of special commands, dvisvgm also provides some of its own to allow authors of L_AT_EX packages to insert additional SVG fragments into the generated files and to interact with the computation of the bounding box. Their general syntax looks like this:

```
\special{dvisvgm}{<cmd> <params>}
```

The `cmd` denotes the command name and `params` the corresponding parameters. For the sake of simplicity, only the text after the colon is mentioned when referring to dvisvgm specials herein.

The command `raw` followed by arbitrary text appends the text to the group element representing the current page. The sibling command `rawdef` does almost the same but appends the text to the initial `defs` element present at the beginning of the SVG file. Both specials are allowed to insert any string and thus can contain XML metacharacters, such as angle brackets, e.g.:

```
raw <circle cx="?x" cy="?y" r="5"/>.
```

The macros `{?x}` and `{?y}` expand to the *x* and *y* coordinate of the current DVI position in the “big point” units ($72 \text{ bp} = 1 \text{ in}$) required in SVG files. The entire character string is then copied to a literal text node of the SVG tree and not evaluated further. Therefore, it’s crucial to ensure that the insertions don’t break the validity of the resulting SVG document, especially if multiple `raw` or `rawdef` commands are used to assemble complex element structures.

Another aspect to take care of regarding raw insertions is the adaptation of the bounding box. As outlined in section 4, dvisvgm computes a tight bounding box for the generated SVG graphics by default. Graphical or textual elements inserted via the raw commands are not taken into account. As a consequence, the bounding box may be too small, so that some parts of the graphic lie outside the viewport. To work around this, dvisvgm offers a special that allows for intervening in the calculation of the bounding rectangle. The command

```
bbox <width> <height>
```

updates the bounding box so that a virtual rectangle of the given width and height and located at the current DVI position will be fully enclosed. It’s also possible to append an optional `depth` parameter to the command:

```
bbox <width> <height> <depth>
```

This encloses another rectangle of the same width but with the negative height `depth`. At present, the dimensions must be given as plain floating point numbers in T_EX pt units without a unit specifier. In a future release, it will be possible to use the various common length units to ease usage of this command. For example, to update the bounding box for the above raw circle element, the two successive dvisvgm specials `bbox 5 5 5` and `bbox -5 5 5` can be used.

In addition to these relative bounding box specials, two absolute variants are supported, which are only briefly mentioned here. More details about them can be found on the manual page.

```
bbox abs <x1> <y1> <x2> <y2>
```

```
bbox fix <x1> <y1> <x2> <y2>
```

The first variant encloses a virtual rectangle given by the coordinates (x_1, y_1) and (x_2, y_2) of two diagonal corners, whereas the second one sets the final coordinates of the SVG bounding box, which will not be changed or reset afterwards.

6 PostScript support

One of the biggest enhancements of the DVI format was certainly the introduction of PostScript specials and their processing by Tomas Rokicki’s dvips. Besides placing advanced drawings in T_EX documents,

it supports injecting code between DVI commands, allowing for the implementation of text transformations, coloring and much more. While dvips can copy the code of the PostScript specials almost literally to the generated files and delegate their processing to the PostScript interpreter, DVI drivers targeting a different output format have to evaluate it somehow.

The implementation of a full-featured PostScript interpreter was certainly out of the scope of dvisvgm. However, I wanted the utility to be able to properly convert as many DVI files as possible, ideally including ones created using PSTricks or TikZ. The most straightforward approach to achieve this was to delegate the complex processing of PostScript code to the free PostScript interpreter *Ghostscript* and let it emit a reduced set of easily parseable statements that dvisvgm could evaluate. This turned out to work reasonably well, especially as a fair amount of PostScript code can completely be processed by Ghostscript without the need to worry about the involved operations. Only a relatively small set of operators that affect the graphics state must be overridden and forwarded to dvisvgm in order to create appropriate SVG components or to update drawing properties.

In contrast to the other programming libraries dvisvgm relies on and which are directly linked into the binary, the Ghostscript library (`libgs`) can be tied to dvisvgm in two different ways. Besides disabling PostScript support completely, it's possible to either link to the Ghostscript library directly, or to load it dynamically at runtime. In the first case, PostScript support is always enabled, while in the second one it depends on the accessibility of the Ghostscript library on the user's system. If `libgs` can't be found or accessed for some reason, dvisvgm prints a warning message and disables the processing of PostScript specials, which of course will likely lead to inaccurate conversion results. To help dvisvgm locate the library, the option `--libgs` or environment variable `LIBGS` can be used, e.g. to specify the absolute path of the correct file. More detailed information on this topic can be found on the FAQ page of the project website.⁵

Although dvisvgm can properly convert a fair amount of PostScript code, there are still some operators and features it does not support yet. These include all bitmap-related operations as well as linear, radial, and function-based shading fills. Furthermore, text output triggered by PostScript code is always converted to SVG `path` elements similar to those described in section 3.2. The differentiated handling of

fonts including the conversion to WOFF only works in conjunction with DVI font definitions.

6.1 Handling clipping path intersections

In order to restrict the area where drawing commands lead to visible results, SVG allows the definition of *clipping paths*. Every clipping path is defined by a set of closed vector paths consisting of an arbitrary number of straight and curved line segments. The regions enclosed by these paths define the visible area, i.e. after applying a clipping path, only those portions of the subsequently drawn graphics that fall inside the enclosed area are visible, while everything else is discarded. Clipping is a basic functionality of computer graphics and supported by various formats and languages, like PostScript, Asymptote, METAPost, and TikZ. So why is it mentioned here? Because one variant of defining clipping paths in SVG may lead to unpredictable, flawed visual results due to absent or incomplete support in SVG renderers.

Besides defining the clipping path explicitly, which is nicely supported by almost all renderers I know of, it's also possible to tell the SVG renderer to compute the intersection of two or more paths and restrict the subsequent drawing actions to the resulting area. The following example defines a lens-shaped path called `lens` by combining two arcs of 90 degrees. The result is assigned to clipping path `clip1`. The second clipping path `clip2` reuses path `lens` but rotated by 90 degrees clockwise around its center.

```
<clipPath id="clip1">
  <path id="lens" d=
    M 0 0
    A 50 50 0 0 1 50 50
    A 50 50 0 0 1 0 0 Z"/>
</clipPath>
<clipPath id="clip2" clip-path="url(#clip1)">
  <use xlink:href="#lens"
    transform="rotate(90,25,25)"/>
</clipPath>
```

The crucial part of this definition is the `clip-path` attribute, which restricts the drawing area of `clip2` to the interior of `clip1` so that the resulting clipping region leads to a curved square, as shown in figure 3.

Graphic elements restricted to `clip2`, like the following rectangle, are now supposed to be clipped at the border of this square.

```
<rect x="17" y="0" width="16" height="50"
  clip-path="url(#clip2)"/>
```

Unfortunately, this isn't the case with all SVG renderers.⁶ Since successive calls of the PostScript operators `clip` and `eoclip` cause consecutive path intersections, which dvisvgm translates to `clipPath`

⁵ dvisvgm.sf.net/FAQ

⁶ Examples can be seen at dvisvgm.sf.net/Clipping.

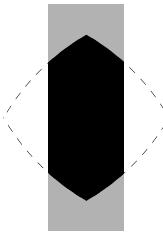
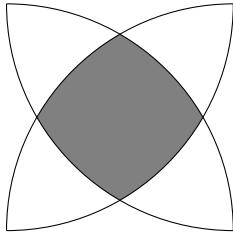


Figure 3: Intersection of two lens-shaped paths (left), and a rectangle clipped on the resulting area.

elements with `clip-path` attributes by default, the generated SVG files are not portable either. In order to prevent the creation of these, the `--clipjoin` option was added some time ago. It tells dvisvgm to compute the path intersections itself with the help of Angus Johnson's great *Clipper* library⁷, which provides an implementation of the Vatti polygon clipping algorithm. For this purpose, dvisvgm approximates all clipping paths by polygons, runs the Vatti algorithm on them to compute the boundaries of the intersection areas, and reconstructs the curved segments of the resulting paths afterwards. In this way, we usually get a compact yet smoothly approximated outline of the final clipping paths. The application of option `--clipjoin` to clipping path `clip2` of the above example leads to the following self-contained path definition composed of four cubic Bézier curve segments:

```
<clipPath id="clip2">
  <path d="
    M 43.2 25
    C 38.8 32.5 32.5 38.8 25 43.2
    C 17.5 38.8 11.2 32.5 6.8 25
    C 11.2 17.5 17.5 11.2 25 6.8
    C 32.5 11.2 38.8 17.5 43.2 25 Z"/>
</clipPath>
```

6.2 Approximation of gradient fills

One of the more powerful and impressive PostScript features is the advanced support of various shading algorithms to fill a region with smooth transitions of colors in several color spaces. These algorithms include Gouraud-shaded triangle meshes, tensor-product patch meshes, and flexible function-based shadings, as well as linear and radial gradients. The current SVG standard 1.1 provides elements to specify gradient fills too but they are limited to the last two mentioned above, and are furthermore somewhat less flexible than the PostScript equivalents. Therefore, it's not possible to map arbitrary gradient definitions present in EPS files or PostScript specials to plain SVG gradient elements. In order to nonethe-

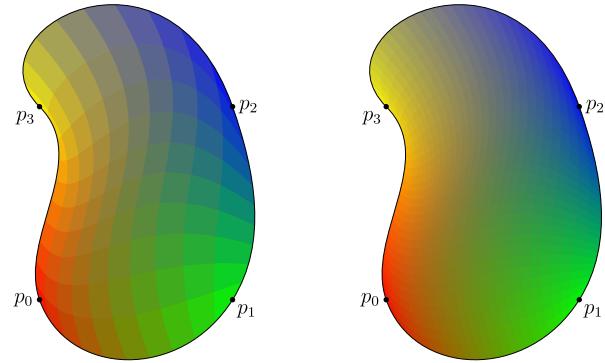


Figure 4: Approximation of tensor product shading using a grid of 10×10 and 30×30 color segments, respectively.

less convert a subset of them, dvisvgm approximates color gradients by filling the specified area with small monochromatic segments as shown in figure 4.

Each segment gets the average color of the covered area according to the selected gradient type and color space. The maximum number of segments created per column or row can be changed by option `--grad-segments`. Greater values certainly lead to better approximations, but concurrently increase the computation time, the size of the SVG file, and, perhaps most important, the effort required to render the file. To slightly counteract this drawback, dvisvgm reduces the level of detail if the extent of the segments falls below a certain limit. In case of tensor-product patches, the segments are usually delimited by four cubic Bézier curves and will then be simplified to quadrilaterals. The limit at which this simplification takes place can be set by option `--grad-simplify`.

An issue that can occur in conjunction with gradient fills is the phenomenon of visible gaps between adjacent segments, even though they should touch seamlessly according to their coordinates. This effect results from the anti-aliasing applied by most SVG renderers in order to produce smooth segment contours which usually takes not only the foreground but also the background color into account. Therefore, the background color becomes visible at the joints of the segments. If desired, the option `--grad-overlap` can be used to prevent this effect. It tells dvisvgm to create bigger, overlapping segments that extend into the region of their right and bottom neighbors. Since the latter are drawn on top of the overlapping parts, which now cover the former joint lines, the visible size of all segments remains unchanged. In this manner we get visual results similar to those shown in figure 4.

⁷ angusj.com/delphi/clipper.php

6.3 Converting EPS files to SVG

Besides the family of special commands provided to embed literal PostScript code directly into DVI files, dvips also introduced a special called `psfile`. Its purpose is to reference an external PS or EPS file and insert its content, possibly after some processing, at the current location of the document. The L^AT_EX command `\includegraphics` from the *graphicx* package, for instance, produces a `psfile` special if the dvips driver is selected. Also, the vector graphics language *Asymptote* [1] uses this special in its intermediate DVI files to combine the graphical and typeset components of the resulting drawings. Because of these major application areas, it was important to make dvisvgm capable of processing `psfile` specials, in order to cover a broader range of documents.

Since the technical details of the command are probably not of much interest for general users, they are not discussed here in more depth. However, one nice bonus feature that was technically available instantly after finishing the implementation of the `psfile` handler can be mentioned. Due to the handler being able to process separate files, it seemed natural to make this functionality available through the command-line interface and so provide an EPS to SVG converter. Little additional code was required to realize this. Thus, as of version 1.2, dvisvgm offers option `--eps` which tells the converter not to expect a DVI but an EPS input file and to convert it to SVG. For example,

```
dvisvgm --eps myfile
```

produces the SVG file `myfile.svg` from `myfile.eps`. This is implemented by creating a single `psfile` special called together with the bounding box information given in the EPS file's DSC header. To do the conversion, only the PostScript handler is required, with none of the DVI-related routines and associated functionality, like font and other special processing.

Since there are already some standalone utilities like ImageMagick and Inkscape available that can do this, dvisvgm's EPS to SVG functionality is probably needed less frequently but might nonetheless be beneficial for the T_EX community.

7 Acknowledgments

I would like to thank Karl Berry, Mojca Miklavec, and, posthumously, Peter Breitenlohner for their invaluable work to make dvisvgm available in T_EX Live and help in tracking down several issues. Also,

thank you to Khaled Hosny for implementing the command-line option `--no-merge` and for providing a Python port of my formerly XSLT-based helper script *opt2cpp*. I furthermore appreciate the amazing work of John Bowman and Till Tantau who added support of dvisvgm to Asymptote and TikZ/PGF, respectively.

There are many more people whom I can't list here individually but who helped enormously to improve the program by reporting bugs, providing code, and sending feature suggestions. Thank you very much to all of you.

References

- [1] John Bowman. Asymptote: Interactive T_EX-aware 3D vector graphics. *TUGboat*, 31(2):203–205, 2010. tug.org/TUGboat/tb31-2/tb98bowman.pdf.
- [2] Adrian Frischauf and Paul Libbrecht. dvi2svg: Using L^AT_EX layout on the web. *TUGboat*, 27(2):197–201, 2006. tug.org/TUGboat/tb27-2/tb87frischauf.pdf.
- [3] Martin Giesecking and Oliver Vornberger. media2mult: A wiki-based authoring tool for collaborative development of multimodal documents. In Miguel Baptista Nunes and Maggie McPherson, editors, *Proceedings of the IADIS International Conference on e-Learning*, pages 295–303, Amsterdam, Netherlands, 2008.
- [4] Michel Goossens and Vesa Sivunen. L^AT_EX, SVG, Fonts. *TUGboat: The Communications of the T_EX Users Group*, 22(4):269–280, 2001. tug.org/TUGboat/tb22-4/tb72goos.pdf.
- [5] David Kastrup. The `preview` package for L^AT_EX. ctan.org/pkg/preview, April 2017.
- [6] Daan Leijen. Rendering mathematics for the web using Madoko. In Robert Sablatnig and Tamir Hassan, editors, *Proceedings of the 2016 ACM Symposium on Document Engineering*, pages 111–114, Vienna, Austria, 2016. www.microsoft.com/en-us/research/wp-content/uploads/2016/08/doceng16.pdf.
- [7] Rudolf Sabo. DVISVG. Master's thesis, Masarykova Univerzita, Brno, Czech Republic, 2004. dvisvg.sourceforge.net/dipl.pdf.
- [8] Peter Selinger. Potrace: A polygon-based tracing algorithm. potrace.sourceforge.net, 2003.

◊ Martin Giesecking
University of Osnabrück
Heger-Tor-Wall 12
49074 Osnabrück, Germany
`martin dot giesecking (at) uos dot de`

Tricky fences

Hans Hagen

Occasionally one of my colleagues notices some sub-optimal rendering and asks me to have a look at it. Now, one can argue about “what is right” and indeed there is not always a best answer to it. Such questions can even be a nuisance; let’s think of the following scenario. You have a project where \TeX is practically the only solution. Let it be an XML rendering project, which means that there are some boundary conditions. Speaking in 2017 we find that in most cases a project starts out with the assumption that everything is possible.

Often such a project starts with a folio in mind and therefore by decent tagging to match the educational and esthetic design. When rendering is mostly automatic and concerns too many (variants) to check all rendering, some safeguards are used (an example will be given below). Then different authors, editors and designers come into play and their expectations, also about what is best, often conflict. Add to that rendering for the web, and devices and additional limitations show up: features get dropped and even more cases need to be compensated (the quality rules for paper are often much higher). But, all that defeats the earlier attempts to do well because suddenly it has to match the lesser format. This in turn makes investing in improving rendering very inefficient (read: a bottomless pit because it never gets paid and there is no way to gain back the investment). Quite often it is spacing that triggers discussions and questions what rendering is best. And inconsistency dominates these questions.

So, in case you wonder why I bother with subtle aspects of rendering as discussed below, the answer is that it is not so much professional demand but users (like my colleagues or those on the mailing lists) that make me look into it and often something that looks trivial takes days to sort out (even for someone who knows his way around the macro language, fonts and the inner working of the engine). And one can be sure that more cases will pop up.

All this being said, let’s move on to a recent example. In Con \TeX we support MathML although in practice we’re forced to a mix of that standard and ASCIIMATH. When we’re lucky, we even get a mix with good old \TeX -encoded math. One problem with an automated flow and processing (other than raw \TeX) is that one can get anything and therefore we need to play safe. This means for instance that you can get input like this:

$f(x) + f(1/x)$

or in more structured \TeX speak:

$\$f(x) + f(\frac{1}{x})\$$

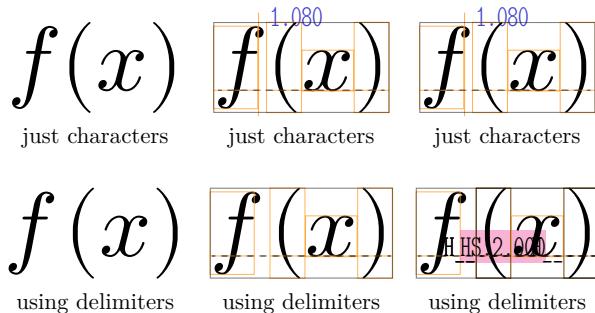
Using \TeX Gyre Pagella, this renders as: $f(x) + f(\frac{1}{x})$, and when seeing this a \TeX user will revert to:

$\$f(x) + f(\left(\frac{1}{x}\right)\right)\$$

which gives: $f(x) + f\left(\frac{1}{x}\right)$. So, in order to be robust we can always use the $\backslash left$ and $\backslash right$ commands, can’t we?

$\$f(x) + f(\left.x\right)\$$

which indeed gives $f(x) + f(x)$, but let’s blow up this result a bit showing some additional tracing from left to right, now in Latin Modern:



When we visualize the glyphs and kerns we see that there’s a space instead of a kern when we use delimiters. This is because the delimited sequence is processed as a subformula and injected as a so-called inner object and as such gets spaced according to the ordinal (for the f) and inner (“fenced” with delimiters x) spacing rules. Such a difference normally will go unnoticed but as we mentioned authors, editors and designers being involved, there’s a good chance that at some point one will magnify a PDF preview and suddenly notice that the difference between the f and $($ is a bit on the large side for simple unstacked cases, something that in print is likely to go unnoticed. So, even when we don’t know how to solve this, we do need to have an answer ready.

When I was confronted by this example of rendering I started wondering if there was a way out. It makes no sense to hard code a negative space before a fenced subformula because sometimes you don’t want that, especially not when there’s nothing before it. So, after some messing around I decided to have a look at the engine instead. I wondered if we could just give the non-scaled fence case the same treatment as the character sequence.

Unfortunately here we run into the somewhat complex way the rendering takes place. Keep in mind that it is quite natural from the perspective of \TeX because normally a user will explicitly use $\backslash left$ and $\backslash right$ as needed, while in our case the

fact that we automate and therefore want a generic solution interferes (as usual in such cases).

Once read in the sequence $f(x)$ can be represented as a list:

```
list = {
  {
    id = "noad", subtype = "ord", nucleus = {
      {
        id = "mathchar", fam = 0, char = "U+00066",
      },
    },
  },
  {
    id = "noad", subtype = "open", nucleus = {
      {
        id = "mathchar", fam = 0, char = "U+00028",
      },
    },
  },
  {
    id = "noad", subtype = "ord", nucleus = {
      {
        id = "mathchar", fam = 0, char = "U+00078",
      },
    },
  },
  {
    id = "noad", subtype = "close", nucleus = {
      {
        id = "mathchar", fam = 0, char = "U+00029",
      },
    },
  },
}
```

The sequence $f \left(x \right)$ is also a list but now it is a tree (we leave out some unset keys):

```
list = {
  {
    id = "noad", subtype = "ord", nucleus = {
      { id = "mathchar", fam = 0,
        char = "U+00066", },
    },
  },
  {
    id = "noad", subtype = "inner", nucleus = {
      {
        id = "submlist", head = {
          {
            id = "fence", subtype = "left",
            delim = { { id = "delim", small_fam = 0,
              small_char = "U+00028", },
            },
          },
        },
        {
          id = "noad", subtype = "ord",
          nucleus = { { id = "mathchar", fam = 0,
            char = "U+00078", },
          },
        },
      },
    },
  },
}
```

```
},
{
  id = "fence", subtype = "right",
  delim = { { id = "delim", small_fam = 0,
    small_char = "U+00029", },
  },
},
},
},
},
}
```

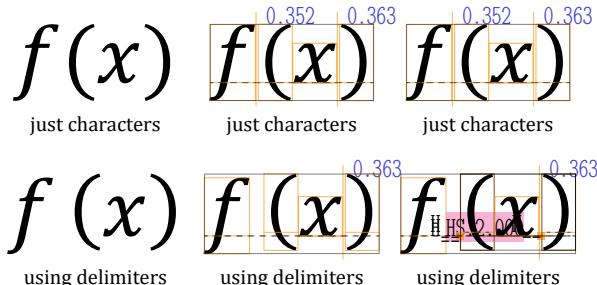
So, the formula $f(x)$ is just four characters and stays that way, but with some inter-character spacing applied according to the rules of TeX math. The sequence $f \left(x \right)$ however becomes two components: the f is an ordinal noad,¹ and $\left(x \right)$ becomes an inner noad with a list as a nucleus, which gets processed independently. The way the code is written this is what (roughly) happens:

- A formula starts; normally this is triggered by one or two dollar signs.
- The f becomes an ordinal noad and TeX goes on.
- A fence is seen with a left delimiter and an inner noad is injected.
- That noad has a sub-math list that takes the left delimiter up to a matching right one.
- When all is scanned a routine is called that turns a list of math noads into a list of nodes.
- So, we start at the beginning, the ordinal f .
- Before moving on a check happens if this character needs to be kerned with another (but here we have an ordinal–inner combination).
- Then we encounter the subformula (including fences) which triggers a nested call to the math typesetter.
- The result eventually gets packaged into a hlist and we're back one level up (here after the ordinal f).
- Processing a list happens in two passes and, to cut it short, it's the second pass that deals with choosing fences and spacing.
- Each time when a (sub)list is processed a second pass over that list happens.
- So, now TeX will inject the right spaces between pairs of noads.
- In our case that is between an ordinal and an inner noad, which is quite different from a sequence of ordinals.

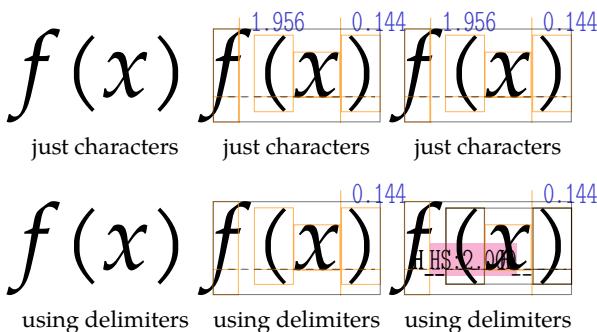
¹ Noads are the mathematical building blocks. Eventually they become nodes, the building blocks of paragraphs and boxed material.

It's these fences that demand a two-pass approach because we need to know the height and depth of the subformula. Anyway, do you see the complication? In our inner formula the fences are not scaled, but this is not communicated back in the sense that the inner noad can become an ordinal one, as in the simple $f()$ pair. The information is not only lost, it is not even considered useful and the only way to somehow bubble it up in the processing so that it can be used in the spacing requires an extension. And even then we have a problem: the kerning that we see between f (is also lost. It must be noted that this kerning is optional and triggered by setting `\mathitalicsmode=1`. One reason for this is that fonts approach italic correction differently, and cheat with the combination of natural width and italic correction.

Now, because such a workaround is definitely conflicting with the inner workings of \TeX , our experimenting demands another variable be created: `\mathdelimitersmode`. It might be a prelude to more manipulations but for now we stick to this one case. How messy it really is can be demonstrated when we render our example with Cambria.



If you look closely you will notice that the parenthesis are moved up a bit. Also notice the more accurate bounding boxes. Just to be sure we also show Pagella:

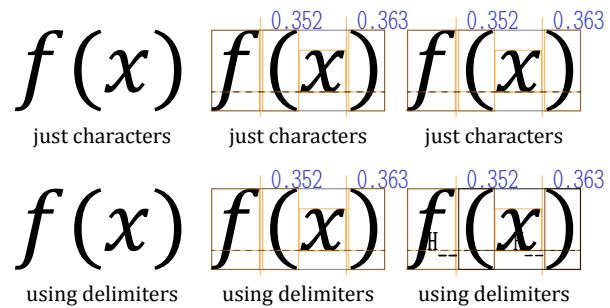


When we really want the unscaled variant to be somewhat compatible with the fenced one we now need to take into account:

- the optional axis-and-height/depth related shift of the fence (bit 1)

- the optional kern between characters (bit 2)
- the optional space between math objects (bit 4)

Each option can be set (which is handy for testing) but here we will set them all, so, when `\mathdelimitersmode=7`, we want cambria to come out as follows:



When this mode is set the following happens:

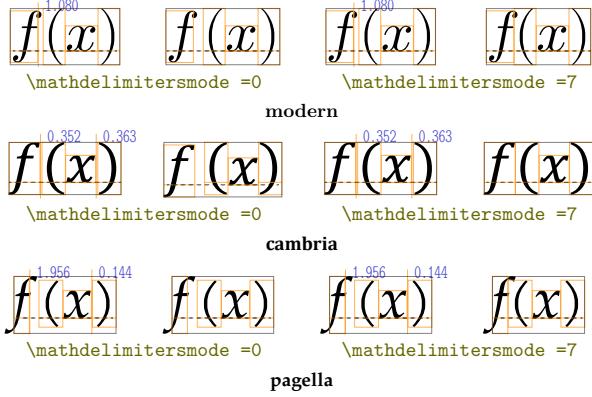
- We keep track of the scaling and when we use the normal size this is registered in the noad (we had space in the data structure for that).
- This information is picked up by the caller of the routine that does the subformula and stored in the (parent) inner noad (again, we had space for that).
- Kerns between a character (ordinal) and subformula (inner) are kept, which can be bad for other cases but probably less than what we try to solve here.
- When the fences are unscaled the inner property temporarily becomes an ordinal one when we apply the inter-noad spacing.

Hopefully this is good enough but anything more fancy would demand drastic changes in one of the most sensitive mechanisms of \TeX . It might not always work out right, so for now I consider it an experiment, which means that it can be kept around, rejected or improved.

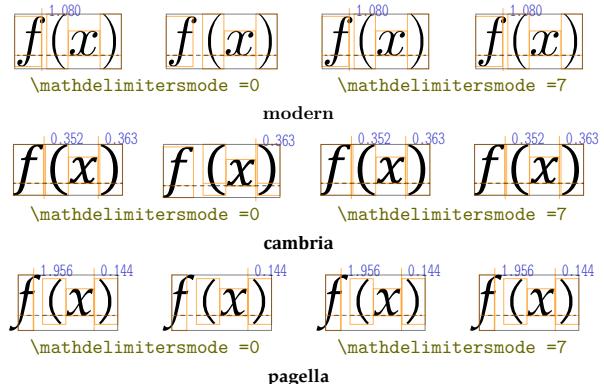
In case one wonders if such an extension is truly needed, one should also take into account that automated typesetting (also of math) is probably one of the areas where \TeX can shine for a while. And while we can deal with much by using Lua, this is one of the cases where the interwoven and integrated parsing, converting and rendering of the math machinery makes it hard. It also fits into a further opening up of the inner working by modes.

Another objection to such a solution can be that we should not alter the engine too much. However, fences already are an exception and treated specially (tests and jumps in the program) so adding this fits reasonably well into that part of the design.

In the following examples we demonstrate the results for Latin Modern, Cambria and Pagella when `\mathdelimitersmode` is set to zero or one. First we show the case where `\mathitalicsmode` is disabled:



When we enable `\mathitalicsmode` we get:



So is this all worth the effort? I don't know, but at least I got the picture and hopefully now you have too. It might also lead to some more modes in future versions of LuaTeX.

In ConTeXt, a regular document can specify `\setupmathfences [method=auto]`, but in MathML or ASCIIMATH this feature is enabled by default (so that we can test it).

We end with a summary of all the modes (assuming italics mode is enabled) in the table below.

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

$f(x)$	$f(x)$													
$f(x)$	$f(x)$													
ns	it	ns it	or	ns or	it or	ns it or	modern							
$f(x)$	$f(x)$													
$f(x)$	$f(x)$													
ns	it	ns it	or	ns or	it or	ns it or	cambria							
$f(x)$	$f(x)$													
$f(x)$	$f(x)$													
ns	it	ns it	or	ns or	it or	ns it or	pagella							

Testing indexes: testidx.sty

Nicola L. C. Talbot

Abstract

The `testidx` package [10] provides a simple method of generating a test document with a multi-paged index for testing purposes. The dummy text and index produced is designed to replicate problems commonly encountered in real documents.

The words and phrases indexed cover the basic Latin set A(a), …, Z(z) and some extended Latin characters, such as Ø(ø), Æ(æ), œ(œ), Å(å), Þ(þ) and Ł(ł), to test the indexing application’s ability to sort according to various Latin alphabets (such as Swedish or Icelandic). Version 1.1 also includes some words starting with digraphs, Dd(dd), Dz(dz), Ff(ff), IJ(ij), Ll(lł), Ly(ły), Ng(ng), and a trigraph Dzs(dzs), to test alphabets where these are considered separate letters (such as Welsh, Dutch or Hungarian).

There are also some numbers and symbols indexed that don’t have a natural word order.

1 Introduction

There are a number of problems that can occur when generating an index using L^AT_EX. These may relate to the index style (`\printindex`), the way the indexing information is written to an external file (`\index`) or the way the indexing application (such as `xindy` or `makeindex`) performs. A large document may have a complicated and slow build process, which can be frustrating when making minor adjustments to the index layout. The `testidx` package provides a way to create a test document that can be used to enhance the required style. Section 5 shows how the sample text can be extended to include tests for other languages or scripts.

The simplest test document is:

```
\documentclass{article}
\usepackage{makeidx}
\usepackage{testidx}
\makeindex
\begin{document}
\testidx
\printindex
\end{document}
```

Version 1.1 of `testidx` comes with the supplementary package `testidx-glossaries`, which uses the interface provided by the `glossaries` package [9] instead of testing `\index` and `\printindex`. In this case, the simplest test document is:

```
\documentclass{article}
\usepackage{testidx-glossaries}
\testidxmakegloss
\begin{document}
```

```
\testidx
\tstidxprintglossaries
\end{document}
```

2 Intentional issues

The dummy text is designed to introduce issues that your style or build process may need to guard against. These allow you to test the document style, the way the indexing information is written to the external file, and the way the indexing application processes that information.

2.1 Stylistic issues

The style issues are those which need addressing through the use of L^AT_EX code within the document itself, or in the class or package that deals with the index style, or within a style file or module used by the indexing application which controls the L^AT_EX code that’s written to the output file. The test document should load the appropriate document class and indexing package to match your real document.

2.1.1 Page breaking

There are enough entries for the index to span multiple pages. If you have letter group headings in your index style there’s a good chance that there will be at least one instance of a page or column break occurring between a heading and the first entry of that letter group. There’s also a chance that a break will also occur between a main entry and the first of its sub-entries.

This does, of course, depend on the font and page dimensions. You may need to adjust the geometry to cause an unwanted break before experimenting with adjusting the style to prohibit it.

2.1.2 Headers and footers

Since the index spans multiple pages, it’s possible to test the headers and footers for the first page of the index as well as subsequent even and odd pages. This is useful if the header or footer content needs to vary and you need to check that this is done correctly.

2.1.3 Line breaking

The index contains a mixture of single words, compound words, phrases, names, places and titles. This means that some of the entries are quite wide, which can cause line breaking problems in narrow columns.

2.1.4 Whatsits

Some of the entries are indexed immediately before the term, for example

```
\index{page break}page break
```

and some are indexed immediately after the term, for example

```
paragraph\index{paragraph}
```

The whatsit introduced by `\index` can cause problems. This is most noticeable in an example equation where the indexing interferes with the limits of a summation. In practice, the `\index` would need to be moved to a more suitable location, but the example provided by the dummy text helps to highlight the problem.

2.2 Index recording issues

The way that indexing typically works is to write the entry data (using `\index`) to an external file that's then input and processed by the indexing application. This write operation can sometimes go wrong causing incorrect information to be written to the external file. (There's no test for incorrect syntax within the argument of `\index`. It's assumed you know how to correctly index entries. The tests here are for the underlying operation of `\index`.)

The `glossaries` package uses a similar method but instead of using `\index`, the file write instruction is internally performed by commands like `\gls` and `\glsadd`.

2.2.1 Page breaking

The dummy text has some long paragraphs with indexing scattered about them. This increases the chance of a page break occurring mid-paragraph (although again it depends on the font and page dimensions). `TeX`'s asynchronous output routine can cause page numbers to go awry, and this provides a useful way to check that the page number is written correctly to the external file.

2.2.2 Extended Latin characters

The indexed entries include terms that contain non-ASCII letters (either through accent commands like `\`` or using UTF-8 characters). The UTF-8 encoding isn't an issue for the modern `XELaTEX` or `LuaLaTEX` engines, but it is a problem for the older `LaTEX` formats. If your engine doesn't natively support UTF-8 and you have characters outside the basic Latin set, then this is something that needs to be tested. The `testidx` package has four modes to test this, depending on your set-up.

1. ASCII with `LaTEX` commands stripped.
(‘Bare ASCII’)

This mode is triggered through the use of `LaTEX` with `testidx`'s `stripaccents` option and without the `inputenc` package [2]. This option is the default, so the first test example above is in this mode. This mode emulates doing, for example:

```
\index{elite@\`elite}
```

So if this is the way that you're indexing words in your real document, this is the mode you need in the test document.

2. Unmodified ASCII. (‘ASCII Accents’)

This mode is triggered by running `LaTEX` with `testidx`'s `nostripaccents` option and without the `inputenc` package. This mode emulates doing

```
\index{\`elite@\`elite}
```

Use this mode if in your real document you are simply doing, for example, `\index{\`elite}`.

3. Active UTF-8.

This mode is triggered if the `inputenc` package with the `utf8` option is loaded and `testidx` is loaded afterwards with the `nosanitize` option. This emulates doing

```
\index{\'elite}
```

Since the `inputenc` package makes the first octet of a UTF-8 character active, this causes the entry to be expanded as it's written to the index file, so that it appears as:

```
\IeC {\`elite}
```

Use this mode if in your real document you are doing, for example, `\index{\'elite}` and you want to test how it's written to the index file. (This mode is the default when using `XELaTEX` or `LuaLaTEX`, but the characters aren't active, so it's much the same as the next mode.)

4. Sanitized UTF-8.

The three modes listed above are for emulating different `\index` usage. This last mode really belongs in the next section as it's provided for testing the indexing application's UTF-8 support, but is included in this list for completeness.

This mode is triggered if `inputenc` is loaded with the `utf8` option and `testidx` is loaded afterwards with the `sanitize` option. This emulates doing

```
\def\word{\'elite}\@onelvel@sanitize\word
\expandafter\index\expandafter{\word}
```

The sanitization isn't applied to any remaining content of `\index`, such as the encap. For example,

```
\index{\`o|see{eth (\`o)}}
```

is implemented such that only the `\`o` part before the encap is sanitized so this would end up written to the index file as

```
\`o|see{eth (\IeC {\`dh })}
```

(`testidx` doesn't modify the `\index` command, but uses the `\expandafter` approach where the control sequence has a combination of sanitized and non-sanitized content.)

There's no support for other encodings.

2.3 Indexing application issues

An indexing application typically reads the external file created by L^AT_EX (the *input file* that contains data, discussed in the previous section) and produces another file (the *output file* that contains typesetting instructions) which can then be read by L^AT_EX (through commands like `\printindex`). The terminology here is a little confusing as the input file from the indexing application's point of view is an output file from L^AT_EX's point of view and vice versa. For consistency, the indexing application's point of view is used here.

The dummy entries are designed to test the indexing application's ability to collate entries into an ordered list where each entry has an associated set of page references (locations) or cross-references. The list may be sub-divided into letter groups, according to the initial letter of each entry. The definition of a 'letter' depends on the collation rule. For example, 'aeroplane', 'Ängelholm', 'Ångström' and 'Åßlar' may all belong to the 'A' letter group according to one rule (such as English) but may belong to different letter groups according to another rule (such as Swedish). In some languages, a 'letter' may actually be a digraph (such as 'dz') or a trigraph (such as 'dzs'). Entries that don't belong in any of the recognised letter groups are typically put into a default or 'symbols' group.

2.3.1 Extended Latin characters, digraphs and trigraphs

As mentioned above, the test entries include some words with extended Latin characters, digraphs and a trigraph to test the localisation support of the indexing application used in the document build process. There are three digraphs (ll, ij and dz) that may instead be represented by a single UTF-8 glyph (℔, ij and dz). The `diglyphs` option will switch to using these glyphs instead, but remember that the document font must support those characters if you want to try this.

2.3.2 Collation-level homographs

The words 'resume' and 'résumé' are both indexed. These should be treated as separate entries even if the comparator used by the indexing application considers them identical. Check that both words appear in the index. Similarly for `index/\index` and `recover/re-cover`.

2.3.3 Compound words

The test entries include space- or hyphen-separated compound words to test the sort rule. Different rules have different ways of treating spaces or hyphens.

One rule may ignore those characters (for example, 'vice-president' < 'viceroy' < 'vice versa') whereas another rule may treat a space as coming before a hyphen (for example, 'vice versa' < 'vice-president' < 'viceroy').

2.3.4 Numbers

The test entries include some numbers (2, 10, 16, 42, 100). The indexing application may identify these as numbers and order them numerically, or it may simply order them as a sequence of non-alphabetical characters (so 2 would be placed after 100).

2.3.5 Symbols

The test entries include two types of symbol entries. The first set are mathematical symbols, such as α (`\alpha`). The second set are the markers used in the dummy text to indicate where the indexing is taking place. The package options `prefix` and `noprefix` determine how these entries are indexed.

The `prefix` option (default) inserts the character `>` before the sort value for mathematical symbols and inserts the character `<` before the sort value for the markers. For example:

```
\index{>alpha@$\alpha$}
```

for α and

```
\index
{<tstidxmarker@{\csname tstidxmarker\endcsname
\space (\tstidxcsfmt {tstidxmarker})}}
```

for the symbol \cdot produced by `tstidx`'s marker command `\tstidxmarker`. This naturally gathers the two types of symbols. A sophisticated indexing application may then be customized to treat the character `>` as the 'maths' letter group and `<` as the 'marker' letter group.

The `noprefix` option doesn't insert these characters. This emulates simply doing

```
\index{alpha@$\alpha$}
```

for α (which puts α in the 'A' letter group) and

```
\index
{tstidxmarker@{\csname tstidxmarker\endcsname
\space (\tstidxcsfmt {tstidxmarker})}}
```

for the marker (which puts this symbol in the 'T' letter group).

A real document will likely provide syntactic commands for this type of indexing. For example, to index a maths symbol that's produced using a single control sequence (such as `\alpha`):

```
\newcommand{\indexmsym}[1]{%
 \index{#1@\csname #1\endcsname}}
```

The symbol is then indexed as, for example,

```
\indexmsym{alpha}
```

The `prefix` option simply emulates a minor adjustment to such a command to alter the sorting.

There are additional maths symbols that aren't governed by the prefix options as they start with alphabetical characters. These are simply indexed in the form:

```
\index{f(x)@$f(\protect\vec{x})$}
```

so they end up in the associated letter group ('F' in the above example).

There are also terms starting with a hyphen (command line switches) to test sorting. For example:

```
\index{-l (makeindex)@\protect\\tstidxappoptfmt{-l}\\protect\\tstidxappfmt{makeindex}}
```

These again aren't affected by the prefix options as the hyphen forms part of the term. Conversely, there are some terms starting with a backslash that have the leading backslash omitted from the sort term. For example

```
\index{index@\protect\\tstidxcsfmt{index}}
```

2.3.6 Multiple encaps

There are three test commands, which simply change the text colour, used as page encapsulator (encap) values. One of the dummy blocks of text has the same word ('paragraph') indexed multiple times with different encap values. For example, no encap:

```
\index{paragraph}
```

The first test encap (`\tstidxencapi`):

```
\index{paragraph|tstidxencapi}
```

Similarly for the second (`\tstidxencapii`) and third (`\tstidxencapiii`) test encaps. If all instances occur on the same page then this causes an encap clash for that entry on that page. The indexing application may or may not have a method for dealing with this situation.

2.3.7 Inconsistent encap in a range

There are some explicit ranges formed using `(` and `)` at the start of the encap value. For example, block 4 of the dummy text includes

```
\index{range|()}
```

which is closed in block 9 with

```
\index{range|)}
```

However in block 5, this term is indexed with one of the test encaps:

```
\index{range|tstidxencapi}
```

This can't be naturally merged into the range and causes an inconsistency. The indexing application may or may not have a method for dealing with this.

2.3.8 Cross-referenced terms

Some terms are considered a synonym of another term. Instead of duplicating the location lists for both terms, it's simpler for one term to redirect to the other in an index. This is typically done with the `see` encap. For example:

```
\index{gobbledgook|see{gibberish}}
```

The dummy text, like a real world document, will only index this type of term once so it only has one location which is encapsulated by `\see{other word}{page}`. Since this command ignores the second argument, no actual location will be visible in the page list.

The other type of cross-reference is done with the `seealso` encap (which has the same syntax as `see`). For example

```
\index{padding|seealso{filler}}
```

These types of entries will be indexed in other places as well to create a location list that has both page references and the cross-referenced term. In some cases (as in the above example) the encap's argument exactly matches the referenced term, but in other cases it doesn't. This inconsistency may or may not cause a problem for the indexing application.

One term in particular that's tested needs checking. The word 'lyuk' is first indexed without an encap, then indexed with the `seealso` encap and later indexed again without an encap. If the indexing application simply treats the `seealso` encap as just another formatting command, this can end up with the rather odd occurrence of the cross-reference appearing in the middle of the location list.

2.3.9 Untidy page lists

Some of the entries are indexed sporadically throughout the dummy text. Depending on the font size and page dimensions, this could result in a sequence of consecutive page numbers that can be concatenated into a neat range or it could lead to an untidy list that has odd gaps that prevent a range formation.

3 testidx-glossaries

The supplementary `testidx-glossaries` package loads `testidx` and `glossaries`. The commands used in the dummy text are altered to use `\glsadd` or `\gls`. The dummy entries all need to be first defined and the indexing activated. This is done with

```
\tstidxmakegloss
```

The glossary is then displayed with

```
\tstidxprintglossary
```

or with

```
\tstidxprintglossaries
```

(which will display all defined glossaries using the analogous command).

There are some minor differences in the package options shared by both `testidx` and `testidx-glossaries`, and there are some supplementary options only available with `testidx-glossaries`:

extra Load the extension package `glossaries-extra` [8].

nodesc Each entry is defined with an empty description (default). The `mcolindexgroup` style is set. You can override this in the usual way. For example:

```
\setglossarystyle{mcolindexspannav}
```

desc Each entry is defined with a description. In this case, the `indexgroup` style is set, but again you can override it.

makeindex This option is passed to `glossaries` and ensures that `\tstidxmakegloss` uses

```
\makeglossaries
```

and `\tstidxprintglossary` uses

```
\printglossary
```

The indexing should be done by `makeindex`, invoked directly or via the `makeglossaries` Perl script or the `makeglossaries-lite` Lua script.

xindy This option is passed to `glossaries` and again ensures that `\tstidxmakegloss` uses

```
\makeglossaries
```

and `\tstidxprintglossary` uses

```
\printglossary
```

The indexing should be performed by `xindy` (again either invoked directly or through one of the provided scripts).

tex This ensures that `\tstidxmakegloss` uses

```
\makenoidxglossaries
```

and `\tstidxprintglossary` uses

```
\printnoidxglossary
```

The indexing is performed by TeX and is *slow*—the document build may appear as though it has hung.

bib2gls This implicitly specifies `extra` and also passes the `record` option to the `glossaries-extra` package. In this case, `\tstidxmakegloss` uses

```
\GlsXtrLoadResources[<options>]
```

and `\tstidxprintglossary` uses

```
\printunsrtglossary
```

In this case, the indexing should be performed by `bib2gls` [7], a Java command line application designed to work with `glossaries-extra`. The `<options>` and the number of instances of

`\GlsXtrLoadResources` varies according to the package settings (such as `prefix` or `diglyphs`).

More detail is provided later on (see page 391).

manual Use this option if you don't want to use the helper commands `\tstidxmakegloss` and `\tstidxprintglossary`. You will need to ensure you pass the appropriate options to the `glossaries` or `glossaries-extra` package and load the files containing the entry definitions.

4 Examples

The following examples can be used to test the various indexing methods. To compile them, you need to have at least `testidx` version 1.1. For the examples using `testidx-glossaries`, it's best to have at least version 4.30 of `glossaries` and version 1.16 of `glossaries-extra`.

The letter groups created by each example are shown in Table 1 (in the order they appear in the index). In the table, ‘Symbols’ indicates the symbols group (which in `xindy` parlance is the default group), ‘Numbers’ indicates the group containing numerical terms and ‘Other’ indicates a headless group beyond the end of the alphabet. Some of the examples create their own custom groups. If a group contains initial letters that may not be expected to appear in that group (such as accented versions) then those letters are included afterwards in parentheses.

The contents of the symbols group for each example are shown in Table 2, where ‘markers’ indicates the marker commands prefixed with <, ‘maths’ indicates the mathematical symbols prefixed with >, ‘switches’ indicates the terms starting with a hyphen, ‘non-ASCII’ indicates the terms where the sort value starts with a non-letter ASCII character (typically the backslash \ at the start of accent or ligature commands, such as \` or \oe) and ‘UTF-8’ indicates the terms where the sort value starts with a UTF-8 character that doesn't fall into any of the recognised letter groups, according to the indexer's alphabet.

The ordering of the switches is shown in Table 3, and the ordering of the mathematical symbols is shown in Table 4 with the corresponding sort values shown in parentheses. These may all be in the symbols group or in their own group or scattered throughout the index in the various letter groups, as indicated in Table 1.

The ordering of the numbers (which may or may not be in their own group) is shown in Table 5, the collation-level homographs in Table 6, and a selection of compound words in Table 7.

The place name Aßlar contains an eszett (ß). In the bare ASCII mode this is indexed as

```
\index{Asslar@\ss lar}
```

while in the ASCII accents mode it's indexed as

```
\index{A\ss lar}
```

and in UTF-8 mode it's indexed as

```
\index{Aßlar}
```

Although Aßlar always appears in the 'A' group, its location within that group varies, as shown in Table 8.

Further tables show location lists:

- Table 9: for the entry with multiple encaps ('paragraph', Section 2.3.6);
- Table 10: for the entry with the explicit range interruption ('range', Section 2.3.7);
- Table 11: for the entry with the mid-*seealso* encaps ('lyuk', Section 2.3.8);
- Table 12: for an entry with a ragged page list ('block', Section 2.3.9).

A shell script was created for each example with the build process so that the complete document build could be timed (using the Unix `time` command). The elapsed real time *<minutes>:<seconds>* for each example is shown in Tables 13 for `testidx` and 14 for `testidx-glossaries`.

► Example 1 (`makeindex` and bare ASCII mode)

This builds on the example shown earlier with a `makeindex` style file to enable the group headings, the `fontenc` package [4] to provide the commands `\dh` (ð), `\th` (þ) and `\TH` (Þ), and the `amssymb` package [5] to provide the spin-weighted partial derivative `\eth` (ð). These extra packages allow for more test entries that would otherwise be omitted.

```
\documentclass{article}
\usepackage[a4paper]{geometry}
\usepackage{filecontents}
\usepackage[T1]{fontenc}
\usepackage{amssymb}
\usepackage{makeidx}
\usepackage{testidx}
\makeindex
\begin{filecontents}{\jobname.ist}
headings_flag 1
heading_prefix "\\heading{"
heading_suffix "}\n"
\end{filecontents}
\newcommand{\heading}[1]{%
\item\textbf{\#1}\indexspace}

\begin{document}
\testidx
\printindex
\end{document}
```

This uses the default settings `prefix` and (since there's no UTF-8 support) `stripaccents`. The heading command is simplistic as these examples are testing the indexing applications rather than the index style. The build process is:

```
pdflatex doc
makeindex -s doc.ist doc
pdflatex doc
```

(where the file is called `doc.tex`).

The terms that are placed in the alphabetical groups have been ordered using a case-insensitive word comparator, the numbers have been sorted numerically (Table 5) and the symbols have been sorted using a case-sensitive comparator, as can be seen by the ordering of the switches (Table 3). Since the accent commands have been stripped, the words are all placed in the basic Latin letter groups (Table 1).

► Example 2 (`makeindex` and ASCII accents mode)

This is the same as the previous example except for the package option:

```
\usepackage[nostripaccents]{testidx}
```

This doesn't strip the accents so, for example, 'élite' is indexed as '\elite'. This causes all the words starting with extended Latin characters to appear in the symbols group (Table 2) due to the leading backslash in the control sequences. Since \AA expands to \r A, Å ends up between œ (\oe) and þ (\th). 'Aßlar' is placed at the start of the 'A' letter group before 'aardvark' (Table 8) since the second character in the sort key is a backslash (from the start of \ss) which comes before 'a'.

► Example 3 (`makeindex`, bare ASCII mode and no prefixes)

This is the same as Example 1 except for the package option:

```
\usepackage[noprefix]{testidx}
```

This doesn't insert the < and > prefixes that kept the markers and maths together in Example 1. The markers remain close to each other as they still start with the same sub-string (now `tstidx` instead of `<tstidx`) but they have been moved to the 'T' letter group. The maths symbols are now scattered about the index (Table 1), for example, α is in the 'A' letter group (since its sort value is now `alpha`). Only the switches remain in the symbols group (Table 2).

► Example 4 (`makeindex`, ASCII accents mode and no prefixes)

This is the same as Example 2 except for the extra package option:

```
\usepackage[nostripaccents,noprefix]{testidx}
```

As with Example 3, the marker and maths entries are no longer in the symbols group (Table 1), but as with Example 2 that group (Table 2) now contains the terms starting with accent commands (as well as the switches).

► **Example 5** (`makeindex -l`)

This is the same as Example 1 except for the build process which uses `makeindex`'s `-l` switch:

```
pdflatex doc
makeindex -l -s doc.ist doc
pdflatex doc
```

This changes the ordering of the compound words shown in Table 7 (except for ‘yo-yo’). The ordering is still case-insensitive for words (Table 8) and case-sensitive for symbols (Table 3).

► **Example 6** (`makeindex` and sanitized UTF-8)

This is like Example 1 but UTF-8 support has been enabled through the `inputenc` package:

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

The default `sanitize` option is on, which means that the UTF-8 characters in the sort key are sanitized and so don't expand when writing the input file. The build process used in Example 1 fails because `makeindex` isn't configured for UTF-8 and the resulting output file is corrupt. This can almost be fixed with `iconv` except near the end of the file, which triggers the error

```
\heading{iconv: illegal input sequence}
```

This is because only the first octet (C3) of a two-octet character has been put in the argument of `\heading`. The only way to avoid this is to omit the headings, so the build process is:

```
pdflatex doc
makeindex -o doc.tmp doc
iconv -f utf8 doc.tmp > doc.ind
pdflatex doc
```

The ‘Other’ groups shown in Table 1 highlight the way that `makeindex` is sorting according to each octet, so the first group after Z contains Á (C3 81), Å (C3 84), Ä (C3 85), Í (C3 8D), Ö (C3 96), Ø (C3 98), Ú (C3 9A), P (C3 9E), æ (C3 A6), é (C3 A9), ð (C3 B0) and þ (C3 BE). From `makeindex`'s point of view, these all belong to the C3 letter group (which is why it tried to write the character C3 as the argument of `\heading` when the headings setting was on).

The next few examples use `xindy` to perform the indexing. The `makeindex` style file (`.ist`) is no longer applicable. An `xindy` module (`.xdy`) is used instead. A straight substitution of `makeindex` with `texindy` causes an error message with the sample entries:

```
ERROR: Cross-reference-target
("\\tstidxstyfmt {inputenc}") does not exist!
```

Unlike `makeindex`, `texindy` recognises the `see` and `seealso` encaps as cross-references (rather than just a formatting command). This error is the result of

```
\index{fontenc@{\tstidxstyfmt {fontenc}}
package|seealso{\tstidxstyfmt {inputenc}}}
('fontenc package, see also inputenc'). texindy checks that the cross-referenced term also exists, but there's no exact match here as the cross-referenced term was indexed slightly differently using
```

```
\index{inputenc package@{\tstidxstyfmt
{inputenc} package}}
```

('inputenc package'). This inconsistency is the result of a stylistic choice to avoid the repetition of the word ‘package’ in the exact match ‘fontenc package, *see also* inputenc package’.

If you want to ignore these kinds of inconsistencies, you can switch off the automatic verification in the `.xdy` file when defining a cross-reference class. For example:

```
(define-crossref-class "seealso"
:unverified)
```

Unfortunately with `texindy` this causes the error

```
ERROR: replacing location-reference-class
`"seealso"' is not allowed !
```

since the `seealso` class has already been defined (in the file `makeindex.xdy`, which is loaded by `texindy` to provide compatibility with `makeindex`). One possible workaround is to define a custom module and use `xindy` directly (instead of using `texindy`).

In your real document you can circumvent this issue by ensuring an exact match in your `see` and `seealso` encaps or by writing your own custom `xindy` module that defines the `seealso` class as unverified.

Alternatively, you can create your own custom cross-reference encaps. For example

```
(define-crossref-class "uncheckedseealso"
:unverified)
(markup-crossref-list
:class "uncheckedseealso"
:open "\seealso" :close "{}")
```

and use this instead. The `tstidx` package allows you to try this out by providing a command to set your own cross-reference encaps value. For example:

```
\tstidxSetSeeAlsoEncap{uncheckedseealso}
```

The problematic cross-reference now becomes

```
\index{fontenc@{\tstidxstyfmt{fontenc}}
package|uncheckedseealso{\tstidxstyfmt
{inputenc}}}
```

which uses `uncheckedseealso` instead of `seealso`.

The examples below circumvent this issue by using `xindy` directly with a custom module.

► **Example 7** (`xindy` and sanitized UTF-8)

The sample `xindy` style provided here mostly replicates `texindy.xdy` but doesn't load `makeindex.xdy`.

The cross-reference classes (`see` and `seealso`) both have the verification check switched off. This custom module also has to define the location classes provided by `makeindex.xdy` and define the test encapsulated values used by `testidx`.

```
\documentclass{article}
\usepackage[a4paper]{geometry}
\usepackage{filecontents}
\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage{amssymb}
\usepackage{makeidx}
\usepackage{testidx}

\makeindex

\begin{filecontents*}{\jobname.xdy}
(require "latex.xdy")
(require "latex-loc-fmts.xdy")
(require "latin-lettergroups.xdy")

(define-crossref-class "see" :unverified)
(markup-crossref-list :class "see"
  :open "\see{" :sep " ; " :close "}{})"

(define-crossref-class "seealso" :unverified)
(markup-crossref-list :class "seealso"
  :open "\seealso{" :sep " ; "
  :close "}{})"

(markup-crossref-layer-list :sep ", ")

(define-location-class-order
  ("roman-page-numbers"
   "arabic-page-numbers"
   "alpha-page-numbers"
   "Roman-page-numbers"
   "Alpha-page-numbers"
   "see"
   "seealso"))

; list of allowed attributes
(define-attributes ((

  "tstidxencapi"
  "tstidxencapii"
  "tstidxencapiii" )))

; define format to use for locations
(markup-locref :open "\tstidxencapi"
  :close "}" :attr "tstidxencapi")
(markup-locref :open "\tstidxencapii"
  :close "}" :attr "tstidxencapii")
(markup-locref :open "\tstidxencapiii"
  :close "}" :attr "tstidxencapiii")

; location list separators
(markup-locref-list :sep ", ")
(markup-range :sep "--")

\end{filecontents*}

\begin{document}
\testidx
\printindex
\end{document}
```

The build process is

```
pdflatex doc
xindy -M doc -L english -C utf8 -t doc.ilg \
      doc.idx
pdflatex doc
```

The ordering for some of the extended characters is a little odd with the `english` setting. For example, ß comes between ‘n’ and ‘p’ (Table 8) and Á, Ä, Å, Í and Ú are all in the O letter group (Table 1). They have not been considered either symbols (like Ć, which doesn’t occur in English words) or sorted according to their base letter (like é, which does). Better results are obtained with the language set to `general`, which is used later in Example 17.

The switches aren’t placed in the symbols group but have instead been placed in the alphabetical letter groups (ignoring the initial hyphen). The numbers (which are now in the symbols group) have been sorted as strings rather than numerically (Table 5).

The term `\index` is present, but the word ‘index’ has been omitted (Table 6) and its page list has been merged with the `\index` locations. A real world document would need to ensure unique sort keys. (For example, use `index.cs` as the sort value for `\index`.) The other collation-level homographs ‘recover’/‘re-cover’ and ‘resume’/‘résumé’ don’t have this problem as the sort values for each pair are non-identical even though the comparator may consider them equivalent.

► Example 8 (xindy, sanitized UTF-8 and letter order)

The sorting in Example 7 can be adjusted to letter ordering by adding the following line to the custom `.xdy` file:

```
(require "letter-order.xdy")
```

This alters the ordering of the compound words (see Table 7), but this doesn’t quite match the order produced by `makeindex`’s letter order option used in Example 5 for the hyphenated words. The terms ‘`\index`’ and ‘`index`’ have again been merged due to their identical sort values (Table 6), and the switches are in the alphabetical letter groups (Table 1) but their locations within those groups have changed as a result of the spaces being ignored.

► Example 9 (xindy, sanitized UTF-8 and ignore hyphen)

The previous example can be slightly altered by changing `letter-order` to `ignore-hyphen`. There’s no difference here from Example 8 in the order of the collation-level homographs ‘recover’ and ‘re-cover’ (Table 6). There is a difference in the ordering of the compound words shown in Table 7, which is back to the word order from Example 7, and the switches are

still in the alphabetical groups, so there's no noticeable difference between this example and Example 7. It seems that `xindy` always ignores hyphens regardless of whether or not the `ignore-hyphen` module is loaded.

► **Example 10** (`xindy`, sanitized UTF-8 and ignore punctuation)

Another option is to use the `ignore-punctuation` module. However, swapping `ignore-hyphen` in the previous example for `ignore-punctuation` causes an error while reading `ignore-punctuation.xdy`:

```
#<OUTPUT STRING-OUTPUT-STREAM>> ends within
a token after multiple escape character
```

The problem seems to come from the line

```
(sort-rule "\\" " ")
```

If I remove

```
(require "ignore-punctuation.xdy")
```

and replace it with the contents of that file without the problematic line, the document is able to compile.

This example differs from the previous one, as it also causes the prefix characters `<` and `>` to be ignored, so this behaves much like the `noprefix` option with the maths and markers placed in the alphabetical letter groups (Table 1).

The sorting is still case-insensitive, but the difference caused by the ignored punctuation can be seen in the ordering of the switches. For example, the term `-l (makeindex)` is now treated as `lmakeindex` (all punctuation stripped) instead of `l(makeindex)` (only hyphen and space stripped), so it's now after `-L icelandic (xindy)` (since '`i`' < '`m`') whereas in the previous example it came before `-L danish (xindy)` ('`('` < '`d`').

► **Example 11** (`xindy`, sanitized UTF-8 and numeric sort)

Example 7 can be easily modified to sort the numbers numerically by adding the line:

```
(require "numeric-sort.xdy")
```

to the start of the `.xdy` file. A separate group for the numbers can also be defined in this file:

```
(define-letter-group "Numbers"
  :prefixes ("0" "1" "2" "3" "4" "5" "6"
  "7" "8" "9") :before "A")
```

The ordering of the defined attributes tells `xindy` the order of precedence when there's an encapsulation clash (see Section 2.3.6). In the previous example, the `tstidxencapi` encapsulation took precedence in the conflict in the 'paragraph' entry (see Table 9), but there are still two instances of page 2 in the location list as the default encapsulation (where no encapsulation has been specified) has been kept as well as the dominant `tstidxencapi`

encapsulation. This can be fixed by adding `default` to the end of the list of allowed attributes:

```
(define-attributes ((

  "tstidxencapi" "tstidxencapii"
  "tstidxencapiii" "default")))
```

This will cause a warning

```
WARNING: ignoring redefinition of
attribute "default" in
(DEFINE-ATTRIBUTES
  (( "tstidxencapi" "tstidxencapii"
  "tstidxencapiii" "default")))
```

This is because `latex-loc-fmts.xdy` already contains an attribute list containing `default`:

```
(define-attributes (( "default" "textbf"
  "textit" "hyperpage")))
```

To remove the warning, delete the line

```
(require "latex-loc-fmts.xdy")
```

from the custom `.xdy` file. Any of the usual L^AT_EX attributes, such as `hyperpage`, that are provided in the file `latex-loc-fmts.xdy` can be added to the custom attributes list if required.

► **Example 12** (`xindy`, sanitized UTF-8, no prefixes and numeric sort)

The previous example is modified here so that it doesn't use the `>` and `<` prefixes. The `testidx` package is now loaded using:

```
\usepackage[noprefix]{testidx}
```

`inputenc` is again loaded to enable UTF-8 support. The markers and maths symbols are now placed in the letter groups (Table 1). For example, α now has the sort value `alpha`, so it's in the A letter group, and ∂ has the sort value `partial`, so it's in the P letter group.

► **Example 13** (`xindy`, active UTF-8 and numeric sort)

Example 11 is modified here so that it doesn't sanitize the sort value. The `testidx` package is now loaded using:

```
\usepackage[nosanitize]{testidx}
```

The `inputenc` package is again loaded to enable UTF-8 support, which means that the first octets of the UTF-8 characters are active so they are expanded when written to the index file. This causes the `xindy` error

```
ERROR: CHAR: index 0 should be less than
the length of the string
```

This error occurs when the sort value is empty.

Recall from Example 7 that the example's custom module loads the file `latex.xdy`. This in turn loads `tex.xdy` which strips commands and braces

from the sort key. This means that the sort keys that solely consist of commands (such as `\IeC{\TH}`) collapse to an empty string, which triggers this error.

As a result of the error, no output file is created, so the document doesn't contain an index. One way to force this example document to have an index is to remove the line

```
(require "latex.xdy")
```

and add the content of `latex.xdy` without the line

```
(require "tex.xdy")
```

but this means that all the words starting with extended characters end up in the symbols group since the initial backslash in `\IeC` is a symbol (which is what we'd get if we use `makeindex` instead).

An alternative approach is to keep `latex.xdy` and add a merge rule for the problematic entries:

```
(merge-rule "\\\TH *" "TH" :regexp :again)
(merge-rule "\\\th *" "th" :regexp :again)
```

(and similarly for other commands like `\ss` and `\dh`) before loading `latex.xdy`.

This example uses this simpler method, which strips all the `\IeC` commands but converts the commands (such as `\TH`) representing characters. This essentially reduces the sort values to much the same as the bare ASCII mode in Example 1. In both this example and Example 1, the sort value for ‘résumé’ becomes ‘resume’. This means that two distinct terms have identical sort values. In `makeindex`'s case, the terms are deemed separate entries as the actual part is different, but `xindy` merges entries with identical sort values, so only one of these two terms (‘résumé’) appears in the index. (As happens with ‘index’ and ‘\index’, and again it's the first term to be indexed that takes precedence.)

The alphabetical ordering is now reasonable for English, but not for other languages, such as Swedish or Icelandic, that have extended characters, such as `ø` or `þ`, that form their own letter groups. (This wouldn't change even if the language option specified with `-L` changes as there are no actual extended characters in the index file, just control sequences representing them.)

This example provides a useful illustration between using TeX engines that natively support UTF-8 and simply enabling UTF-8 support through `inputenc`. Replacing `inputenc` and `fontenc` with `fontspec` and switching to XeLaTeX or LuaLaTeX shows a noticeable difference. It's therefore not enough to have a Unicode-aware indexing application, but it's also necessary to ensure the extended characters are correctly written to the indexer's input file.

► Example 14 (`xindy`, sanitized UTF-8, custom groups and numeric sort)

This example returns to using the `sanitize` option so that the UTF-8 characters appear correctly in the index file. We can build on Example 11 to create two custom groups that recognise the `<` and `>` prefixes:

```
(define-letter-group "Maths"
  :prefixes (">") :before "Numbers")
(define-letter-group "Markers"
  :prefixes ("<") :before "Maths")
```

I also tried to define a similar group for the switches:

```
(define-letter-group "Switches"
  :prefixes ("-"))
```

but this doesn't work (Table 1) as the hyphen is by default ignored (see Example 9). Setting a sort rule for the hyphen doesn't seem to make a difference.

Now the default symbols group (Table 2) only contains the UTF-8 characters that aren't recognised by the language module.

► Example 15 (`xindy -L icelandic`, sanitized UTF-8, custom groups and numeric sort)

It's time to try out some other languages. This example uses the same document and style from Example 14 but substitutes `icelandic` for `english` in the `xindy` call. This results in some extra letter groups (see Table 1).

The Icelandic alphabet has ten extra letters (in addition to the basic Latin set) Á(á), Ð(ð), É(é), Í(í), Ó(ó), Ú(ú), Ý(ý), Þ(þ), Æ(æ) and Ö(ö). There is a letter group for the `ð` entry, but it's headed with the lower case `ð` rather than the upper case `Ð`. (All the other letter groups are headed with an upper case character, including `P`.) There are also letter groups for `P`, `Æ` and `Ö`, but not for the acute accents.

The non-native characters have a more logical ordering than in the English examples with `ß` treated as ‘ss’ (Table 8), but `Ã` and `œ` are in the `Æ` group (Table 1) and `Ø` is in the `Ö` letter group. The symbols group contains the remaining extended characters (Table 2).

► Example 16 (`xindy -L hungarian`, sanitized UTF-8, custom groups and numeric sort)

As above but now using `-L hungarian`. This also results in some extra letter groups (such as `Ö`), but there are some missing groups that should be in the Hungarian alphabet, such as the digraphs `Dz(dz)` and `Ly(lly)`, and the trigraph `Dzs(dzs)`.

The `O` letter group contains an odd collection of extended characters, such as `Ã`, `Â`, `P` and `ð`. As with the `english` setting, `ß` has an unexpected location between ‘n’ and ‘p’ (Table 8).

In theory it should be possible to add letter groups for digraphs and trigraphs using a similar method as the other custom groups:

```
(define-letter-group "Dz"
  :prefixes ("DZ" "Dz" "dz")
  :after "D" :before "E")
```

Unfortunately this doesn't work as the 'D' letter group takes precedence because it was defined first. (The language modules are loaded before the custom module.) A complete new language module is needed to make this work correctly, which is beyond the scope of this article. Another possibility is to use glyphs instead of the digraphs, but this is only possible for digraphs that have a glyph alternative.

► **Example 17** (`xindy`, sanitized UTF-8, selected digraph glyphs, custom groups and numeric sort)
This example is like Example 14 but the `diglyphs` option is used.

```
\usepackage[diglyphs]{testidx}
```

This means that instead of using the two characters 'dz' in words like 'dzéta', the single glyph dz is used. It should now be possible to create the Dz letter group as in the example above but with the glyphs DZ, Dz and dz.

```
(define-letter-group "Dz"
  :prefixes ("DZ" "Dz" "dz")
  :after "D" :before "E")
```

Similarly for IJ, ij and IL, ll. There's no glyph used in the trigraph dzs.

Since these characters are not easily supported by `inputenc` and `fontenc`, it's necessary to use Xe^LA_TE_X or Lua^LA_TE_X instead. This means replacing `inputenc` and `fontenc` with `fontspec`.

```
\usepackage{fontspec}
```

Some fonts don't support these glyphs (ij is the most commonly supported of this set), so the choice here is quite limited. Some fonts support the glyphs in only one family or weight. For example, Linux Libertine O and FreeSerif support all glyphs in the default medium weight but the ll and IL glyphs are missing from bold. I've chosen DejaVu Serif for the document font in this example as it has the best support of all my available fonts:

```
\setmainfont{DejaVu Serif}
```

The change in font slightly alters some of the page lists in the index. The build process is now:

```
xelatex doc
xindy -M doc -L general -C utf8 \
-t doc.ilg doc.idx
xelatex doc
```

(I've set the language to `general` to reflect the mixture of alphabets.)

This example generates a warning from `xindy`:

```
WARNING: Found a :close-range in the
index that wasn't opened before!
Location-reference is 5 in keyword (range)
I'll continue and ignore this.
```

The altered page breaking caused by the font change has resulted in both the opening range produced with

```
\index{range|{}}
```

and the interrupting encap produced with

```
\index{range|tstdxencapi|}
```

to occur on page 2. The open range encap is dropped in favour of the `tstdxencapi` encap. This means that the closing range

```
\index{range|)}
```

on page 5 no longer has a matching opening range, so no range is formed (Table 10).

As can be seen from Table 1, there's no symbols group for this example. The markers and maths have been assigned to their own groups through the use of their < and > prefixes, the numbers are in their own number group, the glyphs dz, ij, and ll have been assigned to separate groups, and the remaining UTF-8 characters have all been assigned to the basic Latin letter groups, as a result of the `general` language setting. The switches still have the hyphen ignored and so are in the letter groups.

The trigraph dzs is still unrecognised, as are the dd, ff, ly and Ng digraphs, which haven't been replaced with glyphs. (As most Te_X users will know, there is a glyph for the ff digraph in most fonts, but although the sequence ff is usually converted to a ligature when typesetting, it's written to the index file as two characters. There's no corresponding glyph for the title case version Ff.)

The examples now switch to `testidx-glossaries`, which provides extra sorting methods. Some of the informational blocks of text are altered by this package, so the page numbers may be different in the location lists due to the difference in some paragraph lengths.

Instead of using `\index`, the terms are first defined using

```
\newglossaryentry{<label>}{<options>}
```

where `<label>` (which can't contain special characters) uniquely identifies the term and `<options>` is a `<key>=<value>` list. The main keys are `name` (the way the term appears in the glossary) and `description`. By default the sort value is the same as the name (as `\index` when `C` isn't used) but the `sort` key can

be used to provide a different value. The files containing these definitions are automatically loaded by `\tstidxmakegloss`.

The terms are then displayed and indexed using commands like `\gls{label}` throughout the document text. This will display the value of the `text` key, which if omitted defaults to the same as `name`.

For example, with the normal indexing methods, the term $f(\vec{x})$ can be displayed and indexed in the text using

```
\[ f(\vec{x})\index{f(x)@\$f(\vec{x})$} \]
```

whereas with `glossaries` the term is first defined in the preamble:

```
\newglossaryentry{fx}{name={$f(\vec{x})$},
  text={f(\vec{x})},
  sort={f(x)},
  description={}}
```

and then used in the document:

```
\[ \gls{fx} \]
```

In the text this does `f(\vec{x})` (the value of the `text` key), in the index this does `$f(\vec{x})$` (the value of the `name` key), and it's sorted by `f(x)` (the value of the `sort` key).

Cross-references are performed using the `see` key, for example:

```
\newglossaryentry{padding}{name={padding},
  see={[seealsoname]filler},description={}}
```

(where the `see` value is a comma-separated list of labels optionally preceded by a tag) or using `\glssee`, for example,

```
\glssee[seealsoname]{padding}{filler}
```

The `glossaries-extra` package provides the `seealso` key, which is essentially the same as `see` with the tag set to `\seealsoname`. If this key is detected, it will be used instead. For example:

```
\newglossaryentry{padding}{name={padding},
 seealso={filler},description={}}
```

These methods essentially index the reference as:

```
padding?\glossentry
{padding}|glsseeformat[\seealsoname]{filler}
```

with Z as the location (the `glossaries` package uses ? instead of @ as the actual character).

Since `makeindex` by default lists upper case alphabetical locations last, this automatically moves the cross-reference to the end of the list.

► Example 18 (`testidx-glossaries` and `makeindex`)

The basic test document is:

```
\documentclass{article}
\usepackage[a4paper]{geometry}
\usepackage[T1]{fontenc}
\usepackage{amssymb}
```

```
\usepackage{testidx-glossaries}
\tstidxmakegloss

\renewcommand*\glstreenamefmt{#1}{#1}
\renewcommand*\glstreegroupheaderfmt{#1}{%
  \textbf{#1}}
```

```
\begin{document}
\testidx
\tstidxprintglossaries
\end{document}
```

The `mcolindexgroup` glossary style sets the name in bold by default, so I've redefined `\glstreenamefmt` to prevent this. (There's no need to distinguish the name when there are no descriptions.)

For this example, my build process is

```
pdflatex doc
makeglossaries-lite doc
pdflatex doc
```

This uses the Lua script rather than the Perl script. The Lua script simply determines the required indexing application (in this case `makeindex`) and the correct options from the `.aux` file and runs it. The `makeglossaries` Perl script does more than this and is used in the next example.

For comparison, an explicit call to `makeindex` was also used:

```
pdflatex doc
makeindex -t doc.glg -o doc.gls -s doc.ist \
  doc.glo
pdflatex doc
```

The only difference in the result is in the build time, which is slightly faster. The times for both build methods are shown in Table 14.

Since the `inputenc` package isn't used, accents are stripped as with Example 1. This means it's emulating, for example:

```
\newglossaryentry{elite}{name={'elite},
  sort={elite},description={}}
```

There are some differences between the index produced in this example and that produced in Example 1 (aside from the page numbering and the differences between the index and glossary styles). The ordering of `\index` and 'index' have changed (Table 6). In Example 1, the control sequence `\index` is indexed as

```
index@\tstidxcsfmt{index}
```

and the term 'index' is just indexed as `index`. With `glossaries` the control sequence is effectively indexed as

```
index?\glossentry{cs.index}
```

and the term is effectively

```
index?\glossentry{index}
```

When `makeindex` encounters terms with identical sort values, it seems to give precedence to terms where the sort value is identical to the actual value. So in the first example, ‘index’ (which has no separate sort) comes before `\index`. With `glossaries`, both have a distinct sort and actual value.

A similar thing happens with ‘resume’

```
resume?\glossentry{resume}
and ‘résumé’
resume?\glossentry{resumee}
```

Since the accents have been stripped, both terms have ‘resume’ as the sort value. (Since active characters can’t be used in labels and labels must be unique, the label for the second term is `resumee`.)

► Example 19 (testidx-glossaries and makeglossaries)

This example uses the same document as the previous one above, but uses the `makeglossaries` Perl script in the build process instead of the Lua script:

```
pdflatex doc
makeglossaries doc
pdflatex doc
```

The difference here can be seen in the location list for the ‘paragraph’ entry (see Table 9). The script has detected `makeindex`’s multiple encapsulation warning and tried to correct the problem. Version 2.20 incorrectly gives precedence to a non-range encapsulation over an explicit range encapsulation which then causes `makeindex` to trigger the error

```
-- Extra range opening operator (.
```

This is the same problem that occurred with `xindy` in Example 17. `makeglossaries` version 2.21 (provided with `glossaries` v4.30) corrects this and gives the range encapsulation precedence. The only problem that remains is just the inconsistent page encapsulator within a range warning.

► Example 20 (testidx-glossaries, bare ASCII mode and xindy)

The test document from Example 18 can be modified to use `xindy` instead of `makeindex` by adding the `xindy` package option:

```
\usepackage[xindy]{testidx-glossaries}
```

The `glossaries` package provides a custom `xindy` module (automatically generated by `\makeglossaries`). Minor adjustments can be made before the module is written using commands or package options. For example, to add the test encapsulation:

```
\GlsAddXdyAttribute{tstidxencapi}
\GlsAddXdyAttribute{tstidxencapii}
\GlsAddXdyAttribute{tstidxencapiii}
```

Again we can take advantage of the < and > prefixes:

```
\GlsAddLetterGroup{Maths}{:prefixes (">"}
  :before "glsnumbers"}
\GlsAddLetterGroup{Markers}{:prefixes ("<"}
  :before "Maths"}
```

(The `glossaries` package provides its own version of the numbers group called `glsnumbers`.)

The `numeric-sort` module isn’t loaded by default, so it needs to be explicitly added if numerical ordering is required:

```
\GlsAddXdyStyle{numeric-sort}
```

The above lines all need to go before

```
\tstidxmakegloss
```

The build process is:

```
pdflatex doc
makeglossaries doc
pdflatex doc
```

The Lua alternative can also be used, or a direct call to `xindy`:

```
xindy -L english -I xindy -M doc -o doc.gls \
  -t doc.glg doc.glo
```

The difference between this example and the earlier `xindy` examples is that the indexing information is written in `xindy`’s native format, for example

```
(indexentry
  :tkey {"elite" "\glossentry{elite}"})
:locref "{}-{3}"
:attr "pageglsnumberformat")
```

(`pageglsnumberformat` is the default encapsulation used by `glossaries` in `xindy` mode when the `format` key hasn’t been set and the `page` counter is used for the locations.)

The example document doesn’t load `inputenc`, which means the bare ASCII mode is on, which is why the accent doesn’t appear in the sort field (identified in `:tkey`). This means that the sort value for ‘résumé’ is once again ‘resume’ and the conflicting unaccented ‘resume’ is lost (Table 6). The hyphens are again ignored so the switches are placed in the alphabetical letter groups (Table 1).

► Example 21 (testidx-glossaries, sanitized UTF-8 and xindy)

This example makes a minor adjustment to the previous one by adding

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

This enables the sanitized UTF-8 mode so the sort values contain UTF-8 characters. (The `glossaries` package automatically sanitizes the `sort` key by default, but the `testidx-glossaries` package will ensure that its own `nosanitize` option is honoured, which just passes `sanitizesort=false` to `glossaries`.)

The build process again uses `makeglossaries`. Since the document hasn't loaded any language packages, the language option written to the `.aux` file defaults to English so `makeglossaries` calls `xindy` with `-L english`. This means the extended characters are ordered in the same way as in Example 14 (Table 1).

► **Example 22** (`testidx-glossaries`, `xindy` and non-standard page numbering)

`makeindex` can only recognise roman (i, I), arabic (1) and alphabetic (a, A) locations. `xindy` has more flexibility, so this example makes a minor adjustment to the previous example to use an unusual page number scheme. This requires `etoolbox` [3] (automatically loaded by `glossaries`) for `\newrobustcmd`, and the `stix` package [1] for the six dice commands `\dicei`, ..., `\dicevi`:

```
\newrobustcmd{\tally}[1]{%
\ifnum\number#1<7
  $`csname dice\romannumeral#1\endcsname$%
\else
  $`dicevi$%
\expandafter\tally\expandafter{\numexpr#1-6}%
\fi
}
\renewcommand{\thepage}{\tally{\arabic{page}}}
```

The page numbers are now represented by dice. For example, page 2 is ☐ and page 10 is ☐☐.

Since the `stix` package by default automatically changes the document font, which will alter the page breaking, I've used the `notext` option to prevent this:

```
\usepackage[notext]{stix}
```

This allows a better comparison with the previous example.

The locations are now written to the indexing file in the form `\tally {⟨n⟩}`, where `⟨n⟩` is the page number. (The backslash is automatically escaped by `glossaries`. The space is significant.) `xindy` needs to be informed of this new location format:

```
\GlsAddXdyLocation{\tally}{%
:sep "\string\tally\space{"
"arabic-numbers" :sep "}"}
```

Aside from the location presentation, there is one difference between this example and the previous one when used with versions of `glossaries` below 4.30, and that's the cross-reference location. For example, with `glossaries` v4.29, the 'lyuk' entry appears as 'see also digraph, ☐, ☐' but for v4.30 it appears as '☐, ☐, see also digraph' (Table 11). This is due to a bug that has been corrected in v4.30.

► **Example 23** (`testidx-glossaries`, bare accents mode and `TEX`)

If, for some reason, you're unable or unwilling to use an external indexing application, the `glossaries` package provides a method of alphabetical sorting using `TEX`. The document from Example 18 can be adapted to use this method by adding the `tex` option:

```
\usepackage[tex]{testidx-glossaries}
```

The accents are stripped by default so the sorting is just performed on the basic Latin set.

The build process is simply

```
pdflatex doc
pdflatex doc
```

This method is considerably longer than the others (see Table 14) and has the worst results.

There's no numbers group with this method. The numbers are included with the symbols (Table 2), but are ordered numerically (Table 5). The ordering of the compound words has changed (Table 7) with somewhat eccentric results. There are no range formations, even for explicit ranges, and the range interruption (Table 10) interrupts the list formatting (a space is missing).

The 'see also' cross-reference in Table 11 doesn't interrupt the location list, but this is only because the `see` key was used when defining the entry (which is why it's at the start of the list). If `\glssee` had been used instead within the document, it would have produced the same result as Example 1.

► **Example 24** (`testidx-glossaries`, bare accents mode and `TEX` with letter ordering)

The previous example used the `glossaries` package's default `sort=standard` setting, which sets the entry `sort` key, if omitted, to the `name` key and optionally sanitizes it. The command `\printnoidxglossary` also accepts a `sort` key in the optional argument to allow different ordering for different glossaries. (This capability is not available with `\printglossary`.) The localised `sort` key allows the values `word` and `letter` for word and letter ordering, so this example replaces

```
\tstidxprintglossaries
```

with

```
\printnoidxglossary[sort=letter]
```

to test letter order sorting with `TEX`. This again takes a long time (Table 14). The ordering of the compound words (Table 7) now matches the `xindy` letter order in Example 8. There's a change in the order of one of the collation-level homographs from the previous example: 're-cover' is now after 'recover' (Table 6). Other than that, this method produces much the same results as the previous example.

So far the examples have all used alphabetical ordering for the majority of the entries based on the value of the `sort` key (or the `name`, if `sort` is omitted). The `glossaries` package also allows sorting according to definition or use. The next few examples illustrate this.

► **Example 25** (`testidx-glossaries` and order of definition with `makeindex`)

The `glossaries` package provides the options `sort=def` and `sort=use` to switch to order of definition or first use within the document. The code used in Example 19 needs to be adjusted to pass this option since `glossaries` is being loaded implicitly:

```
\PassOptionsToPackage{sort=def}{glossaries}
\usepackage{testidx-glossaries}
```

Alternatively (`glossaries` v4.30):

```
\usepackage{testidx-glossaries}
\setupglossaries{sort=def}
```

This method works by overriding the `sort` value so that it's just a number that is incremented every time a new entry is defined. This means that `makeindex` orders numerically, and all entries are placed in the numbers group (Table 1). It therefore makes no sense to use a style with group headings with this option. The entries that are actual numbers (Table 5) are no longer in numerical order according to their value given in the `name` field.

The build process again uses `makerglossaries`, which deals with the conflicting `encaps` for page 3 (Table 9). This method is faster than Example 18 (Table 14) as it's simpler to compare two integers than to perform a case-insensitive word-order comparison between two strings.

► **Example 26** (`testidx-glossaries` and order of definition with `xindy`)

This is like the previous example, but `xindy` is used:

```
\PassOptionsToPackage{sort=def}{glossaries}
\usepackage[xindy]{testidx-glossaries}
```

The attributes (`encaps`) need to be specified as in Example 20, but since we're sorting by order of definition it's not possible to define the `maths` or `markers` groups.

Since numeric comparisons are faster than string comparisons, the `numeric-sort` style from Example 20 is also used (Table 14). This example will still work without that style as the sort values are zero-padded to six digits. (If you have 1,000,000 or more entries, you'll need `numeric-sort` to enforce numerical comparisons.)

The `glossaries` package automatically defines the `numbers` group, so all entries are placed in that. If the package option `glsnumbers=false` is also passed

to `glossaries`, then the entries will instead be placed in the default group.

There's no longer a problem with the collation-level homographs (Table 6) as the sort values are now unique numbers, so 'index' and 'resume' have reappeared in the index.

► **Example 27** (`testidx-glossaries` and order of definition with `TEX`)

This example makes a minor change to the document used in Example 24:

```
\printnoidxglossary[sort=def,nogroupskip,
style=mcolindex]
```

This orders by definition but no actual sorting is performed here. The `glossaries` package keeps track of which entries have been defined in an internal list associated with the glossary that contains the given entry. The entry label is appended to the list when it's defined, so the list is already in the correct order. Each time an entry is used in the document, a record is added to the `.aux` file. This also provides a list of all entries that have been indexed, which is naturally in the order required by `sort=use` (order of use). All that is needed is to iterate over the appropriate list and display each entry that has a record.

Now that `TEX` doesn't have to sort the entries, the build process is much faster (Table 14). The only problem here is that the style must be changed to one that doesn't use group headings, as otherwise `TEX` has to determine the correct heading from the sort value. Unlike the previous two examples, the `sort` key isn't altered to a numeric value (because `sort=def` wasn't passed as a package option). This means that a new group will be started with pretty much every entry unless the entries happen to be defined in alphabetical order. So in this example I've switched the style to `mcolindex` and used the `nogroupskip` option. The build process is the same as for Example 23.

This method has a problem with sub-entries. Unlike `makeindex` and `xindy`, there's no hierarchical sorting with this method (because there's no actual sorting) so if a sub-entry isn't defined immediately after its parent is defined then it won't appear immediately after its parent in the glossary. Furthermore, if a sub-entry is used, its parent won't automatically be indexed.

The dummy text contains a number of top-level entries that are duplicated as sub-entries. For example, the book *Ulysses* is defined as:

```
\newglossaryentry{Ulysses}
{name={\tstidxbookfmt{Ulysses}},
 sort={Ulysses},description={}}
}
```

but a sub-entry is defined immediately after:

```
\newglossaryentry{books.Ulysses}
{name={\tstidxbookfmt{Ulysses}}, 
 parent={books}, 
 sort={Ulysses},description={}
}
```

These are then referenced using:

```
\gls{Ulysses}\glsadd{books.Ulysses}
```

The parent entry (`books`) hasn't been used in the dummy text, so it doesn't appear in the glossary. This leads to the rather odd result:

```
Ulysses 2
Ulysses 2
```

The first instance is the top-level entry and the second instance is the sub-entry. Even if the parent entry (`books`) had been used, it would still be separated from its sub-entry (`books.Ulysses`) as it's not defined immediately before it, but is one of the first entries to be defined.

The location ranges (Table 10) have the same problems as for Example 23, but the build time is significantly faster, although it's still slower than using `makerglossaries` (Table 14).

This method is essentially for non-hierarchical symbols that don't have a natural alphabetical order and the available build tools are somehow restricted.

The `glossaries-extra` package extends the base `glossaries` package, providing new features (such as the `category` key and associated attributes) and re-implementing existing methods (such as the abbreviation handling). This package can automatically be loaded by `testidx-glossaries` through the option `extra`. This also ensures that each entry is assigned a category. For example, the *Ulysses* entry is now:

```
\newglossaryentry{Ulysses}
{name={\tstidxbookfmt{Ulysses}}, 
 category={book}, 
 sort={Ulysses},description={}
}
```

(and similarly for the sub-entry). This doesn't alter the indexing, but it can be used to modify the way the entries are displayed.

► Example 28 (testidx-glossaries and `glossaries-extra` in order of definition)

The `glossaries-extra` package provides another way of displaying the list of entries in order of definition. Unlike the above examples, this includes *all* entries, not just the ones that have been indexed. This is done with

```
\printunsrtglossary[<options>]
```

which simply iterates over all defined entries in that glossary, displaying each one in turn according to its handler, so it's similar to Example 27 but doesn't check if the term has been indexed.

This method doesn't create any external indexing files, so `\tstidxmakegloss` isn't needed in this example. The `.tex` files containing the definitions for the dummy entries can be loaded using `\input` or `\loadglsentries`, but it's simpler to just use:

```
\tstidxloadsamples
```

which means you don't have to worry about remembering the file names. However there's a problem here. The `see` key can only be used after the indexing has been initialised (through `\makerglossaries` or `\makenoidxglossaries`). This was a precautionary measure introduced because the cross-reference information can't be indexed before the associated file has been opened, and users who defined entries before using `\makerglossaries` were puzzled as to why the cross-references didn't show up. The error alerts them to the problem.

The simplest solution is to prevent the use of the `see` key in the test entries with the `noseekey` option provided by `testidx-glossaries`.

```
\documentclass{article}

\usepackage[a4paper]{geometry}
\usepackage[T1]{fontenc}
\usepackage{amssymb}
\usepackage[extra,noseekey]{testidx-glossaries}

\tstidxloadsamples

\setglossarystyle{mcolindex}
\renewcommand*\glstreenamefmt{[1]{#1} }
```

```
\begin{document}
\testidx
\printunsrtglossary[nogroupskip]
\end{document}
```

(An alternative is to pass `seenoindex=ignore` to the `glossaries` package or pass `autoseeindex=false` to the `glossaries-extra` package.) The document build process is simply:

```
pdflatex doc
```

Some terms that are used in the original dummy text provided by `testidx` aren't present in the slightly altered version produced by `testidx-glossaries`. (This is why `imakeidx` is missing from the glossary examples listed in Table 6.) These terms are still defined by `testidx-glossaries` to provide an additional test, if required, for the treatment of non-indexed

entries. Since `\printunsrtglossary` includes all entries, `imakeidx` is once again in the index even though it's not in the dummy text.

The most noticeable difference is the absence of page lists (Tables 9, 10, 12) and cross-references (Table 11). No indexing has been performed so there's no record of where the entries have been used. There are no groups (Table 1). This method suffers from the same problem as Example 27 with the sub-entries separated from their parents.

This example is faster than all the other examples using `testidx-glossaries` (Table 14), but the build only requires a single L^AT_EX call and doesn't perform any sorting, so that's hardly surprising. It's slower than Example 1 (Table 13): `makeidx` is a small, simple package and therefore fast to load whereas `glossaries` and `glossaries-extra` are complicated and rely on a number of other packages.

A few seconds can be shaved off the build time by adding

```
\setupglossaries{sort=none}
```

before the entries are defined. (Only available with `glossaries` version 4.30 onwards.) This skips the code used to set up the sort values (such as sanitizing and escaping special characters for `makeindex` or `xindy`).

The iteration handler recognises three special fields, `group`, `location` and `loclist`, which don't have a key provided by default. The `group` value should be a label identifying the letter group, and will only be checked for by the handler if the `group` key is defined. For example:

```
\glsaddstoragekey{group}{}{\glsgroup}
```

The `location` value may contain any valid code that produces the location list. Although the `group` field must have an associated key of the same name for the handler to recognise it, the `location` field can simply be set using `\GlsXtrSetField`.

The `loclist` value must be in the same format as the internal lists provided by `etoolbox` where each item is in the format

```
\glsnoidxdisplayloc{(prefix)}  
 {<counter>} {<encap>} {<location>}
```

for locations, or

```
\glsseeformat[<tag>]{<label>}{}
```

for cross-references. (This is the same command used by `makeindex` and `xindy` when the `see` key is used. The final argument is the location for the benefit of `makeindex` but is always ignored.) The `loclist` value can't be provided as a key since it requires a specific separator used by `etoolbox`. Instead, each item can be added to the list using

```
\glsxtrfieldlistadd{<label>}{<field>} {<item>}
```

The `group` value must be a label (no special characters) because it's used as a hypertarget with the 'hyper' or 'nav' glossary styles. The corresponding title can be set using

```
\glsxtrsetgrouptitle{<label>} {<title>}
```

If not set, the handler will try `\(<label>\)groupname` (for compatibility with `glossaries`) and if that's not defined the label will be used as the title.

If the `location` field is set then that value will be used as the location list otherwise if `loclist` is set then the list given by that field will be iterated over using the same method used by the handler for `\printnoidxglossary` (which is quite primitive, as can be seen in the results for Examples 23, 24 and 27 in Table 10).

It's therefore possible to manually produce a glossary with groups and locations like this:

```
\documentclass{article}  
  
\usepackage{glossaries-extra}  
  
\setglossarystyle{indexgroup}  
\renewcommand*\glstreenamefmt{[1]{#1}}  
  
\glsaddstoragekey{group}{}{\glsgroup}  
  
\glsxtrsetgrouptitle{42}{B}  
\glsxtrsetgrouptitle{D8}{\O}  
  
\newglossaryentry{books}  
 {name={books},group={42},description={}}  
  
\newglossaryentry{books.Dubliners}  
 {name={\emph{Dubliners}},parent={books},  
 description={}}  
\GlsXtrSetField{books.Dubliners}{location}  
 {1--3}  
  
\newglossaryentry{books.Ulysses}  
 {name={\emph{Ulysses}},parent={books},  
 description={}}  
\GlsXtrSetField{books.Ulysses}{location}{2}  
  
\newglossaryentry{OlstykkeStenlose}  
 {name={\O lstykke-Stenl\o se},group={D8},  
 description={}}  
\GlsXtrSetField{OlstykkeStenlose}{location}{8}  
  
\newglossaryentry{Oresund}  
 {name={\O resund},group={D8},description={}}  
\GlsXtrSetField{Oresund}{location}  
 {9, \emph{see also} \O resund Bridge}  
  
\begin{document}  
 \printunsrtglossary  
\end{document}
```

This produces:

```
B
books
  Dubliners 1–3
  Ulysses 2
  Ø
  Ølstykke-Stenløse 8
  Øresund 9, see also Øresund Bridge
```

On the face of it, this method seems contrary to one of L^AT_EX's biggest advantages in its ability to automate cross-referencing and indexing. However, it's just this method that's used by `bib2gls`, which performs two tasks:

1. fetches entry information from a `.bib` file;
2. performs hierarchical sorting, optionally assigns letter groups, collates location lists and writes the entry definitions to a file that can be input by `\GlsXtrLoadResources`.

The first task is akin to using `bibtex` or `biber`. The second task is similar to that performed by `makeindex` or `xindy`.

The L^AT_EX code generated by `bib2gls` has the entry definitions written in the order obtained from sorting, with parent entries defined immediately before their child entries. The information required by `bib2gls` is provided in the `.aux` file, but this needs to be enabled by passing the `record` option to `glossaries-extra`.

An additional build may be required to ensure the locations are up-to-date as the page-breaking may be slightly different on the first L^AT_EX run due to unknown references being replaced with ‘??’, which can be significantly shorter than the actual text produced when the reference is known.

The command `\glsaddall` can't be used in this mode, but it's possible to instruct `bib2gls` to select all entries. By default it only selects those entries that have been indexed and their dependencies (which includes their ancestors). Since only the required entries have been defined and they have been defined in the correct order, the glossary can be displayed using `\printunsrtglossary`.

► Example 29 (testidx-glossaries and bib2gls)

This example uses `bib2gls`, so this needs:

```
\usepackage[bib2gls]{testidx-glossaries}
```

The entries are defined in various `.bib` files provided with `testidx`. The test document is:

```
\documentclass{article}
```

```
\usepackage[a4paper]{geometry}
\usepackage[T1]{fontenc}
```

```
\usepackage[utf8]{inputenc}
\usepackage{amssymb}
\usepackage[bib2gls]{testidx-glossaries}

\tstidxmakegloss

\renewcommand*\glstreenamefmt}[1]{#1}
\renewcommand*\glstreegroupheaderfmt}[1]{%
 \textbf{#1} }

\begin{document}
\testidx
\tstidxprintglossaries
\end{document}
```

The document build process is:

```
pdflatex doc
bib2gls --group doc
pdflatex doc
```

The `--group` switch enables the letter group formation, which is off by default. Note that UTF-8 support is needed with this switch as the groups may contain extended characters. The build times shown in Table 14 use the above build sequence for the `bib2gls` examples. However, the first instance (or when new entries are referenced) will need:

```
pdflatex doc
bib2gls --group doc
pdflatex doc
bib2gls --group doc
pdflatex doc
```

to ensure the location lists are correct. The `.log` file will warn about undefined references on the first run, so build processes that allow for conditional actions can perform a check for these warnings. For example, using `arara` v4.0:

```
% arara: pdflatex
% arara: bib2gls if found ("log", "Warning:
Glossary entry")
% arara: pdflatex if found ("log", "Warning:
Glossary entry")
% arara: bib2gls: {group: on}
% arara: pdflatex
```

The file `testidx-glossaries-samples-ascii.bib` contains definitions using commands for extended characters, for example:

```
@index{elite,
  name={{'e}lite},
  category={word},
  description={group of people regarded
as the best of a particular society
or organisation}
}
```

(The initial `\'e` is grouped to allow it to work with the case-changing `\Gls`.) None of the sample `.bib` files provide a `sort` key, but `bib2gls` has a primitive

TeX interpreter that recognises accent commands, so it determines that the sort value for this entry is `élite`. This means that it can place this word in the E letter group (if appropriate to the collation rule). In the case of `\O resund`, `bib2gls` determines that it belongs to the Ø letter group (again, depending on the rule). Since with `inputenc` Ø is an active character, `bib2gls` uses numeric identifiers as the group labels (to avoid problems with `hyperref`). Although the entry definition is written with the original \Ø used in the `.bib` file, the letter group title is an extended character taken from the `sort` value, which is why either UTF-8 support is needed or the `--group` option should be omitted.

In ASCII mode, `\tstidxmakergloss` selects the `*-ascii.bib` file, whereas with UTF-8 support, this command selects UTF-8 versions (`*-utf8.bib`) and terms such as `élite` no longer need the interpreter. (Only terms containing `\{ }` or `$` are passed to the interpreter.)

The definition of the test interface command `\tstidxmakergloss` varies according to the package options. If you add the `verbose` option, the transcript will list the exact sequence of resource commands. So for this example, the `.log` file includes:

```
\GlsXtrLoadResources[
  src={testidx-glossaries-mathsym},
  group={Maths},
  sort={letter-case},
  selection={recorded and deps and see},
  ignore-fields={description}]
```

This mimics the `prefix` setting used in earlier examples. The maths symbols are defined in the file `testidx-glossaries-mathsym.bib` like this:

```
@symbol{spinderiv,
  name={$\eth$},
  text={\eth},
  category={mathsymbol},
  description={spin-weighted partial
  derivative}
}
```

Entries defined using `@symbol` or `@number` fall back to the `label` if the `sort` field is missing. This means that ð now has a different sort value (`spinderiv`) from the earlier examples where it was either `>eth` or `eth`. This is reflected in Table 4 where the ordering has changed.

The value of the `src` key identifies the `.bib` file (where the extension is omitted). This may be a comma-separated list. The `group` key sets the `group` field for all the selected entries, which overrides the default method of obtaining the group from the entry's sort value. (This will be ignored if `bib2gls` is

run without the `--group` switch.) The `sort` setting `letter-case` indicates case-sensitive letter order.

The `selection` value `recorded and deps and see` instructs `bib2gls` to select all entries that have been indexed (recorded) in the document (through commands like `\gls`) and their dependencies (such as parent entries) and their cross-references. This ensures that sub-entries, such as `books.Ulysses`, have their parent entry listed. The hierarchical sort ensures the sub-entries are defined immediately after their parent entry to keep them together.

The final key `ignore-fields` tells `bib2gls` to ignore the `description` field (to honour the default `nodesc` package option). The `@index` entry type allows a missing description, unlike the `@entry` type (not used in any of the provided `.bib` files) which requires that field.

The above is the first resource command, which instructs `bib2gls` to create a file called `doc.glstex` (where the main document file is called `doc.tex`) with the required definitions in the appropriate order. A separate file is created for each instance of `\GlsXtrLoadResources`. This allows different ordering within sub-units of the glossary (or index). The use of the `group` key assigns the sub-unit to a single group.

The next resource command is quite similar:

```
\GlsXtrLoadResources[
  src={testidx-glossaries-markers},
  group={Markers},
  sort={letter-case},
  selection={recorded and deps and see},
  ignore-fields={description}]
```

This loads the `.bib` file that contains the definitions of all the markers, again using `@symbol`. The L^AT_EX code is written to `doc-1.glstex`.

The third command is:

```
\GlsXtrLoadResources[
  src={testidx-glossaries-numbers},
  sort={integer},
  selection={recorded and deps and see},
  ignore-fields={description}]
```

This loads the `.bib` file that contains the definitions of all the numbers in the form:

```
@number{10,
  name={10},
  category={number},
  description={ten}
}
```

The `sort` key has been set to `integer` to order these entries numerically. This automatically assigns them to the ‘Numbers’ group so no `group` option is used here. The L^AT_EX code for this resource set is written to `doc-2.glstex`.

The final resource command is:

```
\GlsXtrLoadResources[
  src={testidx-glossaries-samples,
    testidx-glossaries-samples-utf8,
    testidx-glossaries-nodiglyphs-utf8},
  selection={recorded and deps and see},
  ignore-fields={description}]
```

The .bib files listed in `src` vary according to the `testidx-glossaries` package options and document encoding. There's no `sort` option in this resource set. The `glossaries` package loads `tracklang` [11] (described in a previous issue of *TUGboat* [6]). If a document language is detected, `glossaries-extra` will use the `tracklang` interface to write the locale information to the `.aux` file, which `bib2gls` will detect and will use as the default sort. If there is no document language (as in this case), `bib2gls` will fall back on the operating system's locale. In my case, this is `en-GB` so the entries will be sorted according British English. Another user with a different locale may find that the resulting letter groups are different to those shown in Table 1. The optional argument of `\tstidxmakegloss` is appended to this final instance of `\GlsXtrLoadResources` (but not to any of the others), so to replicate this example, you can do

```
\tstidxmakegloss[sort=en-GB]
```

(or just `sort=en`).

The non-native (for English) letters Ø and Ł have been combined into a single group after Z. The rules used by `sort=<locale>` are in the form `<ignore chars> <char group 1> <char group 2> ...`. (You can see the rule in the transcript by running `bib2gls` with the `--debug` switch.) Any characters that don't appear in the rule (such as Ø and Ł) are always placed at the end of the alphabet. `bib2gls` determines the letter group title from the first entry in the group.

The remaining letter groups in this example are sensible for this locale as they are included in the `en` rule. Đ is placed between D and E, and ß is treated as 'ss' (Table 8).

The sort value for each entry is converted to a set of collation keys, where each key is an integer representing a 'letter' as defined by the collation rule. The letter may be more than one character, for example, if the rule includes digraphs or trigraphs. Ignored characters aren't included in the key set. The comparison is performed on this key set rather than on the sort string.

Group titles are determined by taking the first collation key from the set and looking up the corresponding sub-string from the sort value. This sub-string is then converted to lower case and any modifiers are stripped using a normalizer (where possible). If the result is considered equivalent to the original

sub-string according to the collator, then the normalized version is considered the group title and the first character is converted to upper case (except for the Dutch 'ij', which is converted to IJ, see Example 33). For example, the first letter of élite is 'é' which is normalised to 'e'. Since the sort rule considers é and e to belong to the same letter group, the group title becomes E. In the case of Øresund, the result of the normalisation 'o' doesn't match the original, so the group title is Ø.

The multiple encap (Table 9) generates a warning from `bib2gls`. It gives precedence to the first non-default of the conflicting set (`tstidxencapi`, in this case). Precedence can be given to a different encap through the `--map-format` switch.

The range interruption has been moved before the start of the explicit range (Table 10) but the explicit range 2–5 (created with the open and close formats) hasn't been merged with the individual locations 1 and 6 on either side of it. The `notestencaps` option doesn't use any of the test encaps, so with

```
\usepackage[bib2gls,notestencaps]
  {testidx-glossaries}
```

the interrupting entry now has the same format as the explicit range. This means that it can be absorbed into the range, but an explicit range doesn't merge with neighbouring locations, so the location list becomes 1, 2–5, 6.

The space and hyphen characters are in the `<ignore chars>` part of the rule. This means that the locale sorting naturally used by Java (in which `bib2gls` is written) is typically letter order. To implement word-ordering, the sort value is split on word boundaries and joined with | (which is usually in its own letter group before digits). For example, 'sea lion' becomes `sea|lion|` (there's always a final marker so 'seal' becomes `seal|`). This ensures that `bib2gls` defaults to word ordering, matching `makeindex` and `xindy` (Table 7). Java's word iterator doesn't consider hyphens as word boundaries so 'yo-yo' becomes `yo-yo|`.

► Example 30 (testidx-glossaries, bib2gls and letter order)

In this example, the insertion of the break points is disabled:

```
\tstidxmakegloss[sort=en-GB,break-at=none]
```

This results in letter ordering (Table 7). Note that this isn't the same as `sort=letter-case` which simply sorts according to the Unicode values rather than according to a rule.

The 'L' letter group includes the -l and -L switches (Table 1), but these are in a different order (Table 3) than the previous example. In this case

`-1 (makeindex)` appears at the start of the group whereas in the previous case it came between `-L icelandic (xindy)` and `-L polish (xindy)`.

► **Example 31** (`testidx-glossaries`, `bib2gls` and `Icelandic`)

For comparison with Example 15, this example sorts according to the Icelandic alphabet:

```
\tstidxmakergloss[sort=is]
```

This correctly identifies all the Icelandic letter groups as shown in Table 1. (There's no Ó or Ý letter group as there are no terms starting with those letters.) The non-native letters C, Æ, Q, W, Z and Ł have also been assigned their own letter groups. The ordering of 'resume' and 'résumé' (Table 6) is different from the previous example since É comes after E in the Icelandic alphabet (and are considered separate letters). They are no longer collation-level homographs. The non-native ß is treated as 'ss' (Table 8).

► **Example 32** (`testidx-glossaries`, `bib2gls` and `Hungarian`)

For comparison with Example 16, this example sorts according to the Hungarian alphabet:

```
\tstidxmakergloss[sort=hu]
```

In addition to the basic Latin letters A–Z, the Hungarian alphabet also has Á, Cs, Dz, Dzs, É, Gy, Í, Ly, Ny, Ó Ö, Ő, Sz, Ty, Ú, Ü, Ű and Zs. The sample entries don't include any terms starting with Cs, Gy, Ny, Ő, Sz, Ty, Ü, Ű or Zs. Of the other letters, only Ly and Ö have correctly formed letter groups (Table 1). The non-native letters Ð and Æ have formed separate groups, ß has been treated as 'sz' rather than 'ss' (Table 8), and Ø and Ł are collected at the end of the alphabet as they aren't in the rule-set.

This has more success than `xindy` at forming a digraph letter group (Ly) but has missed the Dz digraph and Dzs trigraph.

Since I have Java 8 installed, the above examples are using the locale rules from the CLDR (Common Locale Data Repository). The results may differ with Java 7 which can only use the locale information provided with the JRE (Java Runtime Environment). The locale identifier can include a variant as well as a region, for example, `sort=de-CH-1996` indicates Swiss German new orthography.

► **Example 33** (`testidx-glossaries`, `bib2gls` with custom rules)

This example requires some customisation, so I can't use the convenient `\tstidxmakergloss`. I need to let `testidx-glossaries` know this with the `manual` option to prevent an error occurring:

```
\usepackage[bib2gls,manual]{testidx-glossaries}
I also need to explicitly use
\printunsrtglossary
```

So far, the maths group (where it has been formed) only contains symbols such as α . There are some other maths terms that have a natural alphabetic ordering (such as $f(\vec{x})$ and E) which have been placed in the letter groups. This example gathers them all together into a single group. As mentioned earlier, terms like α have the `category` set to `mathsymbol`. The other mathematical terms are in `testidx-glossaries-samples.bib` and have the `category` set to `math`. It's possible to apply a filter so that only these terms are selected:

```
\GlsXtrLoadResources[
  src={testidx-glossaries-mathsym,
    testidx-glossaries-samples},
  group={Maths},
  sort={letter-case},
  sort-field={name},
  match-op={or},
  match={{category=mathsymbol},{category=math}},
  selection={recorded and deps and see},
  ignore-fields={description}]
```

I've set the sort field to `name`, which means that `bib2gls` will try to interpret the TeX code. It recognises standard maths commands like `\alpha` and can also detect a limited number of packages, such as `amssymb`. This means that the sort code for ð becomes the Unicode character F0 (eth).

The markers use the same code shown in Example 29. After that is the number group, which is much the same, but for illustrative purposes, I've inverted the number ordering:

```
\GlsXtrLoadResources[
  src={testidx-glossaries-numbers},
  sort={integer-reverse},
  selection={recorded and deps and see},
  ignore-fields={description}]
```

Next I want to create a group for the switches. The switches also occur as sub-entries (under the name of the application), so I need to select those switches that don't have a parent:

```
\GlsXtrLoadResources[
  src={testidx-glossaries-samples},
  group={Switches},
  sort={letter-nocase},
  match-op={and},
  match={{category=applicationoption},{parent={})),
  selection={recorded and deps and see},
  ignore-fields={description}]
```

I've used the case-insensitive letter sort which first converts the sort key to lower case and then behaves like `letter-case`.

The remaining entries are the alphabetic terms. The terms that have been previously selected will be ignored (with a warning) as duplicates. I've used a custom sort rule here:

```
\GlsXtrLoadResources[
  src={testidx-glossaries-samples,
    testidx-glossaries-samples-utf8,
    testidx-glossaries-nodiglyphs-utf8},
  selection={recorded and deps and see},
  ignore-fields={description},
  max-loc-diff=3,
  sort=custom,
  sort-rule={' ' < ',' < '(' < ')' < '/' < ']' < '-'
  < a,A & AE,\string\uE6,\string\uC6 % \ae
  & \string\uE1,\string\uC1 % '\a
  & \string\uE4,\string\uC4 % "\a
  & \string\uE5,\string\uC5 % \aa
  < b,B
  < c,C & \string\u107,\string\u106 % '\c
  < d,D < dd,Dd,DD
  < dz,Dz,DZ < dzs,Dzs,DZS
  < \string\uF0,\string\uD0 % \dh
  < e,E & \string\uC9,\string\uE9
  < f,F < ff,Ff,FF < g,G < h,H
  < i,I & \string\uED,\string\uCD % '\i
  < ij,IJ < j,J < k,K < l,L < ll,Ll,LL
  < ly,Ly,LY < m,M < n,N < ng,Ng,NG
  < o,O & \OE,\string\u153,\string\u152 % \oe
  & \string\uF6,\string\uD6 % "\o
  < p,P < q,Q < r,R
  < s,S & \SS,\string\uDF
  & \string\u15B,\string\u15A % '\s
  < t,T
  < th,\string\uFE,Th,TH,\string\uDE % \th
  < u,U & \string\uFA,\string\uDA % '\U
  < v,V < w,W < x,X < y,Y
  < z,Z & \string\u17C,\string\u17B % '\.z
  < \string\uF8,\string\uD8 % '\o
  < \string\u142,\string\u141 % '\l
}
]
```

The `sort=custom` option requires the `sort-rule` key to be also set. Extended characters can be identified with `\u<hex>` but `\string` is needed to prevent expansion when the information is written to the `.aux` file. With X_ƎT_ƎX or Lua_ƎT_ƎX the characters can be written directly.

This rule has only a limited number of punctuation characters for brevity. Extra characters should be added to the rule if required. This is the only example that successfully creates the Dzs trigraph letter group (Table 1). There are also letter groups for the Welsh Dd, Ff, Ll and Ng digraphs, the Dutch IJ digraph, and the Hungarian Dz and Ly digraphs (although the word beginning with 'ly' is actually Polish). There's also a group for both þ and the Th digraph. The eszett ß has been treated as 'ss' (Table 8).

I've listed the hyphen immediately before A (and after the break point marker), which affects the ordering of the compound words (Table 7). This

also means that 'recover' and 're-cover' are no longer collation-level homographs (Table 6) since the hyphen is no longer ignored.

The additional `seealso` key provided by v1.16 of glossaries-extra allows `bib2gls` to treat the `see` and `seealso` cross-references differently. (An entry may have one or the other of those fields, but not both with `bib2gls`.) The `seealso` field can be positioned at the start of the location list using the resource option `seealso=before` or omitted entirely using `seealso=omit`. The default setting is `seealso=after`, which puts it at the end of the list. The separator between the list and the cross-reference is given by `\bibglssealsosep`, which can be redefined after the resources are loaded. In this example, I've done:

```
\renewcommand*{\bibglssealsosep}{ }
\renewcommand*{\glsxtruseseealsoformat}[1]{%
  (\glsseeformat[\seealonename]{#1}{})}
```

This puts the 'see also' cross-references in parentheses, but doesn't affect the 'see' cross-references. For example, 'range separator' is defined with the `see` field, and the result is 'range separator *see* location list', but 'padding' is defined with the `seealso` field, so the result is 'padding 2 (*see also* filler)'.

Implicit ranges are formed from consecutive locations. This can lead to some ragged location lists, such as 1, 2, 4, 5, 7. A tidier approach is to show this as 1–7 passim, where 'passim' indicates the references are scattered here and there throughout the range. The `max-loc-diff` option indicates the maximum difference between two locations to consider them consecutive. The default value is 1, which means that 2 and 3 are consecutive but 2 and 4 aren't. I've set the value to 3 in this example, which means that the location list 2, 5, 6 can be tidied into 2–6 passim. The ragged list for 'paragraph' (Table 9) can't be tidied as there are different encaps. The 'passim' suffix can be altered or removed as required.

► Example 34 (testidx-glossaries, `bib2gls` and non-standard page numbering)

This example uses the same custom `\tally` command from Example 22 for the page numbering. The only modification to Example 33 is the addition of:

```
\usepackage[notext]{stix}
```

and the definition of `\tally` and `\thepage` from Example 22.

`bib2gls` will allow any location format. If it can deduce an associated numeric value, it will try to determine if a range can be formed, otherwise the location will be considered an isolated value that can't be concatenated. (With `glossaries-extra`, it's possible to override the normal location value when

using `thevalue` with `\gls...`, for example, `\glsadd[thevalue={Suppl.\ info.}]{<label>}.` One of the patterns `bib2gls` checks for is `\(<csname>\{<num>\}`, which it interprets as having the numeric value `<num>`. The regular expression for `<num>` can detect roman numerals (I, II, ... or i, ii, ...) or numeric values or single alphabetical characters.

The alphabetic test uses `\p{javaUpperCase}` for the upper case version or `\p{javaLowerCase}` for the lower case version which not only matches A, B, etc., or a, b, etc., but also matches alphabetic characters in other scripts, such as A, B, etc. The numeric value representing the location is obtained from the Unicode value. For example, Latin A has the value 65 whereas Cyrillic A has the value 1040.

The numeric test uses `\p{javaDigit}` to match a digit, which means it not only matches the digits 0, 1, 2, etc., but also digits from other scripts, such as the Devanagari numbering system ०, १, २, etc.

The results from this example are much the same as the previous example except for the page number representation (Tables 9, 10, 11 and 12).

5 Extending the dummy text

New blocks can be added using `\tstidxnewblock`. For example:

```
\tstidxnewblock{The \tstidxword{cat} sat
on the \tstidxword{mat}. The
\tstidxphrase{man in the moon} fell off
the \tstidxphrase{four-poster bed}.}
```

The starred version can be used to capture the block number in a control sequence:

```
\tstidxnewblock*{\moonblock}{The
\tstidxword{cat} sat on the \tstidxword{mat}.
The \tstidxphrase{man in the moon} fell off
the \tstidxphrase{four-poster bed}.}
```

You can then display just this block with

```
\testidx[\moonblock]
```

There are other commands as well, including commands for UTF-8 terms. For example:

```
\tstindexutfword{ch\^ateau}[chateau]{ch\u00e2teau}
```

The first argument is the ASCII version and the final argument is the UTF-8 version. The optional argument is the label, which is only used by `tstidxglossaries`. If you want this support for the glossaries package, you'll need to define the terms as well:

```
\tstidxnewword{cat}{feline animal}
\tstidxnewword{mat}{piece of material
placed on the floor}
\tstidxnewphrase{man in the moon}{pareidolic
image seen in the moon}
\tstidxnewphrase{four-poster bed}{type of bed}
```

The UTF-8 example needs to be defined as follows:

```
\tstidxnewutfword{chateau}{ch\^ateau}{ch\u00e2teau}
{castle}
```

where the first argument is the label.

To integrate this with `\tstidxmakegloss`, just add the definition file name to the comma-separated list given by `\tstidxtexfiles`. For example (using `etoolbox`), if the terms are defined in the file `my-samples.tex`:

```
\appto{\tstidxtexfiles}{,my-samples}
```

With `bib2gls`, the definitions will need to go in a `.bib` file. For example:

```
@index{cat,
  category={word},
  description={feline animal}
}
@index{fourposterbed,
  category={phrase},
  name={four-poster bed},
  description={type of bed}
}
```

(Note that the hyphen and space are stripped from the name to create the label. The `name` field may be omitted if it's identical to the label.) The UTF-8 support is dealt with by having two separate `.bib` files. One contains the ASCII version:

```
@index{chateau,
  category={word},
  name={ch\^ateau},
  description={castle}
}
```

and the other contains the UTF-8 version:

```
@index{chateau,
  category={word},
  name={ch\u00e2teau},
  description={castle}
}
```

These can also be integrated into `\tstidxmakegloss` as follows. The `.bib` file that doesn't require UTF-8 support (the one containing 'cat' in the above) needs to be added to `\tstidxbasebibfiles` (a comma-separated list). For example, if that file is called `my-samples.bib` then:

```
\appto{\tstidxbasebibfiles}{,my-samples}
```

The UTF-8 file (the one containing `ch\u00e2teau`) needs to be added to `\tstidxutfbibfiles`. For example, if the file is called `my-samples-utf8.bib`:

```
\appto{\tstidxutfbibfiles}{,my-samples-utf8}
```

and the corresponding ASCII file needs to be added to `\tstidxasciibibfiles`. For example, if the file is called `my-samples-ascii.bib`:

```
\appto{\tstidxasciibibfiles}{,my-samples-ascii}
```

Table 1: Letter groups**Example Group ordering**

1, 5, 18, 19	Symbols Numbers A (inc. æ, Á, Ä, Å) B C (inc. Ć) D (inc. δ) E (inc. é) F G H I (inc. Í) J K L (inc. Ł) M N O (inc. œ, Ø, Ö) P Q R S (inc. Š) T (inc. þ, Þ) U (inc. Ú) V W X Y Z (inc. Ž)
2	Symbols Numbers 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
3	Symbols Numbers A (inc. æ, α, Á, Ä, Å) B (inc. β) C (inc. Ć) D (inc. δ) E (inc. é, δ) F G (inc. γ) H I (inc. Í) J K L (inc. Ł) M N O (inc. œ, Ø, Ö) P (inc. δ) Q R S (inc. Š, Σ) T (inc. þ, Þ) U (inc. Ú) V W X Y Z (inc. Ž)
4	Symbols Numbers A (inc. α) B (inc. β) C D E (inc. δ) F G (γ) H I J K L M N O P (inc. δ) Q R S (inc. Σ) T U V W X Y Z
6	Symbols Numbers 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 Other (Á, Ä, Å, Í, Ö, Ø, Ú, þ, æ, é, δ, þ) Other (Ć) Other (Ł, œ, Š, Ž)
7–9	Symbols A (inc. æ) B C D (inc. δ) E (inc. é) F G (inc. -g) H I J K L (inc. -1, -L) M (inc. -M) N O (inc. Á, Ä, Å, Í, œ, Ø, Ú, Ö) P Q R S T U V W X Y Z P
10	Symbols A (inc. æ, α) B (inc. β) C D (inc. δ) E (inc. é, δ) F G (inc. -g, γ) H I J K L (inc. -L, -1) M (inc. -M) N O (inc. Á, Ä, Å, Í, œ, Ø, Ú, Ö) P (inc. δ) Q R S (inc. Σ) T U V W X Y Z P
11	Symbols Numbers A (inc. æ) B C D (inc. δ) E (inc. é) F G (inc. -g) H I J K L (inc. -1, -L) M (inc. -M) N O (inc. Á, Ä, Å, Í, œ, Ø, Ú, Ö) P Q R S T U V W X Y Z P
12	Symbols Numbers A (inc. æ, α) B (inc. β) C D (inc. δ) E (inc. é, δ) F G (inc. -g, γ) H I J K L (inc. -1, -L) M (inc. -M) N O (inc. Á, Ä, Å, Í, œ, Ø, Ú, Ö) P (inc. δ) Q R S (inc. Σ) T U V W X Y Z P
13	Symbols Numbers A (inc. æ, Á, Ä, Å) B C (inc. Ć) D (inc. δ) E (inc. é) F G (inc. -g) H I (inc. Í) J K L (inc. -1, -L, Ł) M (inc. -M) N O (inc. œ, Ø, Ö) P Q R S (inc. Š) T (inc. þ, Þ) U (inc. Ú) V W X Y Z (inc. Ž)
14, 21, 22	Symbols Markers Maths Numbers A (inc. æ) B C D (inc. δ) E (inc. é) F G (inc. -g) H I J K L (inc. -1, -L) M (inc. -M) N O (inc. Á, Ä, Å, Í, œ, Ø, Ú, Ö) P Q R S T U V W X Y Z P
15 (Icelandic)	Symbols Markers Maths Numbers A (inc. Á) B C D δ E (inc. é) F G (inc. -g) H I (inc. Í) J K L (inc. -1, -L) M (inc. -M) N O P Q R S T U (inc. Ú) V W X Y Z P Æ (inc. Ä, œ) Ö (inc. Ø) Å
31 (Icelandic)	Maths Markers Numbers A (inc. Á, Å) Á B C (inc. Ć) D D E É F G (inc. -g) H I Í J K L (inc. -L, -1) M (inc. -M) N O Æ P Q R S (inc. Š) T U Ú V W X Y Z (inc. Ž) P Æ Ö (inc. Ø) Ł
16 (Hungarian)	Symbols Markers Maths Numbers A (inc. Á) B C D (inc. dz, dzs) E (inc. é) F G (inc. -g) H I (inc. Í) J K L (inc. -1, -L, ly) M (inc. -M) N O (inc. Á, Ä, Ø, þ, æ, δ, þ) Ö P Q R S T U (inc. Ú) V W X Y Z
32 (Hungarian)	Maths Markers Numbers A (inc. æ Á Á Å) B C (inc. Ć) D (inc. dz, dzs) D E (inc. é) F G (inc. -g) H I (inc. Í) J K L (inc. -L, -1) Ly M (inc. -M) N O Ö Æ P Q R S (inc. Š) T (inc. þ, Þ) U (inc. Ú) V W X Y Z (inc. Ž) Ø (inc. Ł)
17	Markers Maths Numbers A (inc. æ, Á, Ä, Å) B C (inc. Ć), D (inc. dd and dzs) Dz E (inc. é) F (inc. ff) G (inc. -g) H I (inc. Í) IJ J K L (inc. -1, -L, Ł, ly) IL M (inc. -M) N (inc. Ng) O (inc. œ, þ, δ, Ø, þ) P Q R S (inc. Š) T U (inc. Ú) V W X Y Z (inc. Ž)
20	Markers Maths Numbers A (inc. æ, Á, Ä, Å) B C (inc. Ć) D (inc. δ) E (inc. é) F G (inc. -g) H I (inc. Í) J K L (inc. -1, -L, Ł) M (inc. -M) N O (inc. œ, Ø, Ö) P Q R S (inc. Š) T (inc. þ, Þ) U (inc. Ú) V W X Y Z (inc. Ž)
23, 24	Symbols A (inc. æ, Á, Ä, Å) B C (inc. Ć) D (inc. δ) E (inc. é) F G H I (inc. Í) J K L (inc. Ł) M N O (inc. Ø, Ö) P Q R S (inc. Š) T (inc. þ, Þ) U (inc. Ú) V W X Y Z (inc. Ž)
29, 30	Maths Markers Numbers A (inc. æ, Á, Ä, Å) B C (inc. Ć) D D E (inc. é) F G (inc. -g) H I (inc. Í) J K L (inc. -L, -1) M (inc. -M) N O (inc. œ, Ö) P Q R S (inc. Š) T (inc. þ, Þ) U (inc. Ú) V W X Y Z (inc. Ž) Ø (inc. Ł)
33, 34	Maths Markers Numbers Switches A (inc. æ, Á, Ä, Å) B C (inc. Ć) D Dd Dz Dzs D E (inc. é) F Ff G H I (inc. Í) IJ J K L Ll Ly M N Ng O (inc. œ, Ö) P Q R S (inc. Š) T Th (inc. þ, Þ) U (inc. Ú) V W X Y Z (inc. Ž) Ø Ł
25–28	<i>no groups or all entries in one group</i>

Table 2: Symbols**Example Symbol group contents**

1, 5, 18, 19	switches markers maths
2	switches markers maths non-ASCII (Ä, Ö, Á, Ć, Í, Š, Ú, é, Ž, Ł, Ø, Þ, æ, ð, œ, Å, þ)
3	switches
4	switches non-ASCII (Ä, Ö, Á, Ć, Í, Š, Ú, é, Ž, Ł, Ø, Þ, æ, ð, œ, Å, þ)
6	switches markers maths
7–9	numbers markers maths UTF-8 (Ć, Ł, Š, Ž)
10	numbers UTF-8 (Ć, Ł, Š, Ž)
11	markers maths UTF-8 (Ć, Ł, Š, Ž)
12, 14, 15,	UTF-8 (Ć, Ł, Š, Ž)
21, 22	
13	markers maths
16	UTF-8 (Ć, Ł, œ, Š, Ž)
23, 24	switches markers maths numbers
17, 20, 25–34	<i>group missing</i>

Table 3: Switches**Example Switches ordering**

1–6, 18, 19, 23, 24	-L -M -g -l
7–9,	-g -l -L -M
11–17,	
20–22, 30,	
33, 34	
10, 29, 31, 32	-g -L -l -M
25–28	<i>order of definition</i>

Table 4: Maths**Example Maths ordering**

1, 2, 5–11, 13–24	α ($>\alpha$), β ($>\beta$), \eth ($>\eth$), γ ($>\gamma$), ∂ ($>\partial$), \sum ($>\sum$).
3, 4, 12	α (α), β (β), \eth (\eth), γ (γ), ∂ (∂), \sum (\sum).
29–32	α (α), β (β), γ (γ), ∂ (∂), \eth (\eth), \sum (\sum).
33, 34	E (45), $f(\vec{x})$ (66 28 78 20D7 29), n (6E), \eth (F0), ∂ (2202), \sum (2211), α (1D6FC), β (1D6FD), γ (1D6FE).
25–28	<i>order of definition</i>

Table 5: Numbers**Example Number ordering**

1–6,	2, 10, 16, 42, 100
11–24,	
29–32	
7–10	10, 100, 16, 2, 42
33, 34	100, 42, 16, 10, 2
25–28	<i>order of definition</i>

Table 6: Collation-level homographs**Example Ordering**

1–6	<code>imakeidx</code> package, <code>index</code> , <code>\index</code> , indexing application
7–17	<code>imakeidx</code> package, <code>\index</code> , indexing application (' <code>index</code> ' omitted)
18, 19, 23,	illustration, <code>\index</code> , <code>index</code> , indexing
24, 29–34	application
20–22	illustration, <code>\index</code> , indexing application (' <code>index</code> ' omitted)
1–6, 18, 19, 23, 33,	range separator, <code>re-cover</code> , <code>recover</code> , reference
34	
7–17, 24,	range separator, <code>recover</code> , <code>re-cover</code> , reference
20–22,	
29–32	
1, 3, 5, 7–12,	repetition, <code>resume</code> , <code>résumé</code> , rhinoceros
14–17, 21,	
22, 31	
18, 19, 23,	repetition, <code>résumé</code> , <code>resume</code> , rhinoceros
24, 29, 30,	
32–34	
2, 4	(start of group) <code>résumé</code> , Rødovre, raft, ..., repetition, <code>resume</code> , rhinoceros
6	repetition, <code>resume</code> , rhinoceros, ..., roundabout, <code>résumé</code> , Rødovre
13, 20	repetition, <code>résumé</code> , rhinoceros (' <code>resume</code> ' omitted)
25–28	<i>order of definition</i>

Table 7: Compound words**Example Ordering**

1–4, 6, 7, 9–22, 29, 31–34	sea, sea lion, seaborne, seal, sealant gun
5, 8, 24, 30	sea, seaborne, seal, sealant gun, sea lion
23	sea, sealant gun, sea lion, seaborne, seal
1–4, 6, 7, 9–22, 29, 31–34	vice admiral, vice chancellor, vice versa, vice-president, viceregal, viceroy
5	vice-president, vice admiral, vice chancellor, viceregal, viceroy, vice versa
8, 24, 30	vice admiral, vice chancellor, vice-president, viceregal, viceroy, vice versa
23	vice chancellor, viceregal, vice versa, vice admiral, vice-president, viceroy
1–6, 18, 19, 33, 34	yawn, yo-yo , yoghurt
7–17, 20–22, 24, 29–32	youthful, yo-yo , yuck
23	yoghurt, yo-yo , youthful
25–28	<i>order of definition</i>

Table 8: Eszett ('Aßlar')**Example Ordering**

1, 3, 5, 13, 15, 17–20, 23, 24, 29–31, 33, 34	assailed, Aßlar , astounded
2, 4	(start of group) Aßlar , aardvark
6	attributes, Aßlar (end of group)
7–12, 14, 16, 21, 22	anonymous reviewer, Aßlar , applications
32	astounded, Aßlar , attaché case
25–28	<i>order of definition</i>

Table 9: Multiple encap ('paragraph')**Example Location List**

1–6	2, 2, 2, 2, 3, 5
7–10	2, 2, 3, 5
11–16	2, 3, 5
17	2, 3, 4, 6
18	2, 3, 3, 3, 3, 4, 6
19, 25	2, 3, 4, 6
20, 21, 26,	2, 3, 4, 6
29–33	
22, 34	□, □, □, □
23, 24, 27	2, 2, 2, 2, 3, 6
28	<i>locations missing</i>

Table 10: Explicit range interruption ('range')**Example Location List**

1–6	2, 1–4, 6
7–16	1–4, 2, 6
17	1, 2, 6
18, 19, 25	3, 1–6
20, 21, 26	1–6, 3
22	□–□, □
23, 24, 27	1, 2, 2–5, 6
29–33	1, 3, 2–5, 6
34	□, □, □–□, □
28	<i>locations missing</i>

Table 11: Cross-reference interruption ('lyuk')**Example Location List**

1–6,	1, <i>see also</i> digraph, 3
7–17	1, 3, <i>see also</i> digraph
18–21, 25,	2, 3, <i>see also</i> digraph
26, 29–32	
22	□, □, <i>see also</i> digraph
23, 24, 27	<i>see also</i> digraph, 1, 3
33	2, 3 (<i>see also</i> digraph)
34	□, □ (<i>see also</i> digraph)
28	<i>locations missing</i>

Table 12: Ragged page list ('block')**Example Location List**

1–16	1, 2, 4–6
17–21,	2, 5, 6
23–27,	
29–32	
22	□, □, □
33	2–6 <i>passim</i>
34	□–□ <i>passim</i>
28	<i>locations missing</i>

Table 13: Build time (testidx)

Example	Elapsed real time	External tool
1	0:00.73	<code>makeindex</code>
2	0:00.64	<code>makeindex</code>
3	0:00.64	<code>makeindex</code>
4	0:00.69	<code>makeindex</code>
5	0:00.56	<code>makeindex</code>
6	0:00.61	<code>makeindex</code>
7	0:01.17	<code>xindy</code>
8	0:01.13	<code>xindy</code>
9	0:01.07	<code>xindy</code>
10	0:01.13	<code>xindy</code>
11	0:01.12	<code>xindy</code>
12	0:01.32	<code>xindy</code>
13	0:01.38	<code>xindy</code>
14	0:01.19	<code>xindy</code>
15	0:01.13	<code>xindy</code>
16	0:01.17	<code>xindy</code>
17	0:02.43	<code>xindy</code>

Table 14: Build time (testidx-glossaries)

Example	Elapsed real time	External tool
18	0:02.45	<code>makeglossaries-lite</code>
	0:02.08	<code>makeindex</code> (explicit)
19	0:02.42	<code>makeglossaries</code> (<code>makeindex</code>)
20	0:03.19	<code>makeglossaries</code> (<code>xindy</code>)
21	0:03.18	<code>makeglossaries</code> (<code>xindy</code>)
22	0:03.29	<code>makeglossaries</code> (<code>xindy</code>)
23	3:31.69	<code>none</code>
24	3:34.38	<code>none</code>
25	0:02.24	<code>makeglossaries</code> (<code>makeindex</code>)
26	0:03.18	<code>makeglossaries</code> (<code>xindy</code>)
27	0:03.79	<code>none</code>
28	0:01.57	<code>none</code>
29	0:05.33	<code>bib2gls</code>
30	0:05.08	<code>bib2gls</code>
31	0:05.03	<code>bib2gls</code>
32	0:05.06	<code>bib2gls</code>
33	0:06.04	<code>bib2gls</code>
34	0:05.50	<code>bib2gls</code>

References

- [1] STI Pub Companies. The `stix` package, 2015. ctan.org/pkg/stix.
- [2] Alan Jeffrey and Frank Mittelbach. The `inputenc` package, 2015. ctan.org/pkg/inputenc.
- [3] Philipp Lehman and Joseph Wright. The `etoolbox` package, 2015. ctan.org/pkg/etoolbox.
- [4] Frank Mittelbach, Robin Fairbairns, and Werner Lemberg. The `fontenc` package, 2016. ctan.org/pkg/fontenc.
- [5] American Mathematical Society. The `amssymb` package, 2013. ctan.org/pkg/amsmath.
- [6] Nicola Talbot. Localisation of T_EX documents: `tracklang`. *TUGboat*, 37(3):337–351, 2016. tug.org/TUGboat/tb37-3/tb117talbot.pdf.
- [7] Nicola Talbot. `bib2gls`: A command line Java application to convert .bib files to `glossaries-extra.sty` resource files, 2017.
- [8] Nicola Talbot. The `glossaries-extra` package, 2017. ctan.org/pkg/glossaries-extra.
- [9] Nicola Talbot. The `glossaries` package, 2017. ctan.org/pkg/glossaries.
- [10] Nicola Talbot. The `testidx` package, 2017. ctan.org/pkg/testidx.
- [11] Nicola Talbot. The `tracklang` package, 2017. 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/>

A note on \linepenalty

Udo Wermuth

Abstract

This article analyzes the effect of the line-breaking parameter `\linepenalty`. First, its rôle in the problem of typesetting a text in one line or in two lines is studied theoretically. Then the effect of different values for `\linepenalty` are demonstrated for longer paragraphs. Finally, `\linepenalty` is compared to `\looseness`.

1 Introduction

The line-breaking algorithm of TeX selects a shortest path in a network of possible breakpoints ([2] or the reprint in [7], p. 107) using a cost function that calculates *demerits*. For every line, four values are involved in the computation of its demerits and then the sum of all line demerits stands for the *total demerits* of a paragraph. Three of the four values are directly related to the characteristics of the created line, while the fourth value is a constant for all lines of a paragraph: the `\linepenalty`. In TeX78 this constant was hard-coded into the program but with TeX82 it became an integer parameter that the user can change ([5], or the reprint in [6], pp. 273–274). The plain format sets the default value of `\linepenalty`.

The next section gives a brief overview of the rules by which TeX’s line-breaking algorithm calculates the demerits. It also introduces some notation in accordance with [9] but as there are only a few symbols in this article the conventions are not repeated here. Section 3 analyzes what happens to a text, for example, a heading, that can be typeset either in one line or in two lines. Even this simple case gets rather complex and Section 4 summarizes some of the theoretical results and applies them to normal text. The fifth section looks at longer paragraphs and finds some insights when the value of `\linepenalty` is changed. Section 6 compares the effects of the two integer parameters `\linepenalty` and `\looseness`.

The phrases “(possible) solution” or “path in the network”, etc., refer to the network of line breaks [2, Fig. 13] that exists for the given text. They do not mean either that this is the shortest path in the network and thus the typeset solution of the line-breaking problem, or that the path is part of the network which is actually created by TeX’s line-breaking algorithm. Usually, this algorithm removes

the initial segment of a path and thus the path from its memory as soon as it learns that this beginning cannot lead to the shortest path. The phrases here state only that a certain path exists in the whole network.

2 Calculation of demerits

Section 2 of [8] contains a detailed description of the rules that TeX uses to compute the demerits. In order to introduce the notation for this article a brief summary of these rules follows.

The formula ([3], p. 98) that computes the demerits of a line, stated as Λ , is

$$\Lambda = (\lambda + \beta)^2 + \operatorname{sgn}(\pi)\pi^2 + \delta \quad (1)$$

which names the four parameters with Greek letters.

λ is the `\linepenalty`, a constant value set in the plain TeX format to 10.

The badness assigned to the line is called β . The badness is itself the result of a computation which looks at the ratio of used and available stretch- or shrinkability in the line ([3], p. 97). It is a nonnegative number $\leq \operatorname{\textit{pretolerance}}$ in the first pass; otherwise $\leq \operatorname{\textit{tolerance}}$.

Depending on the type of line break a penalty π is charged for the break ([3], p. 96). The value is squared but the sign is kept so that the line demerits are lowered when a negative penalty is given. Penalties lie in the range $-10000 < \pi \leq 10000$; $\pi = -10000$ forces a break but does not add to the line demerits. A break at glue gets $\pi = 0$; otherwise a break at a hyphen uses either `\hyphenpenalty` or `\exhyphenpenalty`, a break in math mode either `\binoppenalty` or `\relpenalty`, and a break at an explicit `\penalty` command uses the given value. The plain TeX format sets the value of the four mentioned parameters to 50, 50, 700, and 500, respectively.

The last parameter is named the *additional demerits* δ . Lines interact with their predecessors: If visually incompatible lines would be output or if two hyphens in a row occur or if the penultimate line of the paragraph ends with a hyphen then additional demerits are added. The term δ is the sum of the parameters for these three mentioned cases: `\adjdemerits`, `\doublehyphendemerits`, and only for the last line `\finalhyphendemerits`. The default values in plain TeX are 10000, 10000, and 5000, respectively.

The Pascal code in [4, §859] shows that the first summand on the right hand side of equation (1) takes a minimum of two numbers and involves the absolute value of $\lambda + \beta$. Actually the summand is

coded as

$$(\min(10000, |\lambda + \beta|))^2.$$

Of course, in the plain \TeX format $\lambda + \beta < 10000$; but the code states that the value

$$10000 - \max(\text{\texttt{pretolerance}}, \text{\texttt{tolerance}}) - 1 \quad (2)$$

builds an upper limit for a positive $\text{\texttt{linepenalty}}$.

If the line demerits are calculated for line number ι then Λ , β , π , and δ receive ι as a subscript. So the total demerits Λ_t of a paragraph with μ lines is given by

$$\begin{aligned} \Lambda_t &= \sum_{\iota=1}^{\mu} \Lambda_{\iota} = \sum_{\iota=1}^{\mu} ((\lambda + \beta_{\iota})^2 + \text{sgn}(\pi_{\iota})\pi_{\iota}^2 + \delta_{\iota}) \\ &= \mu\lambda^2 + 2\lambda B + \sum_{\iota=1}^{\mu} (\beta_{\iota}^2 + \text{sgn}(\pi_{\iota})\pi_{\iota}^2 + \delta_{\iota}) \end{aligned} \quad (3)$$

where the sum of all badness values is called B for short, i.e., $B := \sum_{\iota=1}^{\mu} \beta_{\iota}$.

The total demerits of a paragraph sum certain values that are associated with the path through the network of line breaks that \TeX has identified as the shortest path according to its cost function. This calculation can be performed for any path in this network so the notation Λ_p for *path demerits* is introduced.

If the value of $\text{\texttt{linepenalty}}$ is important it is given as an argument to Λ_t or Λ_p , for example, the left hand side of (3) can be written as $\Lambda_t(\lambda)$.

The total demerits also have an upper limit. In [4, §833] this limit is coded and it is best to have

$$\Lambda_t < 2^{30} - 1 = 1,073,741,823 \quad (4)$$

otherwise \TeX might output overfull lines although line breaks seem to be possible. \TeX does not stop working but except for the end of the paragraph it does not create useful feasible breakpoints that are required for \TeX 's line-breaking decisions [4, §835].

In order to get familiar with the notation a simple lemma is proved. (The symbol ‘□’ marks the end of a proof or of an example.)

Lemma 1. *If $\text{\texttt{linepenalty}} \geq 0$ and if for a line that has neither penalties nor additional demerits the line demerits are larger than the line demerits of a second line with a penalty ≥ 0 , additional demerits ≥ 0 , and a summand $\epsilon \geq 0$ then the badness of the first line is larger than the badness of the second.*

Proof: With (1) the lemma claims for two lines with line demerits Λ and Λ' that with $\epsilon \geq 0$

$$\Lambda > \Lambda' + \epsilon \implies \beta > \beta'$$

if there are no penalties and additional demerits in Λ , i.e., $\pi = 0$ and $\delta = 0$, and if $\pi' \geq 0$ and $\delta' \geq 0$.

$$\Lambda > \Lambda' + \epsilon$$

$$\implies (\beta + \lambda)^2 > (\beta' + \lambda)^2 + \text{sgn}(\pi')\pi'^2 + \delta' + \epsilon$$

by (1). The sum $\text{sgn}(\pi')\pi'^2 + \delta' + \epsilon$ is ≥ 0 as $\pi' \geq 0$, $\delta' \geq 0$, and $\epsilon \geq 0$. It follows that

$$\begin{aligned} &(\beta + \lambda)^2 - (\beta' + \lambda)^2 > \pi'^2 + \delta' + \epsilon \\ \implies &(\beta - \beta')(\beta + \beta' + 2\lambda) > 0. \end{aligned}$$

Badness values are ≥ 0 and $\lambda \geq 0$ so that the term $\beta + \beta' + 2\lambda$ must be > 0 . Its product with $\beta - \beta'$ is greater than 0 so that $\beta - \beta' > 0$ or $\beta > \beta'$. □

Next another well-known property of plain \TeX is stated as a lemma.

Lemma 2. *In plain \TeX the last line of a paragraph that does not end with a penalty item of value -10000 has either badness 0 or its glue shrinks.*

Proof: If the last line contains infinite glue the badness is 0 ([3], p. 97).

Otherwise all glue is finite. In plain \TeX the $\text{\texttt{parfillskip}}$ is defined as $0pt$ plus $1fil$. Without the $\text{\texttt{parfillskip}}$, which is added by \TeX after removing the last glue item in a paragraph ([3, pp. 99–100]), the last line either contains only text or its glue stretches, or shrinks, or has its natural width. In the first two cases the stretchability of $\text{\texttt{parfillskip}}$ makes the badness 0. In the last two cases it does not change anything; a line in which the glue has its natural width has badness 0. □

3 When is a single line broken by \TeX ?

Let's start with perhaps the simplest case in which the effect of $\text{\texttt{linepenalty}}$ as a factor for \TeX 's line-breaking decisions can be analyzed: The network of breakpoints allows \TeX to typeset either a single line or a pair of lines. Under what conditions does \TeX prefer two lines?

Three assumptions are made in this section:

1. The $\text{\texttt{linepenalty}}$ is nonnegative, i.e., ≥ 0 .
2. The $\text{\texttt{parfillskip}}$ is $0pt$ plus $1fil$.
3. The line width of the second line for the paragraph is wider than the width of the material that is moved from the first to the second line.

Negative values for $\text{\texttt{linepenalty}}$ are discussed in Section 5. The reason for the (quite natural) third assumption, which states that a line break produces at most one additional line, is explained in Section 4.

The line demerits of the single line are called Λ_1 . The two-line solution is marked with a prime and its line demerits are called Λ'_1 and Λ'_2 .

Note that the network must be built from the first pass of the line-breaking algorithm, as a single line is a valid solution. Of course, the single line must shrink its glue, as a line with badness 0 is never a candidate to be typeset in two lines if the user has not entered a $\text{\texttt{penalty}}$ command with a negative

value. Without such a penalty a line with badness 0 has the line demerits $\Lambda_1 = \lambda^2$ and a two-line solution must have at least this value for its second line alone: $\Lambda'_2 \geq \lambda^2$. The first line adds the positive value Λ'_1 to the path demerits as no negative penalties are involved, making a two-line solution worse than the one-line result. There is no other case for the glue of the single line as by Lemma 2 the glue cannot stretch.

But let's look at the general case. The path demerits of the single line are computed as

$$\Lambda_p = \Lambda_1 = (\lambda + \beta_1)^2 \quad (5)$$

because no penalty is added in a first pass; i.e., $\pi_1 = 0$, $\delta_1 = 0$ except if the line is very loose but that cannot happen by Lemma 2; so this summand can be dropped too.

For the two-line solution the calculation is

$$\begin{aligned} \Lambda'_p &= \Lambda'_1 + \Lambda'_2 \\ &= (\lambda + \beta'_1)^2 + \text{sgn}(\pi'_1)\pi'^2 + \delta'_1 + \lambda^2 + \delta'_2 \end{aligned} \quad (6)$$

because $\beta'_2 = \pi'_2 = 0$ as the second line must have badness 0 because of the assumptions about the line width and the `\parfillskip`. At a break with a hyphen π'_1 is either the value of `\exhyphenpenalty` if the hyphen is part of the text or `\hyphenpenalty` for user entered discretionary hyphens. In both cases the additional demerits of the second line, δ'_2 , must contain the value of `\finalhyphendemerits`, called δ_f . And δ'_1 is either 0, or if this line is very loose `\adjdemerits`, named δ_a . In this case δ'_2 contains δ_a too. A break in math or at an explicit `\penalty` does not influence the additional demerits.

`TeX` will break the text into two lines if and only if $\Lambda_t = \Lambda'_p < \Lambda_p$.

Case 1: No penalty. This means $\pi'_1 = 0$ and δ'_2 does not contain δ_f ; thus (6) simplifies to

$$\Lambda'_p = (\lambda + \beta'_1)^2 + \delta'_1 + \lambda^2 + \delta'_2. \quad (7)$$

A path that generates two lines is preferred by `TeX` if the right hand side of (7) is smaller than the right hand side of (5):

$$(\lambda + \beta_1)^2 > (\lambda + \beta'_1)^2 + \delta'_1 + \lambda^2 + \delta'_2. \quad (8)$$

To make this inequality true β'_1 must be smaller than β_1 by Lemma 1; the difference is called the “change” χ of badness for the two-line solution, i.e., $\beta_1 - \chi = \beta'_1$ with $\chi > 0$. As the badness β'_1 is greater than or equal to 0 one more inequality is known

$$\beta_1 \geq \chi. \quad (9)$$

All solutions must have a badness of the single line that lies on or above the identity function $g(\chi) = \chi$.

Now $\beta'_1 < 100$ so the first line of the pair is not very loose, thus $\delta'_1 = \delta'_2 = 0$ and inequality (8)

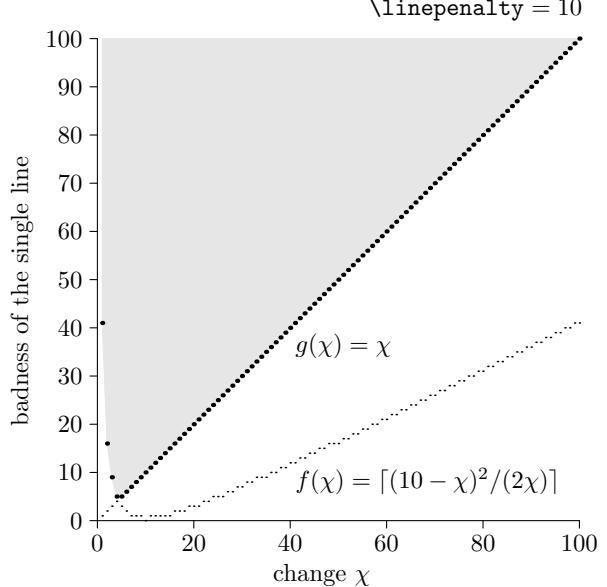


Figure 1: Graphs for functions of Theorem 1

becomes with $\beta'_1 = \beta_1 - \chi$

$$(\lambda + \beta_1)^2 > (\lambda + \beta_1 - \chi)^2 + \lambda^2 \quad (10)$$

$$\iff 2\beta_1\chi > (\lambda - \chi)^2. \quad (11)$$

As $\chi > 0$, inequality (11) can be written as

$$\beta_1 > \frac{(\lambda - \chi)^2}{2\chi}. \quad (12)$$

With (9) the left hand side of (11) is kept equal or made smaller when β_1 is replaced by χ . If this new inequality holds then (12) holds too.

$$\begin{aligned} 2\chi^2 &> (\lambda - \chi)^2 \\ \iff \sqrt{2}\chi &> \lambda - \chi \quad \vee \quad \sqrt{2}\chi > \chi - \lambda \\ \iff \chi &> (\sqrt{2} - 1)\lambda \quad \vee \quad \chi > -(\sqrt{2} + 1)\lambda. \end{aligned}$$

The right-side inequality doesn't say anything new, as $\chi > 0$. If $\lambda = 0$ both inequalities state $\chi > 0$ so that only (12) counts.

This computation proves the following theorem.

Theorem 1. *In plain `TeX` with `\linepenalty` ≥ 0 a text that fits into one line is typeset in two lines containing a line break without penalties if the difference between the badness of the single line and the badness of the first line of the pair is larger than $(\sqrt{2} - 1)\linepenalty$*

or this difference, named “change”, is larger than zero and the badness of the single line is larger than $(\linepenalty - \text{change})^2 / (2 \times \text{change})$. \square

When plain `TeX`'s settings are used the value 10 can be plugged in for `\linepenalty`. Thus the identity function is used as lower limit for the badness when the change is larger than 4 as $(\sqrt{2} - 1) \times 10 \approx$

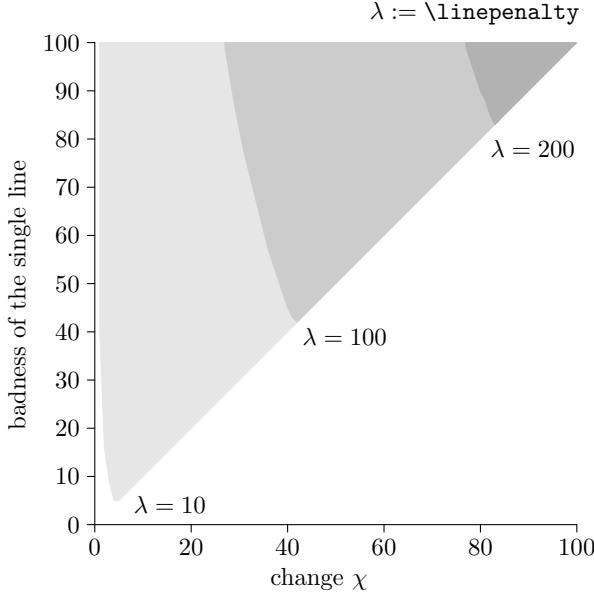


Figure 2: Solution sets for three `\linepenalty` values

4.14. For (1, 2, 3, 4) the badness of the single line must be larger than or equal to (41, 17, 9, 5), respectively; see (12). Figure 1 shows the graphs of two functions for the integer values from 1 to 100. First, the identity is shown as g . The second function f represents in essence the formula of the right hand side of (12). If the badness of the single line lies on or above the thick points for a given χ then two lines are typeset. The gray area forms the *solution set*.

Of course, the `\linepenalty` can be changed. Figure 2 shows the solution sets for three different values of `\linepenalty`: For $\lambda = 10$ all three gray areas count (compare the areas to the solution set shown in Fig. 1), for $\lambda = 100$ the light-colored area is excluded, and for $\lambda = 200$ only the dark area builds the solution set.

Case 2a: Break at hyphen. This important special case of a break with penalties is treated first. The break must be either at an explicit hyphen in the text or at an inserted discretionary break as the network is built from the first pass of TeX's line-breaking algorithm. This means that π'_1 equals either `\exhyphenpenalty` or `\hyphenpenalty` and as explained above there are additional demerits $\delta'_2 = \finalhyphendemerits = \delta_f$.

Now (10) and thus (12) get additional constant terms on their right hand sides; (12) becomes:

$$\beta_1 > \frac{(\lambda - \chi)^2 + \operatorname{sgn}(\pi'_1)\pi'^2 + \delta_f}{2\chi}.$$

As in plain TeX $\pi'_1 = 50$ and $\delta_f = 5000$, the sum $\operatorname{sgn}(\pi'_1)\pi'^2 + \delta_f$ is 7500. A graph for the above inequality similar to Fig. 1 is shown in Fig. 3. The

change must be at least 43 to get two lines. With $\beta_1 = 78$ the change must be larger than 75. The comparison of Figs. 1 and 3 shows that in essence the point from which the identity function dominates the other function is moved on this line to a higher value. (The same effect occurs in Fig. 2.)

Case 2b: Break at positive penalty. Lemma 1 is applicable. So starting in (10) with an $\epsilon \geq 0$, which is the sum of the penalty of the first line of the pair and the additional demerits of both lines added to the right hand side, the equivalent of (12) is

$$\beta_1 > \frac{(\lambda - \chi)^2 + \epsilon}{2\chi}.$$

Similarly, starting with inequality (11) and replacing β_1 by χ gives

$$\begin{aligned} 2\chi^2 &> (\lambda - \chi)^2 + \epsilon \\ \iff & (\chi + \lambda)^2 > 2\lambda^2 + \epsilon \\ \iff & \chi > \sqrt{2\lambda^2 + \epsilon} - \lambda \quad \vee \quad \chi < -\sqrt{2\lambda^2 + \epsilon} - \lambda. \end{aligned}$$

Obviously the second inequality is not relevant.

Thus a generalization of Theorem 1 is proved:

Theorem 2. *Given a text that fits into one line or can be typeset in two lines with a line break that has the value $\epsilon \geq 0$ as the sum of penalties and additional demerits. Let `\linepenalty` ≥ 0 .*

The two-line solution is used by plain TeX if the change > 0 is either at least

$$\sqrt{2\operatorname{linepenalty}^2 + \epsilon} - \operatorname{linepenalty}$$

or the badness of the single line is larger than

$$((\operatorname{linepenalty} - \text{change})^2 + \epsilon) / (2 \times \text{change}). \quad \square$$

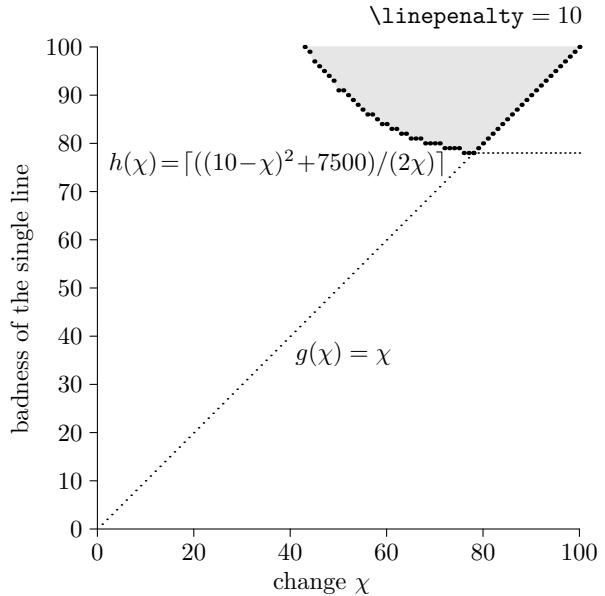


Figure 3: Graphs of functions for a break at hyphen

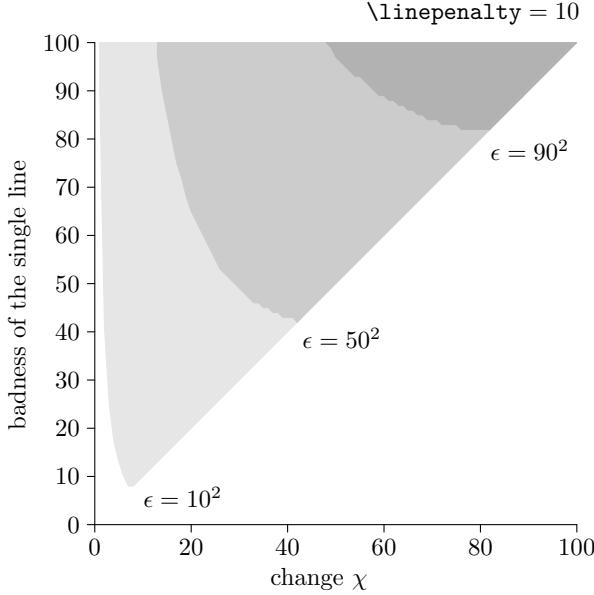


Figure 4: Solution sets for three different penalties

Figure 4 compares the solution sets for three different ϵ similar to Fig. 2.

When the `\linepenalty` is increased to 100 the solution set shrinks as shown in Fig. 2. But Theorem 2 mentions two limits, and the formula $\sqrt{2\lambda^2 + \epsilon} - \lambda$ gives for the three penalties 10, 50, and 90, i.e., the three ϵ amounts 10^2 , 50^2 , and 90^2 :

$$\begin{aligned} \epsilon &= 10^2 \quad 50^2 \quad 90^2 \\ \lambda = 10 \implies \sqrt{200+\epsilon} - 10 &\approx 7.3 \quad 41.9 \quad 81.1 \\ \lambda = 100 \implies \sqrt{20000+\epsilon} - 100 &\approx 41.7 \quad 50 \quad 67.6 \end{aligned}$$

Thus the values of the limit get larger for $\epsilon = 10^2$ and $\epsilon = 50^2$ when λ is changed to 100 but for $\epsilon = 90^2$ it is smaller! Its solution set is not a subset of the solution set when $\lambda = 10$. Figure 5 shows the solution sets for the three values of ϵ with $\lambda = 100$.

Case 2c: Negative penalties. In this case, the inequality (8) is changed to

$$(\lambda + \beta_1)^2 > (\lambda + \beta'_1)^2 + \delta'_1 + \lambda^2 - \epsilon \quad (13)$$

in which $\epsilon > 0$ stands for the sum of penalties of the first line and additional demerits of the second line as in case 2b. That means δ'_2 is contained in ϵ and thus not mentioned in (13). $\delta'_1 = 0$ except the first line of the two-line solution, named L'_1 , is very loose. Then δ'_1 is the value of `\adjdemerits` = δ_a . The bracket notation is used to identify this summand: $\delta_a[L'_1 \text{ very loose}]$. And this means the additional demerits of the second line contains δ_a too. But ϵ is not changed; instead the term δ_a is added twice.

Lemma 1 is not applicable and β'_1 can be larger than β_1 . For some “change” χ , $-100 \leq \chi \leq 100$, let $\beta_1 - \chi = \beta'_1$. In other words: Instead of (9), now two

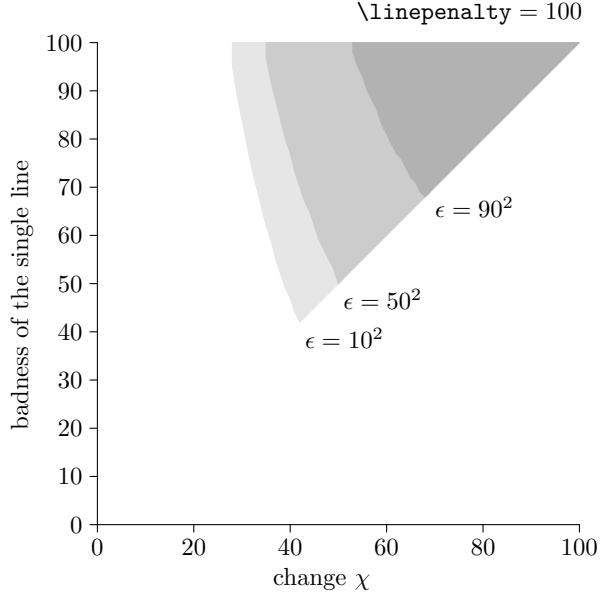


Figure 5: Like Fig. 4 for larger `\linepenalty`

limits for the badness of the single line are required:

$$\beta_1 \geq \chi, \quad \text{if } \chi > 0; \quad (14a)$$

$$\beta_1 \leq 100 + \chi, \quad \text{if } \chi < 0. \quad (14b)$$

Inequality (13) becomes

$$(\lambda + \beta_1)^2 > (\lambda + \beta_1 - \chi)^2 + 2\delta_a[L'_1 \text{ very loose}] + \lambda^2 - \epsilon$$

or after the usual transformations

$$2\beta_1\chi > (\chi - \lambda)^2 + 2\delta_a[L'_1 \text{ very loose}] - \epsilon.$$

If $\chi = 0$ this inequality states that ϵ must be larger than $\lambda^2 + 2\delta_a[L'_1 \text{ very loose}]$ in order to typeset two lines.

If $\chi > 0$ then $\beta'_1 \neq 100$, i.e., L'_1 cannot be very loose and

$$\beta_1 > \frac{(\chi - \lambda)^2 - \epsilon}{2\chi}. \quad (15)$$

As before χ is used to replace β_1 with (14a) to get

$$2\chi^2 > \chi^2 - 2\lambda\chi + \lambda^2 - \epsilon$$

$$\iff (\chi + \lambda)^2 > 2\lambda^2 - \epsilon$$

$$\iff \chi > \sqrt{2\lambda^2 - \epsilon} - \lambda \vee \chi < -\sqrt{2\lambda^2 - \epsilon} - \lambda.$$

Only the first inequality states something new: If $\epsilon \geq 2\lambda^2$ two lines are typeset. Otherwise $\epsilon < 2\lambda^2$ and either $\chi > \sqrt{2\lambda^2 - \epsilon} - \lambda$ or the badness of the first line fulfilling (15) are needed to get two lines.

If $\chi < 0$ the first line of the two-line solution might be very loose and χ must obey (14b).

First, the inequality (15) changes to

$$\beta_1 < \frac{(\chi - \lambda)^2 + 2\delta_a[L'_1 \text{ very loose}] - \epsilon}{2\chi}. \quad (16)$$

Starting from

$$2\beta_1\chi > (\chi - \lambda)^2 + 2\delta_a[L'_1 \text{ very loose}] - \epsilon$$

as above, now (14b) must be used to replace β_1 :

$$2(100 + \chi)\chi > (\chi - \lambda)^2 + 2\delta_a[L'_1 \text{ very loose}] - \epsilon.$$

With the usual transformations this leads to the relevant solution

$$\chi > \sqrt{(100 + \lambda)^2 + \lambda^2 + 2\delta_a[L'_1 \text{ very loose}]} - \epsilon - 100 - \lambda.$$

Thus a somewhat complex third theorem is proved:

Theorem 3. Given a text that fits into one line or can be typeset in two lines with a line break that has the value $-\epsilon < 0$ as the sum of penalties and additional demerits except for `\adjdemerits` if the first line is very loose. Let $\backslash\linepenalty \geq 0$.

If the change is 0 then there are two cases: If the first line of the pair is very loose ϵ must be larger than $\backslash\linepenalty^2 + 2\backslash\adjdemerits$; otherwise $\epsilon > \backslash\linepenalty^2$ is sufficient.

If the change is > 0 then $\epsilon \geq 2\backslash\linepenalty^2$ typeset two lines; otherwise if $\epsilon < 2\backslash\linepenalty^2$ then either the change must be larger than

$$\sqrt{2\backslash\linepenalty^2 - \epsilon} - \backslash\linepenalty$$

or the badness of the single line is larger than

$$\frac{(\text{change} - \backslash\linepenalty)^2 - \epsilon}{2 \times \text{change}} \quad (*)$$

to output two lines.

If the change is smaller than 0 but the first line is not very loose and $\epsilon \geq (100 + \lambda)^2 + \lambda^2$ then two lines are created. Otherwise if ϵ is smaller then either the change must be larger than

$$\sqrt{(100 + \backslash\linepenalty)^2 + \backslash\linepenalty^2 - \epsilon} - 100 - \backslash\linepenalty$$

or the badness of the single line must be smaller than (*) to output two lines.

If the first line of the pair is very loose then two lines are typeset if either the change is smaller than

$$\sqrt{(100 + \backslash\linepenalty)^2 + \backslash\linepenalty^2 + 2\backslash\adjdemerits - \epsilon} - 100 - \backslash\linepenalty$$

and (*) + `\adjdemerits/change` is larger than the badness of the single line or ϵ is larger than

$$(100 + \backslash\linepenalty)^2 + \backslash\linepenalty^2 + 2\backslash\adjdemerits. \quad \square$$

Figure 6 shows in the style of previous figures three instances of negative penalties. All gray areas represent the solution set if $\epsilon = 159^2$. The dots show the limit when the first line of the pair is very loose. If the dots and the lightest gray area are excluded the diagram shows the solution set for $\epsilon = 68^2$.

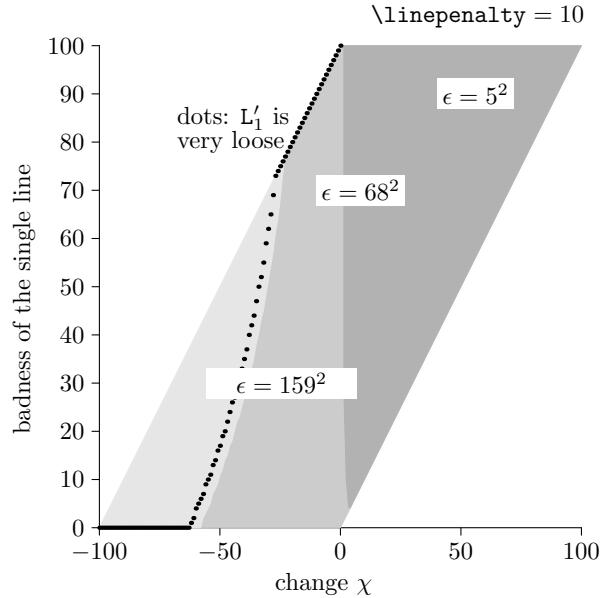


Figure 6: Solution sets for three negative values

doesn't capture all negative change but the smallest positive values build the identity function. If $\epsilon = 5^2$ only the darkest area counts and even for positive change the identity function is not achieved for small values.

Setting $\backslash\linepenalty = 100$ makes all areas smaller, the dots disappear, and for $\epsilon = 159^2$ the left edge of the solution set drops from $(-43, 57)$ to $(-59, 0)$.

4 A few consequences of the theorems

The developed theory helps to understand certain cases for plain TeX when a single line can be broken.

Theorem 1 shows how the two-line solution can be made more likely when no penalties are involved: Reduce the value of $\backslash\linepenalty$! The assignment 2 requires a change that must be larger than $2(\sqrt{2} - 1) < 1$ for all badness values, i.e., a change of 1 or more typesets the two lines.

Theorem 1 also states that with a large value of $\backslash\linepenalty$ TeX will typeset a single line, for example, with a value 242 the line break is impossible as the change must be larger than $242(\sqrt{2} - 1) > 100$.

Theorem 2 makes, among other things, statements about penalties. It proves that a penalty of 110 forces the single line as $((10 - 100)^2 + 110^2)/(2 \times 100) > 100$.

Theorem 3 implies that a `\penalty` of -180 typesets always two lines even if the first line of this pair is very loose and `\adjdemerits` are involved.

Larger `\linepenalty` might break a single line. One interesting consequence of Theorem 2 is that a

larger `\linepenalty` in combination with a positive `\penalty` breaks a line that would be kept as a single line if the default value of `\linepenalty` is used. See the discussion after Theorem 2 comparing Figs. 4 and 5.

Example 1: Description

A single line is broken when `\linepenalty` is increased.

TEX input

```
\toks0={\noindent It's a surprise, but it's true.  
See for yourself now. So~it\penalty95\ is.}  
\linepenalty=10 \the\toks0\par  
\linepenalty=100 \the\toks0\par
```

TEX output

It's a surprise, but it's true. See for yourself now. So it is. It's a surprise, but it's true. See for yourself now. So it is. □

The single line has badness 86 and without the “is.” the badness drops to 0, that is, the first line of the two-line solution produces a change of 86. Using the formula of Theorem 2, once the values 10 and 95^2 and 100 and 95^2 are used for `\linepenalty` and for ϵ , respectively, the results are that the change must be larger than 86 in the first case and larger than 70 in the second to create two lines. Thus, in the first case the break is avoided and in the second it is made.

Instead of an explicit penalty a hyphen can be the reason for a line break:

Example 1 continued: TEX input

```
\toks0={Bob, tell us, what made you  
want to look up {\sl run-in\}/?}  
\linepenalty=10 \the\toks0\par  
\linepenalty=100 \the\toks0\par
```

TEX output

Bob, tell us, what made you want to look up *run-in*?
Bob, tell us, what made you want to look up *run-in*? □

Three lines with one line break. The theory was developed with the starting point that a line break generates two lines; see assumption 3 in Section 3. But TeX is very flexible and a user can construct situations in which a line break generates two additional lines.

Example 2: Description

An unusual `\parshape` is presented that otherwise is not considered in this article.

TEX input

```
\def\weirdparshape{\setbox0=\hbox{\ninemrm {  
    \parshape 3 Opt \hsize Opt \wd0 Opt \hsize  
    Do! not! do! it! Never! No!  
    This \cs{\parshape} is bad.}}}  
\linepenalty=242 \weirdparshape\par  
\linepenalty=10 \weirdparshape\par
```

TEX output

Do! not! do! it! Never! No! This `\parshape` is bad.
Do! not! do! it! Never! No! This `\parshape` is bad. □

Of course, the theory could be extended, but it does not seem worth the effort. Such settings of `\parshape` are never applied to normal text. The author wants to generate a certain effect and controls the situation.

Longer paragraphs. The theoretical results do not apply without change to paragraphs with more than one line because the penultimate line in a long paragraph might be changed too when the last line is broken, i.e., $(L_{\mu-1}, L_\mu) \rightarrow (L'_{\mu-1}, L'_\mu, L'_{\mu+1})$ with $L_{\mu-1} \neq L'_{\mu-1}$. And even if it stays unchanged, i.e., $L_{\mu-1} = L'_{\mu-1}$, the line characteristic might influence the next line through additional demerits in different ways. Finally, the paragraph might be broken in the second pass of TeX's line-breaking algorithm, thus the badness of L'_μ might be larger than the badness of L_μ even if the change is positive.

OK, enough warning notices: There are nevertheless cases in which the theory is applicable to longer paragraphs.

Example 3: Description

Typeset a short text twice with plain TeX. First with the default settings, next with `\linepenalty = 2`.

TEX output

When you start to count where do you start? With zero or with one? Hmm, I start at 1! A CS nerd uses a 0, or? When you start to count where do you start? With zero or with one? Hmm, I start at 1! A CS nerd uses a 0, or? □

Later in Section 5 it is shown that the value 4 for `\linepenalty` is sufficient. It turns out that the value -2 works in this case too; see Section 6.

Next the technique with the penalty of 110 is used. The example also demonstrates that a large `\linepenalty` does not break the last line if the theory is applicable. Here the minimal required value for `\linepenalty` is used.

Example 4: Description

Typeset a short text thrice with plain TeX. First with plain TeX's default settings, second with a `\penalty110` inserted between the last two words and a third time without this penalty but with `\linepenalty = 199`.

TEX definitions

```
\toks0{\noindent This text can be typeset, yes,  
either in two or in three lines and the theory  
of this section applies to the}  
\toks1={\the\toks0{} last line.}  
\toks2={\the\toks0{} last\penalty110\ line.}
```

TeX input

```
\linepenalty=10 \the\toks1\par \the\toks2\par
\linepenalty=199 \the\toks1\par
```

TeX output

This text can be typeset, yes, either in two or in three lines and the theory of this section applies to the last line.

This text can be typeset, yes, either in two or in three lines and the theory of this section applies to the last line. This text can be typeset, yes, either in two or in three lines and the theory of this section applies to the last line. □

As expected the `\penalty110` prevents the line break. A tie would do the job too but if the text later grows, a break at the `\penalty` is still possible.

On the other hand if the text is typeset twice in one paragraph the theory is not applicable; with a line break in the last line the word “three” is moved from the penultimate line of the four-line paragraph to the penultimate line of the five-line paragraph.

Example 5: Description

Typeset the paragraph of example 4 two times as one paragraph: once with plain TeX’s defaults and once with `\linepenalty = 385`.

TeX input

```
\linepenalty=10 \the\toks1{} \the\toks1\par
\linepenalty=385 \the\toks1{} \the\toks1\par
```

TeX output

This text can be typeset, yes, either in two or in three lines and the theory of this section applies to the last line. This text can be typeset, yes, either in two or in three lines and the theory of this section applies to the last line.

This text can be typeset, yes, either in two or in three lines and the theory of this section applies to the last line. This text can be typeset, yes, either in two or in three lines and the theory of this section applies to the last line. □

In order to bring this paragraph to four lines `\linepenalty` must be set to 385.

5 Changing the value of `\linepenalty`

The parameter `\linepenalty` can be changed by the user. In this section an analysis is made when a different `\linepenalty` results in different line breaks and what the trade-offs are.

Example 6: Description

Typeset a paragraph several times with different values for `\linepenalty`. Start with TeX’s default settings.

TeX definitions

```
\linepenalty=10
```

TeX output

The line-breaking algorithm of TeX selects a shortest path in a network of feasible breakpoints using a cost function that calculates *demerits*. For every line

four values are used to compute the demerits for this line and then the sum of all line demerits stands for the total demerits of a paragraph. □

The line-breaking decisions of TeX are listed in the log file if `\tracingparagraphs` is set to 1. This output helps to explain the effect on the line breaks when `\linepenalty` is changed; therefore the trace data is shown. See *The TeXbook* [3], pp. 98–99, or [8], Section 3, for a description of this data.

Example 6 continued: `\tracingparagraphs'` data

1. @firstpass
2. @secondpass
3. []\ninem The line-breaking al-go-rithm of T[]X se-lects a short-
4. @\discretionary via @00 b=5 p=50 d=2725
5. @01: line 1.2- t=2725 -> @00
6. est path in a net-work of fea-si-ble break-points us-ing a
7. @ via @01 b=82 p=0 d=8464
8. @02: line 2.1 t=11189 -> @01
9. cost
10. @ via @01 b=20 p=0 d=900
11. @03: line 2.3 t=3625 -> @01
12. func-tion that cal-cu-lates \ninesl de-mer-its\ninem . For ev-ery line
13. @ via @02 b=48 p=0 d=3364
14. @04: line 3.1 t=14553 -> @02
15. four
16. @ via @03 b=45 p=0 d=13025
17. @05: line 3.1 t=16650 -> @03
18. val-
19. @\discretionary via @03 b=83 p=50 d=11149
20. @06: line 3.3- t=14774 -> @03
21. ues are used to com pute the de-mer-its for this
22. @ via @04 b=39 p=0 d=2401
23. @07: line 4.1 t=16954 -> @04
24. line
25. @ via @04 b=36 p=0 d=12116
26. @ via @05 b=63 p=0 d=5329
27. @08: line 4.1 t=21979 -> @05
28. @09: line 4.3 t=26669 -> @04
29. and
30. @ via @05 b=20 p=0 d=10900
31. @ via @06 b=5 p=0 d=225
32. @010: line 4.2 t=14999 -> @06
33. then the sum of all line de-mer-its stands for
34. @ via @07 b=84 p=0 d=8836
35. @011: line 5.1 t=25790 -> @07
36. the
37. @ via @07 b=0 p=0 d=100
38. @ via @08 b=114 p=0 d=15376
39. @ via @09 b=114 p=0 d=25376
40. @012: line 5.2 t=17054 -> @07
41. to-
42. @\discretionary via @08 b=0 p=50 d=2600
43. @\discretionary via @09 b=0 p=50 d=2600

```

44. @@13: line 5.2- t=24579 -> @@8
45. tal
46. @ via @@8 b=15 p=0 d=10625
47. @ via @@9 b=15 p=0 d=625
48. @ via @@10 b=46 p=0 d=3136
49. @@14: line 5.1 t=18135 -> @@10
50. @@15: line 5.3 t=27294 -> @@9
51. de-
52. @\discretionary via @@10 b=3 p=50 d=2669
53. @@16: line 5.2- t=17668 -> @@10
54. mer-its of a para-graph.
55. @\par via @@11 b=0 p=-10000 d=100
56. @\par via @@12 b=0 p=-10000 d=100
57. @\par via @@13 b=0 p=-10000 d=5100
58. @\par via @@14 b=0 p=-10000 d=100
59. @\par via @@15 b=0 p=-10000 d=100
60. @\par via @@16 b=0 p=-10000 d=5100
61. @@17: line 6.2- t=17154 -> @@12

```

□

Here are a few reasons why this paragraph is a good candidate to see the effect of different values for `\linepenalty`.

Reason 1: The first line breaks at a hyphen so the paragraph needs a second pass; see lines 1–2 of the listing. Thus penalties might occur, the additional demerits are not limited to `\adjdemerits`, and very loose lines are possible.

Reason 2: There are many possible paths in the network; see lines 55–60. (Of course, this is normal for most longer paragraphs.) Thus there are other ways to typeset the text.

Reason 3: Some lines have a rather high badness, but it is possible by adding a word from the neighboring line to lower the badness dramatically; for example, see lines 6–10. The shortest path contains some lines that have one of those large badness values; see lines 4, 7, 13, 22, 37.

Reason 4: Some of the possible line breaks for a penultimate line makes this line end with a hyphen (lines 51–53 and 60 as well as lines 41–44 and 57). So `\finalhyphendemerits` are available as additional demerits.

Reason 5: On the other hand, some possible lines avoiding the hyphen at the end of the penultimate line are very loose; see lines 38–40. Thus very loose lines are indeed available, and not just a possibility as stated in reason 1.

Reasons 1–4 are useful to see an effect for higher positive values of `\linepenalty`, the last one to see an effect if the value is negative.

Table 1 summarizes the paths identified in lines 55–60 of the trace data. The table shows in the first two columns the information of the @@-lines: the sequence number and the fitness class abbreviated to the first letter of very loose, loose, decent, or tight. Then six columns for the possible paths are

Table 1: Badness, penalties, and additional demerits of the line breaks for the six paths of the trace listing

		\par via @@ (* is typeset)					
@@	Class	11 ₀	*12 ₀	13 ₀	14 ₀	15 ₀	16 ₀
1	d	550	550	550	550	550	550
2	l	82	82			82	
3	t			20	20		20
4	l	48	48			48	
5	l			45 ^a			
6	t				83 ₅₀		83 ₅₀
7	l	39	39				
8	l			63			
9	t					36 ^a	
10	d				5		5
11	l	84					
12	d		0				
13	d			0 ₅₀			
14	l				46		
15	t					15	
16	d						3 ₅₀
17	d	0	0	0 ^f	0	0	0 ^f
$\mu =$		6	6	6	6	6	6
$B =$		258	174	133	159	186	116
$\Lambda_p(10) =$		25890	17154	29679	18235	27394	22768

presented (lines 55–60); the heading gives the sequence number after the “via @@”; the subscript 0 is explained later. The table entries are the badness values. A subscript signals that a penalty occurs at the break, a superscript of ‘f’ or ‘a’ that `\finalhyphendemerits` or `\adjdemerits`, respectively, are applied. Line 61 of the listing reports that the line breaks follow the path of the column labeled 12_0 . The column head contains an asterisk to indicate this selection by `TEX`.

The last three rows state the number of lines, μ , the sum of the badness values of the path, B , and the path demerits Λ_p . These values are not found directly in the trace data. They have been computed from the information in the columns.

The theory. An increase of λ by $\kappa > 0$ changes the first summand of the formula (1) for the line demerits

$$(\lambda + \kappa + \beta)^2 = (\lambda + \beta)^2 + 2\beta\kappa + 2\lambda\kappa + \kappa^2.$$

The two summands $2\lambda\kappa$ and κ^2 form a “constant” that is added to every line and therefore they do not change the line-breaking decisions by `TEX`—as long as the limits (2) and (4) are obeyed. The third summand $2\beta\kappa$ increases the influence of the badness β . That means, the penalties and the additional demerits in (1) are less important: If κ is large enough `TEX` selects a path for which more penalties or additional demerits are charged if only the badness values can be made smaller.

In fact, the increment that is necessary to go from one path to another can be calculated with (1) and (3). The path demerits become

$$\begin{aligned}\Lambda_p(\lambda + \kappa) &= \sum_{\iota=1}^{\mu} ((\lambda + \kappa + \beta_\iota)^2 + \text{sgn}(\pi_\iota)\pi_\iota^2 + \delta_\iota) \\ &= \mu(\lambda + \kappa)^2 + 2\kappa B + \Lambda_p(\lambda) - \mu\lambda^2.\end{aligned}$$

The task is to determine $\kappa > 0$ to change the total line demerits to a path with a lower sum of badness—this path gets all subscripted variables and their sums primed. Therefore starting with $B' < B$ and $\Lambda_t(\lambda) = \Lambda_p(\lambda) < \Lambda'_p(\lambda)$ find $\kappa > 0$ such that

$$\Lambda_p(\lambda + \kappa) > \Lambda'_p(\lambda + \kappa) = \Lambda_t(\lambda + \kappa).$$

In the longer form the inequality is

$$\begin{aligned}\mu(\lambda + \kappa)^2 + 2\kappa B + \Lambda_p(\lambda) - \mu\lambda^2 &> \\ \mu'(\lambda + \kappa)^2 + 2\kappa B' + \Lambda'_p(\lambda) - \mu'\lambda^2.\end{aligned}$$

A few simple transformations when $\mu = \mu'$ give

$$2\kappa(B - B') > \Lambda'_p(\lambda) - \Lambda_p(\lambda)$$

or as $B > B'$

$$\kappa > \frac{\Lambda'_p(\lambda) - \Lambda_p(\lambda)}{2(B - B')}.$$
 (17)

Its application. Table 1 shows that there are three paths with a lower sum of badness value than column 12_0 : 14_0 with sum 159, 13_0 with sum 133, and 16_0 with sum 116. Inequality (17) states the following conditions for κ :

Path (Column)	13_0	14_0	16_0
$\kappa >$	152	36	48

(Note, only integer parts of the numbers are shown.)

So $\kappa = 37$ (or $\lambda = 47$) typesets the path of column 14_0 . The path of column 13_0 cannot be reached: 16_0 has a lower sum of badness and needs a lower κ .

Example 6 continued: TeX definitions

\linepenalty=47

TeX output

The line-breaking algorithm of TeX selects a shortest path in a network of feasible breakpoints using a cost function that calculates *demerits*. For every line four values are used to compute the demerits for this line and then the sum of all line demerits stands for the total demerits of a paragraph. □

Of course, the typeset result has one more hyphenated line. Low badness values have been traded in for more penalties. A value $\kappa > 48$ selects the path of column 16_0 but it might not be the value 49. The formula does not know that there is a column in between; so 49 still creates the path of column 14_0 . Using (17) the calculation of κ to go from column 14_0 to column 16_0 gives $\kappa = 53$ or $\lambda = 63$.

Example 6 continued: TeX definitions

\linepenalty=63

TeX output

The line-breaking algorithm of TeX selects a shortest path in a network of feasible breakpoints using a cost function that calculates *demerits*. For every line four values are used to compute the demerits for this line and then the sum of all line demerits stands for the total demerits of a paragraph. □

All paths with a smaller sum of badness have been used. But these are not all possible paths as TeX ignores a path that cannot become the shortest. Except \linepenalty ≠ 10 might change TeX's viewpoint but unfortunately the path is not shown explicitly in the available trace. In total there are six more paths hidden in the data; Table 1' lists them. The paths are still named by the *par* information and now the subscript identifies the variant. The notations (10) and (t) mean that the corresponding @-line does not have its own @@-line in the trace and @@10 follows next. With this data all values of κ for the transitions to a path with a smaller sum of badness can be computed if the paths are sorted by their sum of badness values B:

	16_1	16_0	13_0	14_1	15_1	14_0	13_1	12_0
16_1	0							
16_0	272	0						
13_0	70	-204	0					
14_1	52	-201	-185	0				
15_1	23	-156	-101	-80	0			
14_0	129	52	220	272	657	0		
13_1	6	-106	-62	-52	-37	-673	0	
12_0	112	48	152	179	299	36	2869	0

As noted above 36 is the smallest number in the last row, selecting path 14_0 . In the row for 14_0 52 is the smallest number selecting 16_0 and in its row 272 is selecting 16_1 . So only one more path can be shown for $\kappa > 272$, i.e., κ must be 273 and $\lambda = \kappa + 10 = 283$.

Example 6 continued: TeX definitions

\linepenalty=283

TeX output

The line-breaking algorithm of TeX selects a shortest path in a network of feasible breakpoints using a cost function that calculates *demerits*. For every line four values are used to compute the demerits for this line and then the sum of all line demerits stands for the total demerits of a paragraph. □

Negative values. A negative amount for the integer \linepenalty does not act directly as a bonus if a line is created, as the sum with the badness is squared in equation (1). But a negative value reverts the meaning of badness! For example, a value

Table 1': Badness, penalties, and additional demerits of the line breaks for not-shown paths of the trace listing

		variant of \par via @@					
@@	Class	12 ₂	12 ₁	13 ₁	14 ₁	15 ₁	16 ₁
1	d	550	550	550	550	550	550
2	l	82		82			
3	t		20		20	20	20
4	l	48		48			
5	l		45 ^a		45 ^a	45 ^a	45 ^a
8	l		63			63	
9	t	36 ^a		36 ^a			
(10)	(t)				20 ^a		20 ^a
(12)	(v)	114 ^a	114				
13	d		0 ₅₀				
14	l			46			
15	t				15 ^a		
16	d					3 ₅₀	
17	d	0 ^a	0	0 ^f	0	0	0 ^f
$\mu =$		6	6	6	6	6	6
B =		285	247	171	136	148	93
$\Lambda_p(10) =$		62145	37455	34369	30786	32704	35319

of -110 for `\linepenalty` assigns lines with badness 0 the same demerits as lines with badness 100 get with plain TeX's default settings. And a line with badness 100 gets the value that previously a line with badness 0 received. TeX creates lines that have large badness values if possible! Higher negative values retain this effect, so they act differently from large positive values.

Tables 1 and 1' show that there are paths that have a larger sum of badness than the path of column 12₀. Inequality (17) is changed as in this case $B < B'$. Thus division by $2(B - B') < 0$ inverts the relation:

$$\kappa < \frac{\Lambda'_p(\lambda) - \Lambda_p(\lambda)}{2(B - B')} \quad (18)$$

As in the case of smaller sum of badness values a diagonal matrix with values that are larger than B of 12₀ can be built.

	12 ₀	15 ₀	12 ₁	11 ₀	12 ₂
12 ₀	0	-426	-139	-52	-202
15 ₀		0	-82	11	-175
12 ₁			0	526	-324
11 ₀				0	-671
12 ₂					0

So this time $\kappa = -53$, i.e., $\lambda = -43$, selects column 11₀.

Example 6 continued: TeX definitions

`\linepenalty=-43`

TeX output

The line-breaking algorithm of TeX selects a shortest path in a network of feasible breakpoints using a cost function that calculates *demerits*. For every line

four values are used to compute the demerits for this line and then the sum of all line demerits stands for the total demerits of a paragraph. \square

The matrix states that from row 11₀ the value $\kappa = -672$ moves on to path 12₂.

Example 6 continued: TeX definitions

`\linepenalty=-662`

TeX output

The line-breaking algorithm of TeX selects a shortest path in a network of feasible breakpoints using a cost function that calculates *demerits*. For every line four values are used to compute the demerits for this line and then the sum of all line demerits stands for the total demerits of a paragraph. \square

The limit. The `\linepenalty` has a limit as stated in (2). The setting of 10000 is larger than this limit and the badness values are completely ignored—they do not even have a “little influence” [1, p. 171]. (The effects that are shown in [1] can neither be reproduced with the font `cmt10` nor are they explained by the developed theory.) As badness plays no rôle anymore, TeX tries to avoid hyphens and visually incompatible lines as they add to the line demerits; see (1). In the current example the same line breaks as with `\linepenalty = 10` are used.

But other paragraphs, with a `\linepenalty` value above the limit (2), switch to a path that otherwise can be reached only by a negative penalty.

Table 2: Badness, penalties, and additional demerits of the line breaks for the two paths of example 7

		\par via @@ (* is typeset)	
@@	Class	*6	7
1	v		
2	d	0 ₅₀	
3	t		68
4	d	0	
5	d		5
6	l	32	
7	d		0
8	d	0	0
	$\mu =$	4	4
	B =	32	73
$\Lambda_p(10) =$		4564	6509

Example 7: Description

Typeset a paragraph three times in a forced second pass, i.e., `\pretolerance = -1`: first with `\linepenalty = 10`, second with `\linepenalty = -14`, and third with `\linepenalty = 10000`.

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 TeX? Or shall I ask Siri?

Hi! TeX! Tell me: How is the following long word broken ‘pneumonoultramicroscopicsilicovolcanoconiosis’? I am sure that you are an expert in hyphenation, right TeX? Or shall I ask Siri?

Hi! TeX! Tell me: How is the following long word broken ‘pneumonoultramicroscopicsilicovolcanoconiosis’? I am sure that you are an expert in hyphenation, right TeX? Or shall I ask Siri? \square

Table 2 shows: $\kappa < -23.7 \approx (6509 - 4564)/-82$ by (18), that is $\lambda = -14$, selects the path in column 7. As the path in column 6 contains a hyphen a \linepenalty of 10000 selects the path in column 7, which has no penalties or additional demerits.

Paths with fewer lines. In example 5, the parameter \linepenalty must be set to a large value in order to reduce the number of lines that are typeset from five to four. This is not covered by (17) as now $\mu = \mu' + 1$.

To determine the κ that selects a path with fewer lines the initial inequality must distinguish between μ and $\mu' = \mu - 1$:

$$\begin{aligned} \mu(\lambda + \kappa)^2 + 2\kappa B + \Lambda_p(\lambda) - \mu\lambda^2 > \\ (\mu - 1)(\lambda + \kappa)^2 + 2\kappa B' + \Lambda'_p(\lambda) - (\mu - 1)\lambda^2. \end{aligned}$$

The difference between μ and μ' adds the summand $(\lambda + \kappa)^2 - \lambda^2$ to the left hand side and this allows that the sum of badness can be larger for the shorter paragraph.

A simple rearrangement of the terms gives

$$\kappa^2 + 2\kappa(\lambda + B - B') > \Lambda'_p(\lambda) - \Lambda_p(\lambda). \quad (19)$$

Addition of $(\lambda + B - B')^2$ to both sides gives a quadratic term on the left

$$(\kappa + \lambda + B - B')^2 > (\lambda + B - B')^2 + \Lambda'_p(\lambda) - \Lambda_p(\lambda)$$

and as the right hand side is positive, the square root can be taken. Therefore the following relevant inequality is found

$$\begin{aligned} \kappa > B' - B - \lambda \\ + \sqrt{(\lambda + B - B')^2 + \Lambda'_p(\lambda) - \Lambda_p(\lambda)}. \end{aligned} \quad (20)$$

Table 3 shows the data of the corresponding trace listing for the text of example 5. The notation (10) is explained above; see also “No information in the trace data” in [8], p. 370ff. The path of column 8 is typeset and it has the lowest value for the sum of badness B. Using this data inequality (20) gives $\kappa > 374.8\dots$ for column 6. The required \linepenalty must be set to 385—as mentioned after example 5.

Paths with more lines. In example 3 the parameter \linepenalty was set to 2 to enlarge the number of typeset lines. So this case should be analyzed too.

Table 3: Badness, penalties, and additional demerits of the line breaks for the four paths of example 5

		\par via @@ (* is typeset)			
@@	Class	6	7	*8	9
1	d	2	2	2	2
2	l		19	19	
3	t	96			96
4	l		87		
5	d			5	
6	d	2			2
7	l		13		
8	d			1	
9	l				19 ₁₁₀
(10)	(t)	96			
10	d		0	0	0
	$\mu =$	4	5	5	5
	$B =$	196	121	27	119
	$\Lambda_p(10) =$	22760	11023	1431	24565

This time $\kappa > 0$ is subtracted from λ :

$$\begin{aligned} \Lambda_p(\lambda - \kappa) &= \sum_{\iota=1}^{\mu} ((\lambda - \kappa + \beta_{\iota})^2 + \text{sgn}(\pi_{\iota})\pi_{\iota}^2 + \delta_{\iota}) \\ &= \mu(\lambda - \kappa)^2 - 2\kappa B + \Lambda_p(\lambda) - \mu\lambda^2. \end{aligned}$$

Thus the inequality for the path demerits becomes

$$\begin{aligned} \mu(\lambda - \kappa)^2 - 2\kappa B + \Lambda_p(\lambda) - \mu\lambda^2 < \\ \mu'(\lambda - \kappa)^2 - 2\kappa B' + \Lambda'_p(\lambda) - \mu'\lambda^2 \end{aligned}$$

and with $\mu' = \mu + 1$ this is

$$\Lambda_p(\lambda) - \Lambda'_p(\lambda) < \kappa^2 - 2\kappa(B - B' - \lambda).$$

This time add the term $(\lambda + B' - B)^2$ to both sides and take square roots; then the useful result is

$$\begin{aligned} \kappa > \lambda + B' - B \\ - \sqrt{(\lambda + B' - B)^2 + \Lambda_p(\lambda) - \Lambda'_p(\lambda)}. \end{aligned} \quad (21)$$

The data of Table 4 gives for the transition from the path of column 2 to 6 with (21): $\kappa > 5.3$. Therefore \linepenalty = 10 – 6 = 4 typesets three lines as mentioned above.

Table 4: Badness, penalties, and additional demerits of the line breaks for the five paths of example 3

		\par via @@ (* is typeset)				
@@	Class	*2	3	4	5	6
1	l		86		86	
2	d	4		4		4
3	d			4		
4	l				57	
5	d					0
6	d					6
7	d	7				
(7)	(d)		0	0	0	0
	$\mu =$	2	3	3	3	3
	$B =$	11	100	61	86	10
	$\Lambda_p(10) =$	485	9512	4785	9416	552

Summary. When the `\linepenalty` value is increased, TeX's line-breaking algorithm focuses more on the badness values. If a path exists in the network of line breaks that has the same number of lines but a lower sum of badness compared to the path selected with the default settings, that path might be chosen with the larger `\linepenalty`. This means that more breaks in mathematics and/or at positive `\penalty` commands and/or more hyphens and/or more stacks of hyphens and/or more visually incompatible lines are typeset and at least one of these items is increased.

If a path exists that uses fewer lines for the paragraph, this path can be selected with a large `\linepenalty` even if its sum of badness is higher than that of the paragraph with the default settings. Similarly a path can be selected that has more lines if `\linepenalty` stays positive but is made smaller than 10.

Negative values for `\linepenalty` typically create rather ugly paragraphs as TeX then prefers large badness values for the lines. This effect is not normally desirable for justified text.

6 `\linepenalty` versus `\looseness`

The *TeXbook* has an exercise in which the value 100 for the parameter `\linepenalty` is suggested as a replacement for a negative `\looseness` in an application to a single paragraph ([3], exercise 14.25). The reason refers to efficiency to “achieve almost the same result” if the user is not willing to pay the cost that a nonzero `\looseness` generates. (`\looseness` is explained in [3], pp. 103–104 or see Section 5 of [8] for an analysis of this parameter.)

Figure 2 shows that there are a lot of cases in which two lines are typeset instead of only one if `\linepenalty = 100`. And example 1 proves that the increase of `\linepenalty` can make a paragraph longer. Therefore this parameter might not only fail to reduce the number of lines it might be counterproductive. Although passes can have different numbers of lines for the shortest path, with a small enough negative `\looseness` a paragraph can never get more lines than it has in the earliest pass that typesets it if $\text{\pretolerance} \leq \text{\tolerance}$, as the paths of the first pass are part of the network of line breaks of the second pass.

Inequality (19) can be used to determine in which cases a `\linepenalty` of 100, i.e., $\kappa = 90$, can be successful in general. Here the plain TeX values are used:

$$\begin{aligned} 90^2 + 2 \cdot 90(10 + B - B') &> \Lambda'_p(10) - \Lambda_t(10) \\ \iff 9900 - 180(B' - B) &> \Lambda'_p(10) - \Lambda_t(10). \end{aligned}$$

Therefore the difference between the sum of badness values must be less than $55 = 9900/180$, but of course it must often be much smaller as the difference of the path demerits on the right hand side is positive and usually not very small.

Although Theorem 1 proves that the value 242 for `\linepenalty` acts like `\looseness = -1` for a single line the scenario represents only a special case. For example, as noted in Section 3, the single line is always typeset by TeX's line-breaking algorithm in the first pass. In general these two parameters behave quite differently.

Second pass. This is the fundamental difference between these two parameters: `\looseness` will try hyphenation, i.e., the second pass, if it is not successful in the first.

Thus, hyphens might be introduced at the end of the lines if `\looseness` is used although no reduction of the number of lines is achieved.

Example 8: Description

Typeset the text of example 7 twice: first with plain TeX's defaults and second with `\looseness = -1`.

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 TeX? Or shall I ask Siri?

Hi! TeX! Tell me: How is the following long word broken ‘pneumonoultramicroscopicsilicovolcanoconiosis’? I am sure that you are an expert in hyphenation, right TeX? Or shall I ask Siri? □

The change of `\linepenalty` never forces TeX's line-breaking algorithm to execute another pass. It uses the pass that is necessary to break the lines when `\linepenalty = 10`.

Example 9: Description

Typeset a paragraph twice: first with `\linepenalty = 9799` and second with the default `\linepenalty = 10` and `\looseness = -1`.

TeX output

A short text that cannot be typeset in two lines although looseness does it in the 2nd pass. A surprise! Or?

A short text that cannot be typeset in two lines although looseness does it in the 2nd pass. A surprise! Or? □

TeX typesets the text of the first paragraph in the first pass. The line-breaking algorithm cannot eliminate the third line in this pass. On the other hand this means that the first pass is a failure in TeX's view if `\looseness = -1`. But the second pass is a success: Although it also prefers three lines, there is a way to output only two. The demerits for the three line solution are 6926, those for the pair of lines 15773.

Different cost functions. This leads to the next difference: `\looseness` can choose a line-breaking solution that does not represent the shortest path in the network. This never happens for any setting of `\linepenalty`; it must pick the shortest path.

As `\looseness` has a different cost function to be optimized, penalties larger than -10000 and smaller than 10000 mark places that are as good as others for a line break.

Example 10: Description

A text with two penalties is typeset twice: first with `\linepenalty = 9799` and second with `\looseness = -1` and `\linepenalty = 10`.

TeX input

```
OK! Even 4-digit penalties, positive or negative,
are \penalty9999\ not important for looseness but
linepenalty obeys \penalty-9999\ them.
```

TeX output

OK! Even 4-digit penalties, positive or negative, are not important for looseness but `linepenalty` obeys them.

OK! Even 4-digit penalties, positive or negative, are not important for looseness but `linepenalty` obeys them. \square

Both paragraphs are typeset in the first pass. The `\linepenalty` must pick the shortest path in the network with the cost function of demerits and thus TeX typesets three lines. This cost function is not relevant for `\looseness` if the number of lines of the paragraph can be changed. Only if this is not possible does TeX select the shortest path in the current pass as usual. This means a parameter that does not inhibit some behavior does not count if `\looseness` can be successful.

Success rate. No single value of `\linepenalty` works for all paragraphs but `\looseness` is always successful if the paragraph can be typeset with fewer lines.

Example 4 shows a text that can be typeset in two or three lines; a pair is output if `\linepenalty` is set to 199. Example 5 typesets the text twice and it needs `\linepenalty = 385` to keep four lines. Each additional repetition of the original text requires a larger `\linepenalty`:

number of copies	1	2	3	4	5	6	7
<code>\linepenalty =</code>	199	385	557	738	918	1098	1278

Of course, this is a constructed example, but nevertheless with 25 iterations the `\linepenalty` must be 4519. The next step, i.e., a paragraph with only 52 lines in the shortest form, cannot be typeset with a `\linepenalty` of 4541 anymore and it would need 4599 if the progression continues as before. It violates TeX's limit for the total demerits, see (4),

and the text is not typeset correctly. With the default `\linepenalty` together with `\looseness = -1` no problem occurs.

Note: The large value 9799 for `\linepenalty`, which was used in the last two examples, can be applied for paragraphs with at most ten lines. In order to demonstrate certain effects in examples its usage is needed, but such a large value would not be used for normal copy.

Quality of output. If the number of lines of a paragraph cannot be lowered then `\looseness` still tries to find a line-breaking solution that avoids visually incompatible lines and stacks of hyphens, i.e., it obeys the additional demerits `\adjdemerits` and `\doublehyphendemerits` if possible.

But Section 5 shows that `\linepenalty` trades the smaller sum of badness for penalties and additional demerits. This means that a text that contains no math, no explicit `\penalty` command, and no explicit hyphens must get visually incompatible lines in a first pass if TeX changes the line breaks.

In the second pass `\linepenalty` considers hyphens as `\looseness` does. The latter treats the parameters `\double...` and `\finalhyphendemerits` as usual while the former parameter trades them like `\adjdemerits` for a smaller sum of badness values. That is, TeX not only might typeset more hyphens, but also there might be more visually incompatible lines, more stacks of hyphens, and more hyphenated penultimate lines.

Efficiency. The advantage of increasing the parameter `\linepenalty` instead of using `\looseness` is that `\linepenalty`'s value is always added in the code of the line-breaking algorithm—even if it is zero. A nonzero `\looseness` invokes otherwise unused code and thus slows the algorithm down [4, §§ 873, 875]. As expected, this loss in efficiency is hardly noticed in normal copy with modern equipment.

My outdated computer from 2011 (a 1.8 GHz Dual-Core i7) with my own TeX installation shows the following factors by which the runtime increases. The reference value 1 is used for the time needed by plain TeX to typeset the two lines of example 4 with `\looseness = 0`.

copies	<code>\looseness = 0</code>	<code>\looseness = -1</code>
1	1	1
100	3	11
225	6	100

But even the abnormally long paragraph of the last case with 450 lines needs only one second to get typeset if `\looseness = -1`.

\looseness can be positive. A value larger than zero for the parameter `\looseness` tries to make the paragraph longer.

Negative values for `\linepenalty` usually typeset low quality paragraphs. This was proved in Section 5. Thus it is not a good idea to use them except when they would lengthen paragraphs; Section 5 shows that values < 10 can be successful.

Example 3 uses the value 2 for `\linepenalty` to typeset three instead of two lines. It happens that the `\linepenalty` can be -2 (but not -3).

Example 11: Description

Typeset example 3 with `\linepenalty = -2`.

TEX output

When you start to count where do you start? With zero or with one? Hmm, I start at 1! A CS nerd uses a 0, or?

The next example uses a small positive and a high negative value for `\linepenalty`. But even in a forced second pass (used by `\looseness = 1`) these values do not make TeX typeset an additional line.

Example 12: Description

Typeset a paragraph four times: first with plain TeX's settings, second with `\looseness = 1`, third in a forced second pass with `\linepenalty = 1`, and finally, still in the second pass, with a `\linepenalty` of -9999 .

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.

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.

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.

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.

Summary. TeX must work harder if `\looseness` is negative but a large value of `\linepenalty` is not a replacement. A large value for the parameter `\linepenalty` might even increase the number of lines output for a paragraph.

- If a paragraph can be typeset with fewer lines than TeX's default settings produce then

`\looseness = n` for some $n \leq -1$ is successful;
`\linepenalty > 10` might be successful but only if

the pass has not to be changed; otherwise the paragraph is treated like an unsuccessful case.

- If fewer lines for the paragraph are impossible `\looseness` tries the final pass and thus might insert hyphens but outputs the shortest path;
`\linepenalty` outputs the (new) shortest path that might now have lines with lower badness, but then more breaks in mathematics or at positive `\penalty` commands or more hyphens or more stacks of hyphens or more visually incompatible lines are used.

Both parameters might have a “negative impact” on paragraphs that cannot be shortened but the outcome with a large `\linepenalty` seems to be worse. Its only advantage is that it does not change the pass and that it obeys other TeX parameters.

References

- [1] David Bausum, *TeX Reference Manual*, Norwell, Massachusetts: Kluwer Academic Publishers, 2002
tug.org/utilities/plain/cseq.html#linepenalty-rp
- [2] 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 [7], 67–155
- [3] Donald E. Knuth, *The TeXbook*, Volume A of *Computers & Typesetting*, Boston, Massachusetts: Addison-Wesley, 1984
- [4] Donald E. Knuth, *TeX: The Program*, Volume B of *Computers & Typesetting*, Boston, Massachusetts: Addison-Wesley, 1986
- [5] Donald E. Knuth, “The Errors of TeX”, *Software—Practice and Experience* **19** (1989), 607–685; reprinted as Chapter 10 in [6], 243–291
- [6] Donald E. Knuth, *Literate Programming*, Stanford, California: Center for the Study of Language and Information, CSLI Lecture Notes No. 27, 1992
- [7] Donald E. Knuth, *Digital Typography*, Stanford, California: Center for the Study of Language and Information, CSLI Lecture Notes No. 78, 1999
- [8] Udo Wermuth, “Tracing paragraphs”, *TUGboat* **37**:3 (2016), 358–373
tug.org/TUGboat/tb37-3/tb117wermuth.pdf
- [9] Udo Wermuth, “The optimal value for `\emergencystretch`”, *TUGboat* **38**:1 (2017), 64–86
tug.org/TUGboat/tb38-1/tb118wermuth.pdf

◊ Udo Wermuth
Dietzenbach, Germany
u dot wermuth (at) icloud dot com

Errata for previous articles. Here are two corrections for errors that are not merely typographical.

In [8], p. 365, left column, the `\hsize` of the books *The Art of Computer Programming* by D. E. Knuth should be 348 pt.

In [9], p. 75, left column no. 10, the value of `t` is wrong; the stated value represents `2t`.



The Treasure Chest

This is a selection of the new packages posted to CTAN (ctan.org) from August–October 2017, 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

fonts

* `coelacanth` in fonts

L^AT_EX support for Coelacanth, inspired by the classic Centaur design.

`dejavu-otf` in fonts

Support for TrueType DejaVu font family.

`spark-otf` in fonts

Support OpenType sparkline fonts.

graphics

`dynkin-diagrams` in graphics/pgf/contrib

Draw mathematical Dynkin diagrams with TikZ.

`endofproofwd` in graphics

An additional end-of-proof symbol.

`istgame` in graphics/pgf/contrib

Draw game trees with TikZ.

`tikzducks` in graphics/pgf/contrib

Rubber ducks, with adornments, in TikZ.

info

* `amscls-doc` in info

Comprehensive user documentation for AMS document classes.

`latex-refsheet` in info

Cheat sheet for KOMA-Script thesis.

macros/generic

`fixjfm` in macros/generic

Work around bugs in p^TE_X's JFM format.

`ifxptex` in macros/generic

Detect p^TE_X and its derivatives.

`simplekv` in macros/generic

Key/value system for (I)TeX.

`upzhkinsoku` in macros/generic

Supplementary Chinese kinsoku (line breaking rules, etc.) for Unicode.

macros/latex/contrib

`abnt` in macros/latex/contrib

Brazil's ABNT rules for academic texts.

`algobox` in macros/latex/contrib

Typeset Algobox programs.

`beilstein` in macros/latex/contrib

L^AT_EX and BIBL^ETEX support for the Beilstein Journal of Nanotechnology.

`cesenaexam` in macros/latex/contrib

Typeset examinations.

`cheatsheet` in macros/latex/contrib

Class to typeset cheat sheets.

`dijkstra` in macros/latex/contrib

Dijkstra's algorithm for weighted graphs in L^AT_EX.

`ducksay` in macros/latex/contrib

Draw ASCII art of animals saying a message.

`eq-save` in macros/latex/contrib

Save and return to `exerquiz` quizzes.

`eqnnumwarn` in macros/latex/contrib

Warn about displaced equation number in amsmath environments.

`fetchcls` in macros/latex/contrib

Fetch current class name.

`forms16be` in macros/latex/contrib

Initialize form strings using BigEndian UTF-16BE.

`hithesis` in macros/latex/contrib

Harbin Institute of Technology thesis template.

`komacv-rg` in macros/latex/contrib

Help creating CVs based on the `komacv` class.

`ku-template` in macros/latex/contrib

Include Copenhagen University logos.

`limecv` in macros/latex/contrib

CV class for X^EL^AT_EX and LuaL^AT_EX.

`mensa-tex` in macros/latex/contrib

Typeset simple school cafeteria menus.

`multilang` in macros/latex/contrib

Maintaining multiple translations of a document.

`musicography` in macros/latex/contrib

Accessing symbols for music writing in pdfL^AT_EX.

`notestex` in macros/latex/contrib

Note-taking package for students.

`octave` in macros/latex/contrib

Typeset musical pitches with octave designations.

`pm-isomath` in macros/latex/contrib

Poor man's ISO math for pdfL^AT_EX.

`termcal-de` in macros/latex/contrib

German localization for the `termcal` package.

`theatre` in macros/latex/contrib

Sophisticated support for typesetting stage plays.

`witharrows` in macros/latex/contrib

Typeset arrows on right side of math alignments.

***xltabular in macros/latex/contrib**

Combine `longtable` and `tabularx`: header/footer definitions, X specifier, possible page breaks.

zhlipsum in macros/latex/contrib

Dummy Chinese text, using UTF-8.

macros/latex/contrib/beamer-contrib**hackthefootline in m/l/c/beamer-contrib**

Arbitrary footline selection for standard themes.

macros/luatex**fontloader-luaotfload in macros/luatex/generic**

Offer a few alternative font loaders.

macros/xetex/latex**na-box in macros/xetex/latex**

Arabic-aware version of `pas-cours`.

na-position in macros/xetex/latex

Tables of relative positions of curves and asymptotes or tangents in Arabic documents.

xexchangebar in macros/xetex/latex

Version of `changebar` for X_ET_EX.

support**bib2gls in support**

Java application to extract glossary information from `.bib` files; see article in this issue, pp. 373–399.

web**web2w in web**

Convert Knuth's `tex.web` to CWEB; see article in this issue, pp. 353–358.

Another seasonal puzzle: XII take II

David Carlisle

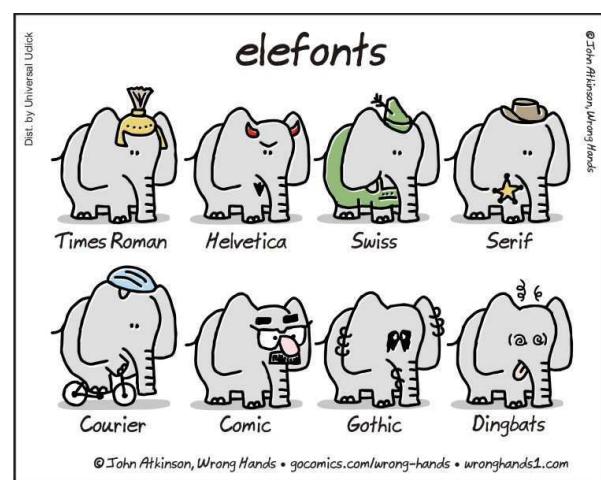
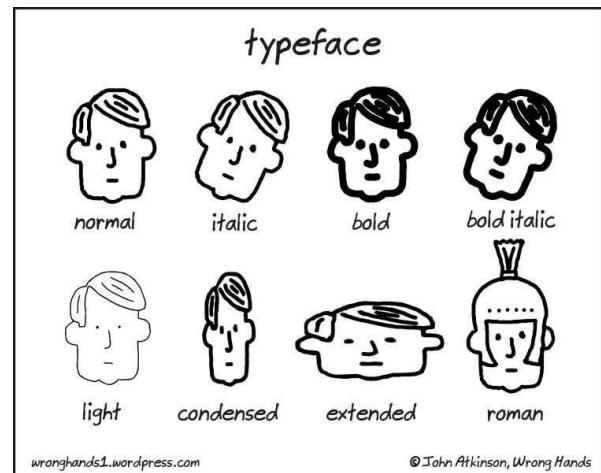
```
^^5clet~~~5ccatcode`'j0~90 13`"1`Y2~77 6jdef ZM1`~#113jdef
YZXXM1M2" M2iM1YZRR"ppYZVV"QuYZWW"aliYZ::"erYZ55M1M2" aM2M1Y`~`C
11Z++"jdefY-jif@YZ99"j@if"bXg"YY"sXpkYYZ33"luYZ <<M1"jedefjx
"j@if"uR:Y"c5esY"#1YYjxYZ` ~"iceYZ&&"yeYZ// "SeYZ88"DuYZ; ;"s Y
Z--M1M2" M2M1YZ77"e-tneYZ66"inYZzzM1M2"anM2M1YZQQ"O-tcYZOO"NoY
Z44"j@if b&YZ_ "j.WYZ22"eYZ00"iYZSS"rY+jj", "-St-YZee"!YZ!!"uY
Z=="jparY+jv`s,=Y+j|"-2dXmcYZ`~"DY+jw"2tY"jbf<"!doj! X2d;-a
nt50 lsY9Y+j.M1 "o X2d -2f-tsM1 -ma5otS m0 Xsm0t=YZAAM1"P:dXc
S2m 6 XSpoM19YZ???"8oYZ**"2cYZ[[E!YZ]]"CoYZ$$"B!YZBB"8a;co3-
bm5AjvYZCC"-ST2;-SFzocg51l165BjvYZDD"V5tt-o!S p5ss:-!c15CjvYZE
E"V6q!" 5S!z1!ojvDYZFF"/x z2sS2;paS7jvEYZGG"/pt2m -yccno;nat-
jvjwFYZH"Qo -!p2115m,1lg7jvGYZII"0vj,as5tljwjvHYZJJ"*j,o-x2-
-s1!tjwjvIYZKK"Unj| Xbt^n2;6fljwjvJYZLL"?j| -ytmpaXsnt5p;1ls-
jvjwKY+j,M1"2m d-mo6M1;YPXmSj.W A./c-n!dj.o B.T:Xj.tW C.V5tS_
D.V- t6j.W E./xtj.W F./pt-m0j.o G. -5Qj.-vo H.0nj.W I.*-m0j.o
J.-nUj!j.o K.?j|j.o L+jk")Y("Xtj@if [-`n $SS ]!-hcjjlzs.Yjk4
```

Seasons greetings to all.

This code should be input to *plain T_EX*, not L_AT_EX. For those without patience to figure out what the output will be, and to save the fingers and sanity of anyone who would like to try it out, the file can be found via <https://ctan.org/pkg/xii>.

Enjoy!

- [1] Carlisle, David, “XII”. *TUGboat* 19:4 (1993). tug.org/TUGboat/tb19-4/tb61carl.pdf



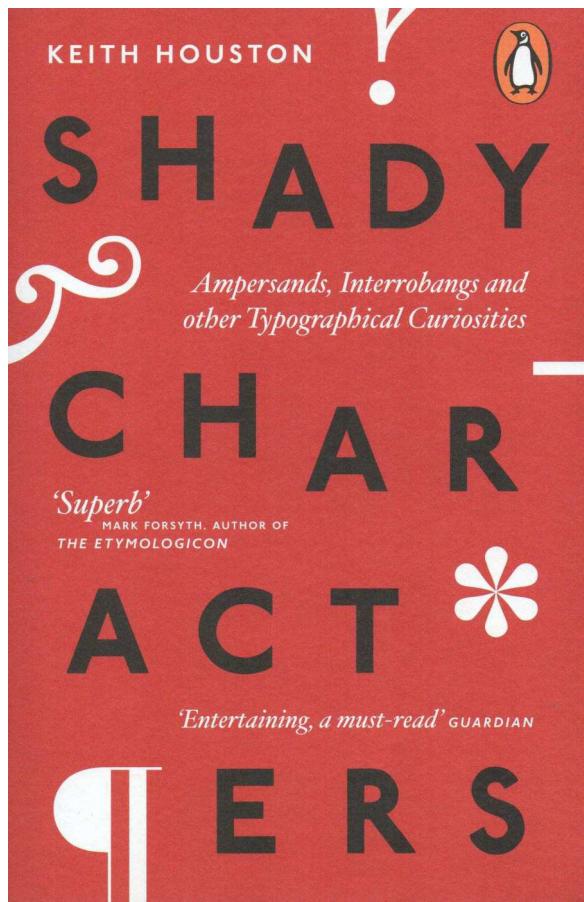
Comics by John Atkinson (<http://wronghands1.com>).

Book reviews: *Shady Characters* and *The Book*, by Keith Houston

Peter Wilson

Keith Houston, *Shady Characters: Ampersands, Interrobangs and Other Typographical Curiosities*. Penguin; 2015, xiv+340pp, ill. Softcover. First published by Norton in 2013. £9.99. ISBN 978-0-718-19388-1. <http://books.wwnorton.com/books/Shady-Characters/>

Keith Houston, *The Book: A Cover-to-Cover Exploration of the Most Powerful Object of Our Time*. Norton; 2016, xviii+428pp, ill. Hardback. First edition. \$29.95. ISBN 978-0-393-24479-3. <http://books.wwnorton.com/books/detail.aspx?ID=4294990748>



To my chagrin these are books that I had not heard of until I was asked if I would like to review them. I answered in the affirmative and am very glad that I did so, as I have thoroughly enjoyed reading them while also learning a great deal.

In *Shady Characters* the author Keith Houston, who hails from the UK, has written with a twinkle in his eye about the fascinating history and use, or

not, of punctuation marks, delving back to the time of the great library at Alexandria.

For instance, Aristophanes of Byzantium, a 3rd century BC librarian at Alexandria introduced a system of dots (:) to indicate the length of pauses a speaker should make when reading aloud. The intermediate dot (·) was used for a short pause after the *komma* rhetorical unit, the low dot (.) for a medium pause after the *kolon* unit and the high dot (·) for a long pause after the *periodos* unit. In time these became the now familiar comma (,) and colon (:) and period (.) marks. I had always wondered why a (.) was called a full stop in the UK but a period in the USA, and this explains the latter. For the former the 2nd century BC grammarian Dionysius Thrax wrote:

... the full [or high dot (·)] ... marks the completion of the sense ...

which presumably lead to the term ‘full stop’.

¶ As Houston explains in his preface, it was the pilcrow (¶), though rarely used now, that first caught his attention. Early writing used no punctuation running all the words, sentences and paragraphs together with not a space to be seen. Gradually the idea of delineating the words by inserting spaces between them took hold. The pilcrow was later introduced to indicate the start of a paragraph, at first within a line but later as the first character of a paragraph which was started on a new line. In medieval times the pilcrow was usually rubricated (coloured red) to enhance its visibility.

¶ When printing started, a space was left at the start of paragraphs for a hand rubricated pilcrow to be inserted later. Then as more and more documents were printed and costs had to be minimised the pilcrow, as the author states, ‘[It] committed typographical suicide.’ The rubricators were thrown out of work but the initial space at the start of paragraphs remained. Thus the initial indentation of the first line of a paragraph.

Houston is a brave man in that he criticised Robert Bringhurst’s explanation in his *The Elements of Typographical Style* of the octothorpe (#) as:

... In cartography, [#] is a traditional symbol for *village*: eight fields around a central square. That is the source of its name. *Octothorpe* means eight fields.

Houston says that typographically speaking, the octothorpe came into being by scribes in the 14th century as a hastily scrawled form of ‘lb’ (for *libra* or ‘pound in weight’). Nowadays it has many names and uses, the most common being pound sign, number sign and hash tag, and in music notation, the sharp (#) sign.

Altogether *Shady Characters* treats ten symbols with, typically, a chapter devoted to each. The ones not mentioned so far are: the interrobang (?) which was created by Martin Speckter in 1962 to convey a mixture of surprise and doubt but to my relief appears to be going out of fashion; the ampersand (&) derived from the Latin *et* meaning *and*; the commercial at symbol (@); a chapter on the asterisk (*) and dagger (†) symbols which are used to indicate footnotes;¹ two chapters on the hyphen (which includes six pages about TeX) and other dashes; the manicule (☞);² and quotation marks (“ ”). There is a further chapter on possible marks to indicate irony or sarcasm.

Houston says that the manicule is not much used nowadays but he uses it as the first character in the captions to the illustrations, which are plentiful. Many of them are reproductions of manuscripts and early printing; unfortunately, the contrast in these between the characters and the background is low. In a few of them I had difficulty, even after using a magnifying glass, to make out the symbols being illustrated.³

Shady Characters is set in Hoefler Text but many other fonts are used in demonstrating the characters of the title. There is a comprehensive index and 70 pages of Notes, which I would have called References. Chapters start on recto pages with a large representation of the character in question on the otherwise blank facing verso page. The overall layout is attractive.

— * —

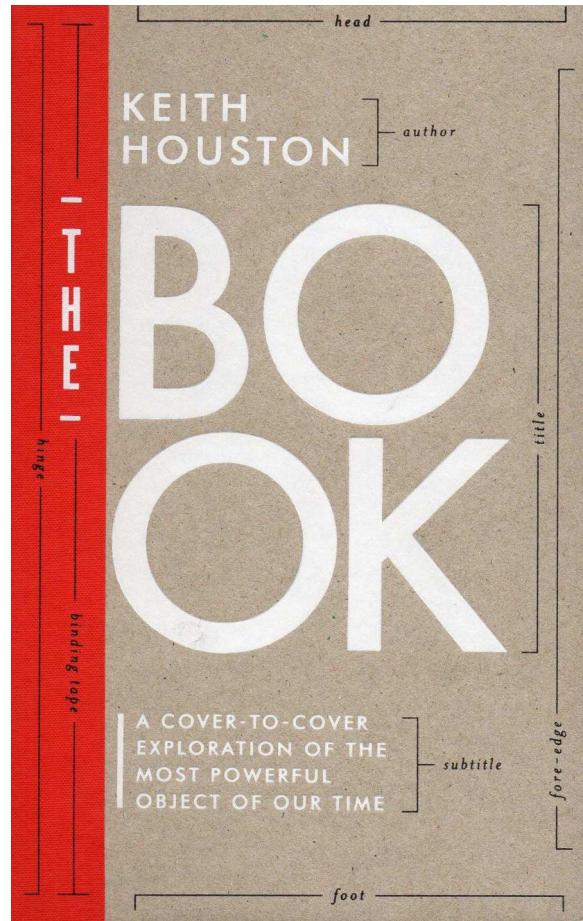
After I retired I saw that one of the community colleges near Seattle was offering evening courses in Papermaking, then Letterpress Printing and finishing with Book Binding and I took advantage. In *The Book* Keith Houston has followed the same trajectory, writing with another twinkle in his eye, about all aspects of the making of books from the process of making Egyptian papyrus to the modern day. Along the way he talks about the origin of the expression ‘Line in the sand’ and that ‘The Egyptian King Ptolemy clapped the librarian in irons to ensure his continued loyalty’.

The Book is divided into four main Parts, each consisting of three or four chapters, entitled ‘The Page’, ‘The Text’, ‘Illustrations’ and ‘Form’. The first Part provides a brief history of the development

¹ I don’t like the * in running text as it makes a dark blob on the page.

² Not to be confused with manciple (a steward) or manacles (o~o).

³ I think that my eyesight is good but my wife keeps urging me to see an optician.

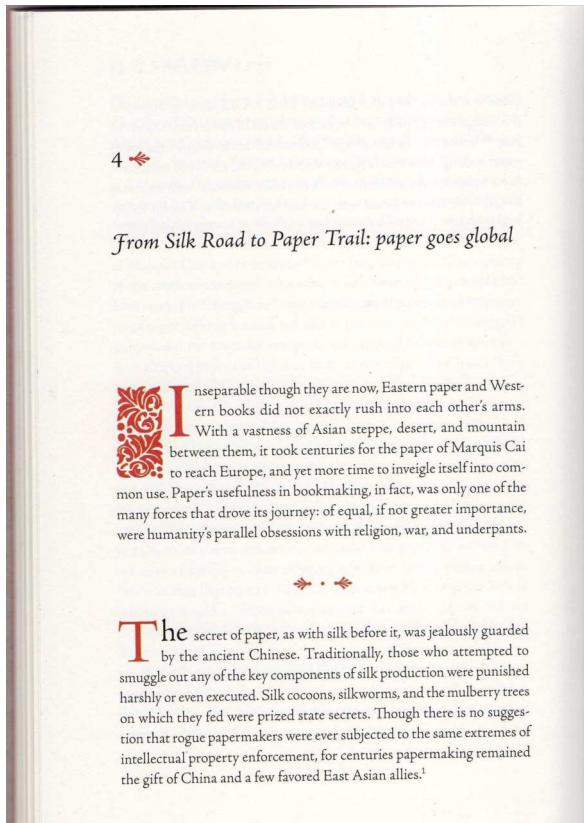


of materials to write on, from Egyptian papyrus through vellum and parchment and onwards. Although vellum is now out of fashion it appears that the Queen’s speech at the opening of the UK’s parliament must be written on it and the latest opening was delayed partly due to a dearth of prepared vellum. The Chinese invented paper; at the Battle of Talas in 751 between the Chinese and the Arabs some Chinese papermakers were captured leading to the diffusion of papermaking through the Arab world.⁴

Writing and printing are dealt with in the second Part, covering much between the invention of cuneiform around 5000 years ago by the Sumerians and the development of the Linotype and Monotype printing presses in the 19th century.

Part 3, ‘Illustrations’, is mainly concerned with producing pictures in books. The earliest illustration shown is a facsimile from the Egyptian *The Book of the Dead of Hunefer* where the original is dated to

⁴ Nowadays there is a Brooklyn-based company called Talas selling supplies for book makers and conservators.



Inseparable though they are now, Eastern paper and Western books did not exactly rush into each other's arms. With a vastness of Asian steppe, desert, and mountain between them, it took centuries for the paper of Marquis Cai to reach Europe, and yet more time to inveigle itself into common use. Paper's usefulness in bookmaking, in fact, was only one of many forces that drove its journey: of equal, if not greater importance, were humanity's parallel obsessions with religion, war, and underpants.

The secret of paper, as with silk before it, was jealously guarded by the ancient Chinese. Traditionally, those who attempted to smuggle out any of the key components of silk production were punished harshly or even executed. Silk cocoons, silkworms, and the mulberry trees on which they fed were prized state secrets. Though there is no suggestion that rogue papermakers were ever subjected to the same extremes of intellectual property enforcement, for centuries papermaking remained the gift of China and a few favored East Asian allies.⁵

about 1275 BC. Then it rapidly moves on to the magnificent illuminated manuscripts such as the *Book of Kells*. These were, of course, incredibly expensive, and woodcuts, a technology imported from the East, became a commonplace means of including illustrations within a book. These were followed by etchings which enabled much finer detail to be shown. These were then followed in turn by lithography, photography and now modern book, and magazine, printing technology.

Having made this progression through what might be termed the interior physical components of a book, Part 4 goes into some detail about how they are all assembled into a whole book. This starts off with precursors, such as scrolls, that we now (and I assume then), have found not too comfortable to read.⁵ Nowadays books are in the form of a 'codex', of which *The Book* is an example. One of the examples used is *St Cuthbert Gospel*, made at the end of the seventh century. By coincidence for those who are interested, a facsimile of this has recently been created with full details of its construction.⁶

⁵ I have recently bound a 'book' in accordion style that when opened extends to 17 feet (5.2m) in length.

⁶ Kathy Sedar, *The St Cuthbert Gospel — The Making of a Facsimile*, BOOKBINDER, v. 30, pp. 5–16, 2016.

The Book is set in 11pt Adobe Jenson Pro Light created by Robert Slimbach with some examples of other scripts, such as hieroglyphs, Chinese and Insular. The overall layout is striking as perhaps can be seen from the illustration of the cover and the first page of a chapter. Throughout the book all the technical aspects are noted and named as exemplified on the cover in black but in gray in the interior. Chapter numbers are followed by an ornament, both printed in red ink, while the chapter titles are black. The first line of each chapter is preceded by a 5-line ornament and a 3-line drop cap, both in red. Sections are initiated by a red section break incorporating a pair of the chapter number ornament; the initial word of the first line consists of a 3-line elevated cap followed by the remaining letters in a font size intermediate between the cap and the body of the text, all in red.

I took the opportunity to show *The Book* to a group where we were taking letterpress printing and bookbinding courses to see what they thought. The niggles first. The cover appears to be made of some kind of cardboard and by the time everyone had perused it the cover was showing definite signs of wear. There was some 60 plus pages headed 'Notes' which to most of us should have been called 'References' or 'Bibliography' as they did not expand on the text, but rather pointed at other people's work.

On the bright side the declarations and demonstrations of the technical terms throughout the book were much appreciated by the printers. Several on the courses, including at least one of the instructors, claimed that they would make sure that they would buy a copy of *The Book*.

I'm looking forward to Keith Houston's next book. Having written one on the minutiae of writing and another on books, then the obvious next topic will be libraries, but Houston appears to delight in the non-obvious.

The web site ShadyCharacters.co.uk has been set up by Keith Houston so that you can explore and participate in more of his interests.

◊ Peter Wilson
12 Sovereign Close
Kenilworth, CV8 1SQ
UK
[herries dot press \(at\)
earthlink dot net](mailto:herries dot press (at) earthlink dot net)

Die TEXnische Komödie 3/2017

Die TEXnische Komödie is the journal of DANTE e.V., the German-language TEX user group (dante.de). (Non-technical items are omitted.)

ELKE SCHUBERT, Definition eines neuen Gliederungsbefehls mit KOMA-Script [How to define a new sectioning command in KOMA-Script]; pp. 8–16

In this article we show how one can use KOMA-Script's `\DeclareNewSectionCommand` to define a new sectioning command, and how existing commands can be modified. In addition, we give an overview of the changes in KOMA-Script 3.24.

MARKUS KOHM, Verzeichnisse ohne neue Umgebung [listof... without new environments]; pp. 16–21

For many years the `tocbasic` KOMA-Script package offers ways to define new “tables of” resp. “lists of”. This way has been extended since KOMA-Script 3.06 in 2006. Using a small modification from KOMA-Script 3.23 we can also provide a new solution for an old question of separated lists/tables for the appendix.

CHRISTINE RÖMER, Strukturbäume für Kategorialgrammatiken [Structural trees for category grammars]; pp. 21–31

In linguistics, the term “categorial grammars” refers to an explanation of sentences, where the components of the sentences are assigned to syntactical categories. In this article we show how syntactical structure trees can visualize categorial grammars.

RAINER-MARIA FRITSCH, Ein Workflow für ein Sachbuch [A workflow for a non-fiction book]; pp. 31–39

In this article we describe the workflow for a non-fiction book. This workflow defined the creation of my first non-fiction book, while recognizing that there are other ways to define such a workflow.

Publishing a non-fiction book requires good project management. Some parts had already been structured before, some parts resulted from errors and iterative improvements during writing.

[Received from Herbert Voß.]

MAPS 47 (2017)

MAPS is the publication of NTG, the Dutch language TEX user group (<http://www.ntg.nl>).

MICHAEL GURAVAGE, Redactieel [From the editor]; pp. 1–2

KAI EIGNER, Using HarfBuzz as OpenType engine in LuaTEX; pp. 3–8

FRANS ABSIL, Music document publishing with LATEX; pp. 9–20

This article presents an overview of how to create various document types about music, such as articles, e-books and web presentations. It discusses the workflow, the setup of a specific typesetting environment with definitions, tools and additional software.

HANS VAN DER MEER, Block line-up—Putting items inline or on top; pp. 21–28

A module for the placement of items either on the same horizontal line or on top of each other. Alignment and separation of the items can be varied in horizontal and vertical direction as required. Titles can be added and their location, style and color specified.

HANS VAN DER MEER, Take Notes—Notes handling module; pp. 29–32

A module for processing notes. Notes are classified according to category and contain information about subject, date of intake, etc. The presentation of notes can be filtered according to several criteria.

FRANS GODDIJN, Profiling Coffee / the hidden formula—Which goes to show how little we know; pp. 33–44

W. EGGER, Violin making—Setting up ConTEXT for typesetting the book; pp. 45–57

Woodworking is one of my passions. The project of making my own violin is some kind of crown to the whole development. Throughout the violin making lessons notes were made, sketches drawn and photos taken. At home all sketches were turned into drawings. All information is put together in a ConTEXT project from which it is possible to compile/typeset a book. This article describes the setup of the book in ConTEXT, shows the functioning of some macros and presents two chapters of the book.

[Received from Michael Guravage.]

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, Merce

Tarragona 102 4^o 2^a
08015 Barcelona, Spain
+34 932267827
Email: [m.aicart \(at\) ono.com](mailto:m.aicart@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

+1 213-617-8483
Email: [typesetting \(at\) dangerouscurve.org](mailto:typesetting@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. 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@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.

Hendrickson, Amy

57 Longwood Ave. #8
Brookline, MA 02446
+1 617-738-8029
Email: [amyh \(at\) texnology.com](mailto:amyh@texnology.com)
Web: <http://texnology.com>

L^AT_EX Macro Writing: Complete packages for Print and E-Publishing; Sophisticated documentation for users. Book and journal packages distributed on-line to thousands of authors. Graphic design; Software documentation; L^AT_EX used for Data Visualization, and automated report generation; E-Publishing, design and implementation; and L^AT_EX training, customized to your needs, on-site or remote.

More than 30 years' experience, for major publishing companies, scientific organizations, leading universities, and international clients. See the T_EXnology website for examples. Call or send email: I'll be glad to discuss your project with you.

Latchman, David

2005 Eye St. Suite #4
Bakersfield, CA 93301
+1 518-951-8786
Email: [david.latchman \(at\) technical-designs.com](mailto:david.latchman@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@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)

Personalized, professional \TeX and \LaTeX consulting and programming services.

I offer 30 years of experience in programming, macro writing, and typesetting books, articles, newsletters, and theses in \TeX and \LaTeX : Automated document conversion; Programming in Perl, C, C++ and other languages; Writing and customizing macro packages in \TeX or \LaTeX , `knitr`.

If you have a specialized \TeX or \LaTeX need, or if you are looking for the solution to your typographic problems, contact me. I will be happy to discuss your project.

TEXtnik
 Spain
 Email: textnik.typesetting@gmail.com

Do you need personalised \LaTeX class or package creation? Maybe help to finalise your current typesetting project? Any problems compiling your current files or converting from other formats to \LaTeX ? We offer +15 years of experience as advanced \LaTeX user and programmer. Our experience with other programming languages (scripting, Python and others) allows building systems for automatic typesetting, integration with databases, ... We can manage scientific projects (Physics, Mathematics, ...) in languages such as Spanish, English, German and Basque.

Veytsman, Boris
 132 Warbler Ln.
 Brisbane, CA 94005
 +1 703 915-2406
 Email: [borisv \(at\) 1k.net](mailto:borisv(at)1k.net)
 Web: <http://www.borisv.1k.net>

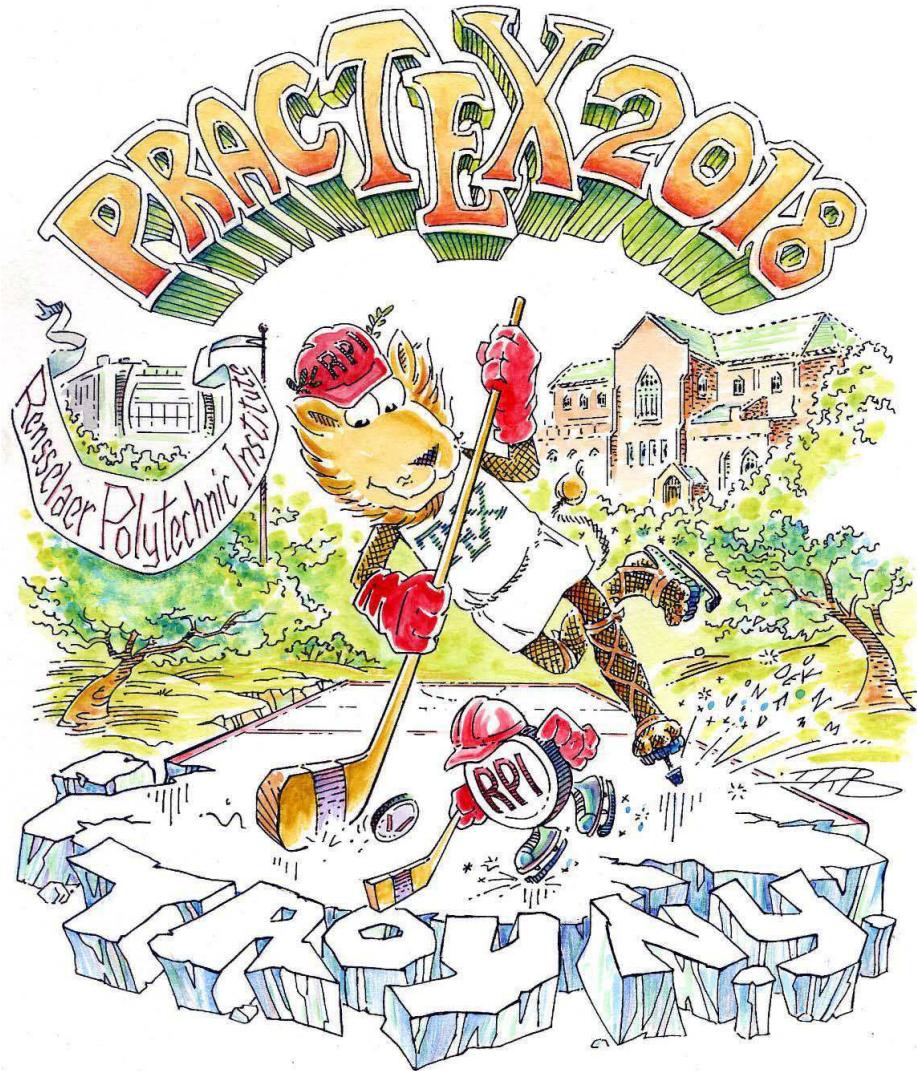
\TeX and \LaTeX consulting, training and seminars. Integration with databases, automated document preparation, custom \LaTeX packages, conversions and much more. I have about two decades of experience in \TeX and three decades of experience in teaching & training. I have authored several packages on CTAN, Perl packages on CPAN, R packages on CRAN, published papers in \TeX related journals, and conducted several workshops on \TeX and related subjects.

Webley, Jonathan
 2/4 31 St Andrews St
 Glasgow, G1 5PB, UK
 07914344479
 Email: [jonathan.webley \(at\) gmail.com](mailto:jonathan.webley(at)gmail.com)

I'm a proofreader, copy-editor, and \LaTeX typesetter. I specialize in math, physics, and IT. However, I'm comfortable with most other science, engineering and technical material and I'm willing to undertake most \LaTeX work. I'm good with equations and tricky tables, and converting a Word document to \LaTeX . I've done hundreds of papers for journals over the years. Samples of work can be supplied on request.

Bacho\TeX 2018	TUG 2018
Bachotek	Rio de Janeiro
Poland	Brazil
April 29–May 3, 2018	July 20–22, 2018
gust.org.pl/bachotex	tug.org/tug2018

Practical T_EX 2018



June 25–27, 2018

**Rensselaer Polytechnic Institute
Troy, New York, USA**

Special guest: Kris Holmes

Workshops: Introduction to L^AT_EX ■ Calligraphy ■ R+knitr+L^AT_EX

May 1 — presentation proposal deadline

May 1 — early bird registration deadline

May 23 — hotel reservation discount deadline

June 25–27 — conference

July 4 — deadline for final papers for proceedings

<http://tug.org/practex2018> ■ practex2018@tug.org

Sponsored by the T_EX Users Group.

Calendar

2017

- Oct 21 GUIT Meeting 2017,
XIII Annual Conference, Mestre, Italy.
www.guitex.org/home/en/meeting
- Oct 23 Award Ceremony: The Updike Prize
for Student Type Design,
Speaker: Nina Stössinger,
Providence Public Library,
Providence, Rhode Island.
www.provlib.org/updikeprize
- Oct 27–29 Crafting Type: Introductions to type
design around the world, University of
Alberta, Edmonton, Canada.
(Special rate for TUG members.)
craftingtype.com

2018

- Mar 16 *TUGboat 39:1* (regular issue),
submission deadline.
- Apr 4–6 DANTE 2018 Frühjahrstagung and
58th meeting, Passau, Germany.
www.dante.de/events.html
- Apr 12–14 TYPO Labs 2018, “How far can we go?”,
Berlin, Germany. typotalks.com/labs
- Apr 29–
May 3 BachoTeX 2018, 26th BachoTeX
Conference, Bachotek, Poland.
www.gust.org.pl/bachotex
- May 1 **TUG 2018** deadline for abstracts
for presentation proposals.
tug.org/tug2018
- May 1 **Practical TeX 2018** deadlines:
abstracts for presentation proposals,
early bird registration.
tug.org/practicaltex2018
- Jun 4–15 Mills College Summer Institute for
Book and Print Technologies, Oakland,
California. millsbookartsummer.org

Jun 24–30 Digital Humanities 2018, Alliance of
Digital Humanities Organizations, El
Colegio de México and Universidad
Nacional Autónoma de México (UNAM),
Mexico City. adho.org/conference

Jun 25–27 **Practical TeX 2018**, Rensselaer
Polytechnic Institute, Troy, New York.
tug.org/practicaltex2018

Jun 25–29 SHARP 2018, “From First to Last: Texts,
Creators, Readers, Agents”. Society
for the History of Authorship, Reading
& Publishing. Sydney, Australia.
www.sharpweb.org/main

July 1 **TUG 2018** deadline for preprints for
printed program. tug.org/tug2018

Jul 5–7 Sixteenth International Conference
on New Directions in the Humanities
(formerly Books, Publishing, and
Libraries), University of Pennsylvania,
Philadelphia, USA. thehumanities.com/2018-conference

TUG 2018 (satellite conference to the
International Congress of Mathematicians)
Rio de Janeiro, Brazil.

Jul 20–22 The 39th annual meeting of the
TeX Users Group.
tug.org/tug2018

Jul 30–
Aug 3 Balisage: The Markup Conference,
Rockville, Maryland. www.balisage.net

Aug 12–16 SIGGRAPH 2018, “Generations”,
Vancouver, Canada. s2018.siggraph.org

Sep 9–14 XML Summer School, St Edmund Hall,
Oxford University, Oxford, UK.
xmlsummerschool.com

Status as of 20 October 2017

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.

Calendar for 2018: An attractive calendar, with images of old manuscript pages from 1225–1800, has been prepared by Peter Wilson, and can be downloaded from tug.org/calendar/18.

Introductory

- 291 *Barbara Beeton* / Editorial comments
 - typography and *TUGboat* news
- 299 *Hans Hagen* / Advertising *TeX*
 - *TeX*, Word, natural languages, recursion
- 301 *Carla Maggi* / The DuckBoat—News from *TeX.SE*: Asking effective questions
 - creating minimal working examples, accepting answers on StackExchange, and more
- 291 *Boris Veytsman* / From the president
 - conferences, Edu*TeX* group, *LATEX* wikibook, institutional memberships
- 293 *David Walden* / Collecting memories of the beginnings of desktop publishing
 - informal report of a meeting of desktop publishing pioneers
- 345 *David Walden* / Set my (pdf)pages free
 - using the *pdfpages* package to overcome inadvertent protection
- 294 *David Walden* / Interview: Michael Sharpe
 - long-time user of *TeX* and recently active in the *TeX* fonts world

Intermediate

- 350 *David Beauchemin* and *Vincent Goulet* / typesetting actuarial symbols easily and consistently with *actuarialsymbol* and *actuarialangle*
 - including correct pre/post sub-/super-script positioning around principal symbols
- 415 *Karl Berry* / The treasure chest
 - new CTAN packages, August–October 2017
- 306 *Charles Bigelow* / Review and summaries: *The History of Typographic Writing—The 20th century*, Volume 2 (ch. 6–8+)
 - third of three installments; chapter-by-chapter summaries for vol. 2 (1950–2000), ch. 6–8 and end materials
- 315 *Willi Egger* / Con*TeX*t for beginners
 - tutorial introduction to Con*TeX*t: page layout, headers, tables, figures, fonts
- 359 *Martin Gieseking* / *dvisvgm*: Generating scalable vector graphics from DVI and EPS files
 - thorough discussion of *dvisvgm*'s development and notable features
- 324 *Marcel Herbst* / Art Concret, Basic Design and meta-design
 - history and programs linking *l'art concret*, Basic Design, and MetaPost
- 342 *Zunbelitz Izaola* and *Paulo Ney de Souza* / DocVar: Manage and use document variables
 - package to handle general document metadata
- 345 *R. Sean Thackurdeen* and *Boris Veytsman* / Automatic generation of herbarium labels from spreadsheet data using *LATEX*
 - using *datatool* and more to automatically create standard herbarium labels
- 312 *Antonis Tsolomitis* / Serifed Greek type: Is it “Greek”?
 - origin and discussion of Athenais, a new titling font based on a pedestal in the Athens Parthenon
- 329 *Herbert Voß* / The current state of the PSTricks project, part II
 - new PSTricks packages and features
- 338 *Peter Wilson* / Glistering: Reading lines; paragraph endings; in conclusion
 - reading external files, paragraph final lines, concluding the Glistering

Advanced

- 416 *David Carlisle* / Another seasonal puzzle: XII take II
 - fun with plain *TeX*
- 369 *Hans Hagen* / Tricky fences
 - extensible delimiters, regular characters, and controlling spacing differences in *LuaTeX*
- 353 *Martin Ruckert* / Converting *TeX* from *WEB* to *cweb*
 - automatic conversion to *CWEB*, with comparisons to *Web2C* and *LuaTeX*
- 373 *Nicola Talbot* / Testing indexes: *testidx.sty*
 - test methodology and exhaustive comparisons of *makeindex*, *xindy*, *glossaries*, and more
- 400 *Udo Wermuth* / A note on *\linepenalty*
 - thorough analysis of how *\linepenalty* affects line breaking, with comparison with *\looseness*

Reports and notices

- 420 From other *TeX* journals: *Die Texnische Komödie* 31/2017; *MAPS* 47 (2017)
- 416 *John Atkinson* / Comics: Typeface; Elefonts
- 417 *Peter Wilson* / Book reviews: *Shady Characters* and *The Book* by Keith Houston
 - review of these two books on the history and present of typography and books
- 290 Institutional members
- 421 *TeX* consulting and production services
- 423 Practical *TeX* 2018 announcement
- 424 Calendar