

# TUGBOAT

Volume 36, Number 3 / 2015

<b>General Delivery</b>	179	From the Board of Directors / <i>TeX Users Group Board</i>
	180	Editorial comments / <i>Barbara Beeton</i> Help wanted—The UK <i>TeX</i> FAQ; <i>B&amp;H</i> and the Wingdings font; Choice of font <i>does</i> make a difference; Safe <i>TeX</i> ; More help wanted: Unicode symbol names in languages other than English; What is a template?; Still more help wanted: The <i>L<sup>A</sup>TeX</i> Wikibook
	182	Adrian Frutiger, 1928–2015 / <i>Norbert Preining</i>
	184	Thomas Koch, 1964–2014 / <i>Joachim Schrod</i>
	185	DANTE e.V. 2015 meeting reports / <i>Stefan Kottwitz</i>
<b>Education</b>	188	<i>TeX</i> in schools: Just Say Yes! / <i>Simon Laube</i>
<b>Fonts</b>	190	Smoky letters / <i>Linus Romer</i>
	191	About the DK versions of Lucida / <i>Charles Bigelow</i>
<b>Publishing</b>	200	History of cookbooks / <i>Taco Hoekwater</i>
<b>Typography</b>	208	Typographers' Inn / <i>Peter Flynn</i>
<b>L<sup>A</sup>TeX</b>	210	<i>L<sup>A</sup>TeX</i> news, issue 22, January 2015 / <i>L<sup>A</sup>TeX Project Team</i>
	212	<i>L<sup>A</sup>TeX</i> news, issue 23, October 2015 / <i>L<sup>A</sup>TeX Project Team</i>
	214	Introduction to list structures in <i>L<sup>A</sup>TeX</i> / <i>Thomas Thurnherr</i>
	217	<i>gradstudentresume</i> : A document class for graduate student CVs / <i>Anagha Kumar</i>
	220	Glisternings: Longest string; Marching along; A blank argument; A centered table of contents / <i>Peter Wilson</i>
	227	Chemistry in <i>L<sup>A</sup>TeX2<sub>ε</sub></i> —an overview of existing packages and possibilities / <i>Clemens Niederberger</i>
<b>Software &amp; Tools</b>	234	Automating <i>L<sup>A</sup>TeX(3)</i> testing / <i>Joseph Wright</i>
	237	Two applications of SWIGLIB: GraphicsMagick and Ghostscript / <i>Luigi Scarso</i>
<b>Macros</b>	243	Typesetting the “Begriffsschrift” by Gottlob Frege in plain <i>TeX</i> / <i>Udo Wermuth</i>
	257	GMOA, the ‘General Manipulation Of Arguments’: An extension to the <i>13expan</i> package of the <i>expl3</i> bundle and language / <i>Grzegorz Murzynowski</i>
<b>Hints &amp; Tricks</b>	268	Production notes / <i>Karl Berry</i>
	269	The treasure chest / <i>Karl Berry</i>
<b>Reviews</b>	271	An online glossary of typographic terms by Janie Kliever / <i>Boris Veytsman</i>
<b>Abstracts</b>	272	GUST: EuroBachoTeX 2015 proceedings
	273	<i>Die TExnische Komödie</i> : Contents of issue 4/2015
<b>Cartoon</b>	274	Jim Benton / <i>A summons</i>
<b>TUG Business</b>	274	TUG institutional members
<b>Advertisements</b>	275	<i>TeX</i> consulting and production services
<b>News</b>	276	Calendar

## T<sub>E</sub>X Users Group

*TUGboat* (ISSN 0896-3207) is published by the T<sub>E</sub>X Users Group. Web: <http://tug.org/TUGboat>.

### Memberships and subscriptions

2015 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.

Membership in the T<sub>E</sub>X 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 is open only to named individuals, and carries with it such rights and responsibilities as voting in TUG elections. For membership information, visit the TUG web site.

Rates will be the same in 2016. Anyone joining or renewing before March 31 receives a \$20 “early bird” discount:

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

Also, (non-voting) *TUGboat* subscriptions are available to organizations and others wishing to receive *TUGboat* in a name other than that of an individual. The subscription rate for 2016 will be \$110.

### Institutional membership

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

### Trademarks

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

T<sub>E</sub>X is a trademark of American Mathematical Society.  
METAFONT is a trademark of Addison-Wesley Inc.  
PostScript is a trademark of Adobe Systems, Inc.

[printing date: November 2015]

Printed in U.S.A.

## Board of Directors

Donald Knuth, *Grand Wizard of T<sub>E</sub>X-arcana*<sup>†</sup>

Kaveh Bazargan, *President\** (suspended)

Jim Hefferon\*, *Vice President*

Karl Berry\*, *Treasurer*

Susan DeMeritt\*, *Secretary*

Pavneet Arora

Barbara Beeton

Kaja Christiansen

Michael Doob

Steve Grathwohl

Klaus Höppner

Steve Peter

Cheryl Ponchin

Norbert Preining

Arthur Reutenauer

Boris Veytsman

Raymond Goucher, *Founding Executive Director*<sup>†</sup>

Hermann Zapf (1918–2015), *Wizard of Fonts*<sup>†</sup>

\*member of executive committee

<sup>†</sup>honorary

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

### Addresses

T<sub>E</sub>X Users Group

P. O. Box 2311

Portland, OR 97208-2311

U.S.A.

### Electronic Mail

(Internet)

General correspondence,  
membership, subscriptions:  
[office@tug.org](mailto:office@tug.org)

### Telephone

+1 503 223-9994

Submissions to *TUGboat*,  
letters to the Editor:  
[TUGboat@tug.org](mailto:TUGboat@tug.org)

### Fax

+1 815 301-3568

Technical support for  
T<sub>E</sub>X users:  
[support@tug.org](mailto:support@tug.org)

### Web

<http://tug.org/>

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

Contact the  
Board of Directors:  
[board@tug.org](mailto:board@tug.org)

Copyright © 2015 T<sub>E</sub>X 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<sub>E</sub>X Users Group must approve translations of this permission notice itself. Lacking such approval, the original English permission notice must be included.

[The Belgian typographer Fernand Baudin] claims that literacy involves not only a knowledge of how to form letters and assemble them into strings, but also an understanding of how to organize the elements of the text into a coherent visual structure.

Charles Bigelow, in the Foreword to *How Typography Works (and why it is important)*, by Fernand Baudin (1989)

# TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP

EDITOR BARBARA BEETON

VOLUME 36, NUMBER 3

PORTLAND

•

OREGON

•

•

2015

U.S.A.

### **TUGboat editorial information**

This regular issue (Vol. 36, No. 3) is the last issue of the 2015 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 (<http://tug.org/store>), and online at the *TUGboat* web site, <http://tug.org/TUGboat>. Online publication to non-members is delayed up to one year after print publication, to give members the benefit of early access.

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

### **Submitting items for publication**

Proposals and requests for *TUGboat* articles are gratefully received. Please submit contributions by electronic mail to [TUGboat@tug.org](mailto:TUGboat@tug.org).

The first issue for 2016 will be a regular issue, with a deadline of March 11, 2016. The second 2016 issue will be the proceedings of the TUG'16 conference (<http://tug.org/tug2016>); the deadline for receipt of final papers is August 8. The third issue deadline is September 29.

The *TUGboat* style files, for use with plain *TeX* and *L<sup>A</sup>TeX*, are available from CTAN and the *TUGboat* web site, and are included in common *TeX* distributions. We also accept submissions using ConTeXt. Deadlines, templates, tips for authors, and other information is available at:

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

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

### **TUGboat editorial board**

Barbara Beeton, *Editor-in-Chief*

Karl Berry, *Production Manager*

Boris Veytsman, *Associate Editor, Book Reviews*

### **Production team**

William Adams, Barbara Beeton, Karl Berry,

Kaja Christiansen, Robin Fairbairns,

Robin Laakso, Steve Peter, Michael Sofka,

Christina Thiele

### **Other TUG publications**

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

If you have such items or know of any that you would like considered for publication, send the information to the attention of the Publications Committee at [tug-pub@tug.org](mailto:tug-pub@tug.org).

### **TUGboat advertising**

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

### **Trademarks**

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

*METAFONT* is a trademark of Addison-Wesley Inc.

*PostScript* is a trademark of Adobe Systems, Inc.

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

---

**TUG 2016 ~ Toronto, Canada**  
**July 25–27, 2016,**  
**excursions before and after**  
**<http://tug.org/tug2016>**

---

**From the Board of Directors**

TUG Board of Directors

*(This letter was sent to all TUG members on October 13, 2015, and posted at <http://tug.org/election> shortly thereafter. We are reprinting it, as sent, here in TUGboat since it potentially affects the entire TeX community. Questions or comments can be sent to [board@tug.org](mailto:board@tug.org).)*

— \* —

We, as directors of the TeX Users Group, write to you about a difficult decision we have had to make. With regret, we announce that we have voted to suspend Kaveh Bazargan as TUG President, effective immediately. We need to explain this, and give the reasons for our actions.

As you know, Kaveh Bazargan was elected President of TUG this last spring. We directors were looking forward to working with Kaveh on widening TUG's audience and reach.

The TUG President, like the TUG directors, has the privilege and obligation of representing the interests of all TUG members to the best of his or her abilities.

Unfortunately, at the time of the election, unbeknownst to the directors, Kaveh was involved in a lawsuit against another member of TUG. The lawsuit remains ongoing today, and involves TeX-related business activities by the parties. The existence of the lawsuit was not disclosed at the time of the election. (Of course, the TUG organization has no standing, and takes no position whatsoever, on the merits of either party's case.)

In our opinion, legal entanglement with another TUG member is a clear conflict of interest, preventing proper fulfillment of the President's duties. Any decision made or initiative undertaken by the TUG President while pursuing a lawsuit against another TUG member would, at the very least, appear to be tainted.

In addition, the directors were informed that the fact of Kaveh's election as TUG President was included in documents submitted to the court. We believe that TUG should not take sides, or even appear to take sides, in a lawsuit to which it is not a party. Such an implication appears inevitable to us when the TUG presidency is used as a material fact in a court case.

Thus, we asked Kaveh to voluntarily suspend his presidency for the duration of the lawsuit and any related legal matters. We were not successful in convincing him that this would be best for TUG. Further, he neither made an explanation as to why

he did not reveal the existence of the lawsuit at the time of the election, nor made any offer to mitigate its effects now.

Therefore, we concluded that the actions of the President were not in the interests of TUG. To fulfill our responsibilities as TUG directors, we felt we needed to act according to Article IV, Section 5, of the TUG bylaws (<http://tug.org/bylaws>):

A Board member who by action or inaction shall be deemed to be no longer working in the interests of TUG may be suspended as Director by a vote of the entire Board, provided that at least 75% of the Board votes in favor of suspension.

As 14 members of the Board, out of a total of 16, voted for his suspension (one director abstained for health reasons, and the 16th board member is the President), hence 87.5% which is greater than the 75% required, Kaveh Bazargan is now suspended from the office of TUG President. He has the right to appeal his suspension according to the TUG bylaws.

A final note: this decision to suspend the sitting President was perhaps the most difficult made by any of us in the course of our involvement with TUG. We carefully reviewed the bylaws and the entire record of correspondence with Kaveh, and intensely deliberated over a period of weeks, ever since becoming aware of the issue. We communicated our concerns to Kaveh several times, and we took into consideration that Kaveh received the majority of votes in an independent election and was thus entitled to substantial regard. We would like the TUG membership, and the wider TeX community, to know that we made our best efforts to arrive at a solution that was respectful of the election results, while also recognizing the expectations of conduct from all of its officers.

Ultimately, as the duly-chosen directors of the TeX Users Group, we made the decision for suspension in the belief that it is the course of action that is best for TUG.

Sincerely,

Pavneet Arora ■ Barbara Beeton ■ Karl Berry  
 ■ Kaja Christiansen ■ Sue DeMeritt ■ Michael Doob  
 ■ Steve Grathwohl ■ Jim Hefferon ■ Klaus Höppner  
 ■ Steve Peter ■ Cheryl Ponchin ■ Norbert Preining  
 ■ Arthur Reutenauer ■ Boris Veytsman

— \* —

*(Addendum: per the TUG bylaws, while the suspension of the president is effective, the vice-president assumes the duties of the office of president.)*

◊ TUG Board of Directors  
[board@tug.org](mailto:board@tug.org)

## Editorial comments

Barbara Beeton

### Help wanted — The UK T<sub>E</sub>X FAQ

An era has come to an end. Robin Fairbairns, long-time keeper of the CTAN node at the University of Cambridge (UK) has retired, and with this, the Cambridge CTAN node itself.

Hosting of CTAN has been consolidated on the DANTE server, but a separate project hosted at Cambridge, the UK T<sub>E</sub>X FAQ, which Robin has lovingly curated for many years, has been relocated. The node name `tex.ac.uk` has been retained, and is now resident on a new server. The announcement of the move can be read at [uk.tug.org/2015/07/07/tex-ac-uk-server-on-the-move/](http://uk.tug.org/2015/07/07/tex-ac-uk-server-on-the-move/).

Joseph Wright reported to TUG 2015 on the “State of the (UK-)T<sub>E</sub>X FAQ” ([river-valley.zeeba.tv/state-of-the-uk-tex-faq/](http://river-valley.zeeba.tv/state-of-the-uk-tex-faq/)). Going into his well deserved retirement, Robin left behind a long list of FAQ topics that deserve updating. Even if everything on the list is taken care of, the list will continue to grow. In order to ensure its continued validity and usefulness, assistance is solicited for its maintenance. Volunteers can make themselves known by sending a message to the site managers at `faq-devel@tex.ac.uk`.

Many thanks to Robin for his devoted and careful work on the FAQ (and for many other things as well), and the best of wishes for a long and happy retirement.

### Bigelow & Holmes and the Wingdings font

Dingbats have been in the repertoire of printers since the creation of movable type, and even before that in manuscripts (although they weren’t called dingbats then). Also known as fleurons, they are used to indicate breaks in continuity, highlight important points in a text, occupy otherwise empty space in an attractive manner, form borders (see Peter Wilson’s “Glisterings” column in *TUGboat* 32:2, pages 202–205), and plenty more.

The best known font of dingbats is probably the one by Hermann Zapf, one of the “base 35” PostScript fonts; others exist as well. When a font of dingbats was wanted for Windows, Microsoft bought the rights to Bigelow & Holmes’ Lucida Icons, Lucida Arrows, and Lucida Stars, then combined its favorites into a single font. The name “Wingdings” was coined, combining the two words, “Windows” and “Dingbats”. The font became an unexpected success.

Read the story here: [www.vox.com/2015/8/25/9200801/wingdings-font-history](http://www.vox.com/2015/8/25/9200801/wingdings-font-history).

### Choice of font *does* make a difference

The *Nature* website, on 29 October 2015, reported a “Grant application rejected over choice of font” ([www.nature.com/news/grant-application-rejected-over-choice-of-font-1.18686](http://www.nature.com/news/grant-application-rejected-over-choice-of-font-1.18686)).

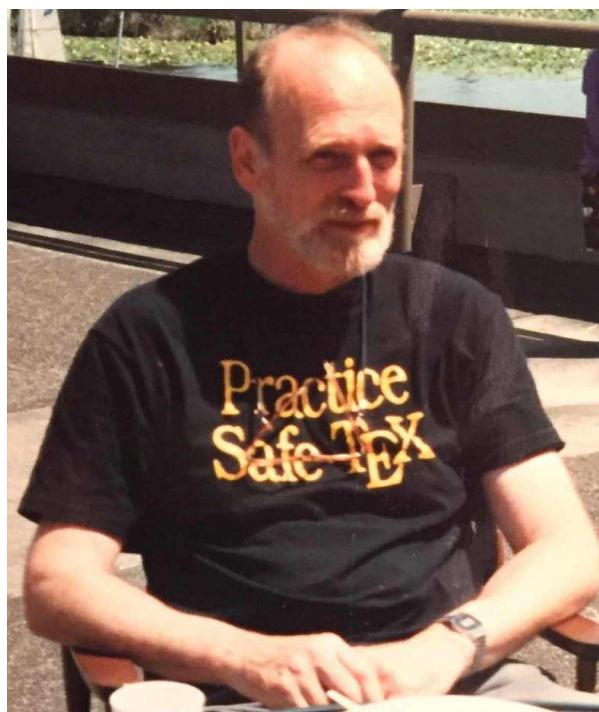
Briefly, submission guidelines (which were updated between the time the applicant started writing her proposal and the time it was submitted, changing the relevant requirement) specified a particular font. A different font used on a fellowship application had more compact letter widths, which could result in more text fitting within the allowed page limits. The application was summarily rejected on grounds that had nothing to do with the technical merits of the proposal.

Moral of story: Be *very* careful to follow instructions to the letter when submitting a proposal asking for funding.

A sidebar on the page revisits the kerfuffle raised by a review comparing Word and L<sub>A</sub>T<sub>E</sub>X reported in my column earlier this year (*TUGboat* 36:1, page 5).

### Safe T<sub>E</sub>X

Pierre MacKay was always willing to support a good cause. These t-shirts appeared at a TUG meeting in the mid-1980s. Maybe some of you old-timers remember the stir they caused.



Thanks to Diana Wright for providing the photo.

## More help wanted: Names of Unicode symbols in languages other than English

In support of assistive technology and accessibility for math, I've been asked whether there is any "official", or even organized effort to provide translations of Unicode names into languages other than English. The specific request was about German; for example, for U+2A0C one would look for translations of "quadruple integral operator" such as "Vierfachintegral".

The MathJax project is currently building in a full accessibility feature, and since most screenreaders do not voice things like codepoints from mathematical blocks (which are mostly humanly unintelligible anyhow), the names generated by the current tool are being used in the speech text strings. Localization is a highly desirable goal.

If you know of a resource that already exists with such information, or is being developed, please share the information with Peter Krautzberger, <[peter.krautzberger@mathjax.org](mailto:peter.krautzberger@mathjax.org)> with a copy to me.

### What is a template?

A template (to me) is a receptacle just the right shape and size into which to pour contents, suitable for producing a desired result. A template for a L<sup>A</sup>T<sub>E</sub>X document is (or should be) an outline that a user can fill in for a particular purpose.

One area in which (L<sup>A</sup>)T<sub>E</sub>X templates are particularly useful is for academic theses. A student embarking on writing a thesis will need all the support s/he can get, since that exercise is the capstone of years of effort, and the public demonstration that the effort was worthwhile.

Many universities have very particular specifications for the format of a thesis. Templates for that purpose are in plentiful supply,<sup>1</sup> and steadily increasing.<sup>2</sup> Even more are posted somewhere on the web, many on CTAN, but many not. Another "productive" area for template creation is for CVs. Whole sites are devoted to collecting such templates.

Unfortunately, the quality of posted templates is uneven — many show evidence of accretion over "generations" of student thesis writers and lack of careful initial design or subsequent maintenance. Old (plain T<sub>E</sub>X) font commands, redundantly-loaded packages, and conflicting packages are only some of the unadvertised flaws. And there is no concerted effort to "clean them up" or even evaluate them: a poten-

<sup>1</sup> See Peter Flynn's survey in *TUGboat* 33:2, pages 172–177; [tug.org/TUGboat/tb33-2/tb104flynn.pdf](http://tug.org/TUGboat/tb33-2/tb104flynn.pdf).

<sup>2</sup> Two more were reported at TUG 2015; see *TUGboat* 35:2, pages 128–132; [tug.org/TUGboat/Contents/contents36-2.html](http://tug.org/TUGboat/Contents/contents36-2.html).

tial user has little guidance in selecting a reliable offering.

Publishers are not immune from this practice. Reports of problems with the templates provided for various journals or books demonstrate such problems every day. Although sometimes an author has a special requirement that wasn't predicted when a template (or a document class) was created, lack of maintenance is more frequently the culprit.

A discussion about this topic has been going on for months on the T<sub>E</sub>X stackexchange chat (<http://chat.stackexchange.com/rooms/41/tex-latex-and-friends>), and some ideas have been proposed to establish a working group to explore possibilities for improvement. If you are interested, send mail to [tugboat@tug.org](mailto:tugboat@tug.org) with your ideas; we will collect information and keep potential participants in such a discussion informed of plans.

### Still more help wanted: The L<sup>A</sup>T<sub>E</sub>X Wikibook

A comprehensive online L<sup>A</sup>T<sub>E</sub>X manual sounds like a great idea, for finding quick information on how to approach unfamiliar problems. And that's what the L<sup>A</sup>T<sub>E</sub>X Wikibook tries to be. It is apparently the resource that is most often high on the list from a Google search.

Unfortunately, this resource is far from a compendium of "best practices", so I hesitate to recommend it to new users — the individuals most in need of such help. Wikibooks are intended to be "crowd maintained", and anyone with knowledge and time is encouraged to step up and contribute.

A question on this topic has been posted to T<sub>E</sub>X stackexchange: "How can we, as a community, improve the LaTeX WikiBook?" <http://meta.tex.stackexchange.com/q/6393>. So far, no one contributing to the discussion has said that this Wikibook is fine as is. One contributor has gone so far as to say that a Wiki is unsuitable for beginners and intermediate users of L<sup>A</sup>T<sub>E</sub>X because it is "essentially a reference source", and that indeed a Wiki *format* is inherently unsuitable for a pedagogic purpose.

The fact remains, the L<sup>A</sup>T<sub>E</sub>X Wikibook is not going to go away. Like the template confusion, what is there now will simply lead to more and more questions, with many repeats. Join the discussion, and help to organize the available resources in a way that will both improve the quality of what is there, and make undertaking L<sup>A</sup>T<sub>E</sub>X less forbidding for newbies.

◊ Barbara Beeton  
[http://tug.org/TUGboat/tugboat\(at\)tug.org](http://tug.org/TUGboat/tugboat(at)tug.org)

## Adrian Frutiger, 1928–2015

Norbert Preining

*The textural quality of a typeface is like the timbre of a musical instrument, and the individual letters are like musical notes. A text composed in one typeface can look very different when composed in another because a complex visual sensation emerges from the repetition and interaction of nearly subliminal design features of the type. Frutiger's exploration of manifold type forms delights the eye with textural variations and deepens the understanding with abstract forms that express their distinctive meanings.* (Charles Bigelow [1])

On September 10, 2015, Adrian Frutiger passed away after a long and productive life. He has left us a rich heritage of fonts he designed, as well as important works on communication and visual language.

### Life

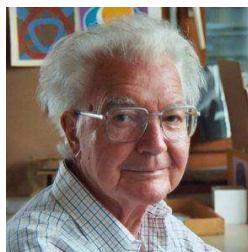
Born in Unterseen, a small village in the mountainous regions of the Berner Oberland, in 1928, Adrian Frutiger moved to Interlaken and later to Zürich to learn the craft of the typesetter. In 1949 he enrolled in the School of Applied Arts, from which he graduated in 1951 with distinction. His final project, a woodcut series about the development of Western type, inspired the interest of Charles Peignot of the Paris foundry *Deberny & Peignot*, where Frutiger worked until 1961, when he established a freelance studio. In 1972, his first design for Linotype was released, Iridium, followed by many others for Linotype over the next 35 years [7].

However successful his professional life, his private life was struck by a series of calamities: His first wife Paulette died after giving birth to their son, and both children with his second wife Simone experienced mental health problems and committed suicide, leading the Frutigers to establish the *Fondation Adrian et Simone Frutiger* to improve mental health support.

After having spent most of his life in France, particularly Paris, he returned to Switzerland and passed away in Bremgarten near Bern at age 87.

### Fonts

Adrian Frutiger designed about 40 type families, including masterpieces Frutiger and Univers, and notably also OCR-B for optical character recognition.



With Univers, he created one of the first super-families, one design with, initially, 21 sets (or styles). Originally targeting the Lumitype phototypesetting machine where manufacturing costs

were much lower per set, its enthusiastic acceptance led Peignot to publish Univers also in lead type.

Besides widespread adoption in advertising and industry, such as by Apple, Deutsche Bank, and General Electric, Univers was used for the 1972 Olympic Games in Munich, as well as at the Paris Orly Airport [4]. For Univers he also created a numerical coding system to overcome international discrepancies in naming. Despite the renown of this famous creation, which continues to thrive throughout all changes in the typographic world, Frutiger always remained modest about his achievements:

*I don't want to claim the glory. It was simply the time, the surroundings, the country, the invention, the postwar period and my studies during the war. Everything led towards it. It could not have happened any other way.* (Adrian Frutiger [2])

Based on an original design from 1970 for the Charles de Gaulle Airport, which was named after the suburb of the airport's location, Roissy, Frutiger released an improved design for print, giving it his own name, Frutiger. This became another internationally famous widespread typeface, due to its mixture of ideas from Univers with organic influences from Gill Sans, creating an extremely legible font of generally humanist design.

*Frutiger is basically the best signage type in the world because there's not too much 'noise' in it, so it doesn't call attention to itself. It makes itself invisible, but physically it's actually incredibly legible.* (Erik Spiekermann [6])

The design has seen many re-interpretations, including Frutiger Next and Frutiger Neue, as well as the serifed versions such as Frutiger Serif.

Other designs to be especially noted are Avenir, a more humanist version of the geometric sans-serif types faces of the early 20th century, and OCR-B, which in 1973 became the world standard for optical character recognition.

While Frutiger's most famous typefaces are sans-serif, he also designed excellent serif typefaces, such as Egyptienne, Centennial, Méridien (recently revised and re-released as Frutiger Serif), and Iridium.



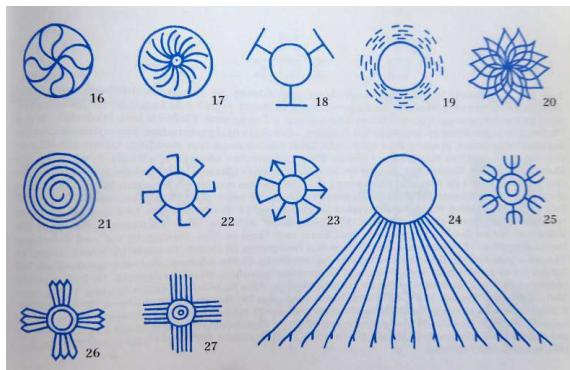
In all his designs, he focused on readability and fonts as tools.

*The whole point with type is for you not to be aware it is there. If you remember the shape of a spoon with which you just ate some soup, then the spoon had a poor shape. Spoons and letters are tools. The first we need to ingest bodily nourishment from a bowl, the latter we need to ingest mental nourishment from a piece of paper. (Adrian Frutiger [5])*

# Avenir Centennial Didot Egypt Frutiger Frutiger Serif OCR-B Univers

## Visual communication—signs and symbols

Besides being a prolific font designer, Frutiger was interested in the visual language of symbols, their development and interaction. His book *Der Mensch und seine Zeichen* (Signs and Symbols) [3] is a profound study on the development, history, and use of all kinds of symbols. In this book, translated into many languages, Frutiger explores the depth and breadth of symbols, but the most important aspect for him, easily seen from the German title, is the human (“Der Mensch”). Symbols are created, changed, and used by and for humans. His studies exhibit connections between various cultures when it comes to sign usage and design.



He also explores such signs as a modern language, and their importance for visual communication and identity building, along with the development of the Roman alphabet. He concludes this book with a very wise statement:

*Alphabetic signs alone have long been insufficient to record and convey human thoughts and statements. Orientation and communication would be impossible today without diagrams, signs and signals. Written or printed expression is necessarily complemented by pictorial communication. (Adrian Frutiger [3])*

With the loss of Adrian Frutiger, one of the great minds of typography is gone. He and Hermann Zapf [8] together formed the ground of font design and typography like no others in the twentieth century. Frutiger’s knowledge and insights into the craft will be guidance for many generations of typeface designers to come:

*On my career path I learned to understand that beauty and readability—and up to a certain point, banality—are close bedfellows: the best typeface is the one that impinges least on the reader’s consciousness, becoming the sole tool that communicates the meaning of the writer to the understanding of the reader. (Adrian Frutiger [7])*

## References

- [1] Charles Bigelow. Philosophies of Form in Serifed Typefaces of Adrian Frutiger. <http://goo.gl/1aekRD>.
- [2] Eye Magazine. Reputations: Adrian Frutiger. <http://goo.gl/BwjAow>.
- [3] Adrian Frutiger. *Signs and Symbols: Their Design and Meaning*. Watson-Guptill Publications, 1998.
- [4] Linotype. Nachruf auf Adrian Frutiger. <http://goo.gl/J4CQ3X>.
- [5] Linotype. Typography as the highest form of visual communication—a talk with Adrian Frutiger. <http://goo.gl/4AKvDv>.
- [6] New York Times. Adrian Frutiger dies at 87. <http://goo.gl/5IdcBp>.
- [7] Swiss Foundation Type and Typography, editors. *Adrian Frutiger Typefaces: The Complete Works*. Birkhäuser Verlag, 2008.
- [8] TUGboat, 36(2):92–99, 2015. Remembrances of Hermann Zapf. <http://tug.org/TUGboat/Contents/contents36-2.html>.

◊ Norbert Preining  
Japan Advanced Institute of  
Science and Technology  
Nomi, Ishikawa, Japan  
norbert (at) preining dot info  
<http://www.preining.info>

## Thomas Koch, 1964–2014

Joachim Schrod

A little more than a year has passed since Thomas Koch, former president of DANTE e.V. from 1998–2002, died suddenly and unexpectedly on July 24, 2014.

I don't remember exactly when we first met. I think it was at the Karlsruhe TUG conference in 1989, but it may also have happened at DANTE's autumn conference in Eichstätt, in the same year. What I do remember very vividly, as if it had happened yesterday, is this first evening we spent together.

Thomas was a very unusual person. He had already finished a full master's degree study in theology, went to the seminary to get his ordination to (Catholic) priesthood, and abandoned that goal to start studying computer science instead. He consistently evinced two main traits: he was interested in all kinds of people, and he was interested in talking about all kinds of topics, without limits. This first evening, we talked about books and typography—of course, it was a TeX conference—jumped from there to critical editions, meandered through medieval library attitudes, sprang to baroque music, compared that to contemporary modern music and free jazz, went on about electronics' usage in music and ended up discussing the future of IT data centers. There may have been even more topics, but these I still remember after 26 years.

You could throw any and all topics at him; he was always interested and willing to talk. Curiosity, like a child, always willing to be taken by surprise, was one of his main traits. Thomas was new to the TeX community at this time. Nevertheless, he immediately started to engage himself. Eventually, he became the DANTE coordinator for OS/2 in 1995—these were the times when TeX communities existed mostly to help folks get a TeX system up and running at all. In that capacity, he helped many people in the German TeX community and made himself a name as a sturdy guy whom you can rely upon.

His special background, having studied both a humanist and a scientific subject at a master's degree level, together with his broad interest in topics and people, made him well known and fully accepted by all parts of the German TeX community. When differences about the future course of DANTE led to internal frictions, he stood up, was voted in as



president in 1998, and took our organization's helm to steer us back to calmer waters. He succeeded in scaling back animosities, calmed down controversies, and brought quarreling factions back to the same table. Sometimes having been a priest-in-training seems to help, after all. In his presidential term, he was instrumental in guiding DANTE to focus on delivering better services to both its user group members and the larger TeX community as a whole. His term as president resulted in major investments by DANTE into the TeX Live CD, CTAN CD, ProText CD, and other developments that were not on the front burner before. Investments that are very visible until this day—I just received this year's TeX Collection disc from TUG this week.

His career at T-Systems, where he was responsible for the enterprise and cloud architecture of this major German IT outfit, didn't leave him much time, though, so he stepped down from his president's job in 2002 and passed that to Volker RW Schaa. Of course, he was still involved in the German TeX community, and sought after for his experience as former president—and besides, all his work also didn't prevent him being our best man at Christine's and my wedding in 2007. In 2014, we saw each other quite often; the last time, we discussed his plans for his 50th birthday, soon to come.

That phone call on a Friday, a year ago, when his partner Dorothea called us to say that he died unexpectedly and without any known cause, destroyed an important part of my life. It needed a year before I was able to write this commemoration... please bear with me if it got too personal. RIP, Thomas, we miss you. I miss you.

## DANTE e.V. 2015 meeting reports

Stefan Kottwitz

Editor's note: DANTE e.V. ([www.dante.de](http://www.dante.de)), the German-speaking TeX user group, holds regular meetings, both large and small. Stefan offered to report on two major ones this year, and we were pleased to accept, following his report on TUG'15. These were originally posted on [latex-community.org](http://latex-community.org) by the author; edited for *TUGboat*, with permission.

### DANTE spring meeting 2015

From April 16th to April 19th, 2015, the spring meeting of the DANTE e.V. took place at the University of Applied Sciences in Stralsund, Germany.

The director of the economic research institute opened the meeting with a few words, Herbert Voß then spoke for DANTE. We were sitting in a big lecture room, with WiFi and power sockets provided. There was another room for coffee breaks, where we could find delicious cake and nice talks at bar tables.

Dominik Wagenführ gave the first talk. He spoke about producing e-books in EPUB format from existing L<sup>A</sup>T<sub>E</sub>X documents. He showed several tools for conversion, such as `latex2rtf` and `tex4ht`. His conclusion was that existing conversion methods are not perfect, but it's doable with some manual effort.

Walter Entenmann followed with a presentation about TeX and Perl working together. Perl is a very capable scripting language with particular strengths in string processing. He demonstrated a workflow where Perl works on a data set and generated a TeX file, which is then processed by pdftEX to produce a PDF file.

Martin Schröder spoke about TeX in the third millennium. His presentation is nearly a tradition, as he has made similar talks during other meetings, and they develop as TeX and friends develop.

Dominik Wagenführ ended the first day's session with a talk on his own template for job applications.

The evening meeting was in the Spanish restaurant Bodega at the new market place. I came a bit late and it was harder to find a place than in the lecture room. But I was lucky and arrived at a nice table with interesting talks. I bet the other tables would tell the same.

Shortly before 9 pm we challenged the restaurant by all paying at the same time, and went on a tour at 9 with a "nightwatchman" through the old town.

At the second day, we started at 9:15 with the formal meeting. It was opened by Herbert Voß. He went first to association internals such as revenues and expenses, elections, and a change in the rules of the user group. Then it presented DANTE's

participation in events such as open source meetings and in project funding. When we talked about projects, I briefly mentioned the TeX projects I currently maintain, such as the TeX Internet forums [latex-community.org](http://latex-community.org), [texwelt.de](http://texwelt.de) and [golatex.de](http://golatex.de), and we talked about possible support by DANTE for the server operation.

Martin Kraetke of the company le-tex started the afternoon program. He presented the program `docx2tex`, software to convert Word documents to L<sup>A</sup>T<sub>E</sub>X. It is a command-line tool that generates XML as an intermediate format and at the end also outputs a L<sup>A</sup>T<sub>E</sub>X document. Using a sample document, he demonstrated the functionality and came to a pretty good result.

Joachim Schrod discussed the state of CTAN and explained the services it provides. He also gave this presentation at this year's TUG meeting, and I wrote about it in the proceedings issue of *TUGboat*.

After a coffee break, Till Tantau presented his graphics package `TikZ`. This is the front-end for the graphics language PGF, which he developed over more than a decade. PGF stands for Portable Graphics Format. Impressively, it works with all TeX engines (pdftEX, X<sub>G</sub>T<sub>E</sub>X, LuaTeX, ConTeXt, TeX in DVI mode), which allows for flexible usage. This is one reason for its success. The (also excellent) PSTRicks package, on the other hand, has had a harder time since PDF output has become dominant and working via PostScript has been, for some users, still a hurdle. The other advantage, in my opinion, is the comfort of the graphics description languages at the front end. For the development of `TikZ`, Till Tantau received this year's honorary prize of DANTE e.V., together with the co-developers Vedran Miletic, Mark Wibrow and Joseph Wright.

Returning to Till's lecture, at first he showed that even the huge `TikZ` package with (today) 4080 files and an 1165-page manual started small: it originally came with 22 files and 27 pages of documentation. This was version 0.62. He wrote it to use for ten images in his doctoral thesis.

Then he showed some special points of his package. First, he demonstrated his passion for detail with arrowheads. With `TikZ`, they are adjustable in many ways, and can even automatically bend when an edge is bent. Then, he demonstrated the automatic generation of graphs: one defines some nodes and certain edges relationships, plus certain desired characteristics, then PGF/TikZ constructs a tree or a graph. It does this meeting requirements such as avoiding overlaps, having the fewest intersections, maximum symmetry, minimal variations of the edge lengths from a preset length, and minimal variance of

the angles. The result should have a pleasing appearance to the eye. And that's usually what we want: graphics for best visual understanding by humans.

The resulting graph can be determined in even more detail: it can take format specifications, be power-based by edges which work like springs and react to pressure and pull, nodes having charges that can repel, having important nodes with gravity, or magnetism with alignment tendency at certain lines. At the end, we would release those nodes and edges, wait and see what we may get as an equilibrium state according to our definitions. Sounds complicated, but it is a smart thing: we start with certain node relationships plus some meaningful internal properties, and *TikZ* delivers to us a useful graph which matches our logical specification.

Here, we combine three languages: *L<sup>A</sup>T<sub>E</sub>X* for the document, *TikZ* for the graphics, and a DOT-like language for describing the graphics with an concise and powerful syntax. In addition, there's *Lua* for programming the underlying algorithms, as *T<sub>E</sub>X* does not suit the job here. That's worth knowing, because we need to compile such graphs with *Lua(L<sup>A</sup>)T<sub>E</sub>X*.

In the following discussion, Dominik wanted to know why the graph algorithms have been implemented in *Lua*, instead of using existing *GraphViz* libraries. They could be called externally. Till explained: the graph generation happens in the middle of the *T<sub>E</sub>X* run, with sizes and node contents developing at runtime. It is quite difficult to generate C++ class objects for such external libraries, to pass them and then to process the results. Therefore, a direct implementation is a natural decision, and it avoids dependencies. *LuaT<sub>E</sub>X* is sufficient and already comes with *T<sub>E</sub>X*. We don't have to get C++ libraries running on different systems.

One more interesting point: Till used a PDF shading function to generate a scalable Mandelbrot set image. It's unusual for PDF as a fairly rigid page description language to be able to calculate iteratively or recursively like PostScript. It's especially notable because by using a shading function he exploited a leak in PDF.

I was interested in what news we can expect in the near future for *TikZ*. Of course nobody can know for sure, as everything depends on time and interests, but Till Tantau showed clear interest in the use of SVG format as an additional output format. This format allows, for example, animations, and it is very portable. Modern web browsers can handle SVG.

The last presentation for the day was by Uwe Ziegenhagen. It was about the Org mode of emacs. This turns Emacs into a tool for outlining texts, for collecting notes, for creating todo lists and project

planning. It can output in various formats: *L<sup>A</sup>T<sub>E</sub>X*, ODT, HTML and DocBook, for example. Uwe explained the installation and demonstrated the usage with a sample document. Finally, he explained how you can configure the export, for example, which *L<sup>A</sup>T<sub>E</sub>X* packages should be used and which macros would be assigned.

After so much time in the lecture room, many of us chose not to take the bus but rather walked back from the university into the old town. That was a nice walk along the waterfront, taking almost an hour, with pleasantly sunny but cold weather. Again, it was a good opportunity to chat.

For the evening, there were tables booked in the Golden Lion restaurant in the Old Market. That was a short walk through the old town area, near our hotel. A soup was served at the table, then we could select from a very rich and very good buffet. We had good discussions, so the evening passed by quickly. After midnight, our remaining small group walked back to the hotel.

On the third day, Günther Partosch presented two talks. One was about making PDF documents following archival standards, and the additional work required. Such documents should survive changes in operating systems and technologies and should be reproducible in the same way on different systems. So, a basic requirement is embedding all fonts, images and color information in the file itself. He used the *hyperxmp* package for embedding metadata, and *hyperref* with the *pdfa* option to generate a (mostly) PDF/A compliant document. *glyptounicode.tex* was used to map glyphs to Unicode characters, as required by PDF/A. Compression needs to be switched off.

In the second talk, he demonstrated how to generate a glossary, a list of acronyms and a list of symbols, all using the *glossaries* package. He did it using real code examples.

Doris Behrend showed us examples of schoolwork and exams of the last fifty years. We could see how aesthetics and aspirations developed over time.

Then there was barbecue at the campus. One more tourist highlight followed: a visit of the Ozeaneum Stralsund, a huge aquarium and sea museum.

With the interesting talks, the many chats in the breaks, and the outside program, the meeting was a great experience. Many thanks to DANTE and the organizers, especially to Christina Möller, Silke Krumrey, and the university of Stralsund.

## DANTE autumn meeting 2015

DANTE's autumn meeting took place on Saturday, September 5th, 2015, at the Graz University of

Technology in Austria. Coming from Hamburg in northern Germany, I likely had the farthest journey, though it was pretty easy, as I could take a flight via Frankfurt to Graz. I must admit that it was not too difficult as I work for an airline.

At 9 am on Saturday morning, our president, Martin Sievers, started the meeting with introductory words. The local organizer Andreas Läßer gave the initial welcome, and provided information about things beside the official program. We had the usual functional meeting time with talk about ongoing projects and all the things DANTE is involved with. As often happens, we had a short discussion about the usefulness nowadays of the *TeX* Collection DVD. I understood that the consensus is to keep it. One factual reason is that the production is quite cheap. Furthermore, sending them to all members is easier than organizing who likes to get it and who does not. I suggested to add a simple suggestion to the cover letter: if you don't plan to use the DVD, give it as a present to friends or persons who would like to test *TeX*. So it's also good for spreading the word, which probably would not be the case if it came on a USB drive. Besides that, there's an enormous cost difference in several thousand factory-made DVDs compared to USB drives. Personally, I appreciate having the DVD in the hand. I remember times travelling around without a good Internet connection, desperately searching in newsstands for computer magazines which might have a *TeX* distribution included. I never found one at that time. There are many Linux versions on DVD in computer magazines, but I never saw a *TeX* DVD or CD with a magazine.

Herbert Voß made the first presentation. He showed how to proceed from the commonly used pdf*LATEX* to Xe*LATEX* and Lua*LATEX*. Besides showing the few necessary steps, he demonstrated how to use the system fonts in Linux and Windows with *LATEX*. A skeptical user asked, why change the engine, as there are already many high quality fonts available? Herbert's answer was clear: switching the engine and changing some lines in the preamble, which is well-documented, is much easier than integrating a new font into *LATEX*. If a font lacks direct *TeX* support, it can become difficult with pdf*LATEX*. And there are occasions when you don't have a choice, such as when a university or company requires the use of a particular corporate typeface.

Martin Sievers followed with a talk about making historical-critical editions with *LATEX*. He showed how to use the package *(r)eledmac* for this purpose.

Then we had lunch in a tavern nearby. The lunch and soft drinks were without charge; thanks

to DANTE, the budget allowed lunch and dinner free of charge for meeting attendees.

After lunch, I gave a short talk about discussion and support for *LATEX* on Internet forums. We took a tour through some forums which I maintain, such as [latex-community.org](http://www.latex-community.org), [TeXwelt.de](http://TeXwelt.de), [goLaTeX.de](http://goLaTeX.de), and [TeXnique.fr](http://TeXnique.fr), the last of which was recently started with French *TeX* friends. I demonstrated our fully automated method of adding *LATEX* examples with output image and thumbnails, including compiling and Ghostscript conversions, to *TeX* galleries, such as provided on [TeXample.net](http://TeXample.net) and [LaTeX-Cookbook.net](http://LaTeX-Cookbook.net). Following this, we had a talk about functionality and possible improvements and future plans. People were interested, as DANTE supports the server hardware which runs the forums. We spoke about data dumps of publicly accessible data, NNTP interfaces for the forum software. Furthermore, there were some interesting thoughts which could be programmed, such as a central dashboard serving several forums and sites with aggregated RSS feeds and a central panel for interested users, consolidated cross-site search, and further small improvements for convenience.

Herbert Voß then made a second presentation, this time about automated document generation. He discussed a shell script which collects weather data over time from a web site, inserts it into a *LATEX* document, processes it with pdf*LATEX* and copies it to a server for worldwide access.

The final talk was made by Doris Behrend. She provided an experience report about using *LATEX* in a seminar at a secondary school. There, writing with *LATEX* was compulsory. She talked about the challenges and results with *LATEX* beginners at this level. There were interesting and good results shown. Of course, we cannot expect a high typographic aspiration of *LATEX* beginners. I guess, for those people who mastered that seminar, using *LATEX* at a university would be natural from the start.

In the evening, we met for dinner in the Glöcklbräu tavern. Besides excellent spare ribs, they served an excellent local beer.

On Sunday, there was a bus excursion in the southern Styrian wine country. I did not attend since I had to return home, but I heard only good things.

Thanks to DANTE and to the sponsoring institutes of the TU Graz, especially to Andreas Läßer, for organizing this great meeting.

◊ Stefan Kottwitz  
 stefan (at) texblog dot net  
<http://www.latex-community.org>

---

## TeX in schools: Just Say Yes!

Simon Michael Laube

### Abstract

This article not only describes why using L<sup>A</sup>T<sub>E</sub>X as an application program for typesetting in schools is a good idea, but also lists several benefits.

### 1 Introduction

The modern student has to write reports and articles often—especially if one studies any kind of engineering. When it comes to scientific publishing, (L<sup>A</sup>)T<sub>E</sub>X is definitely the right choice for typesetting, so introducing L<sup>A</sup>T<sub>E</sub>X into schools could obviously be useful. However, due to the commonly used word processors young people do not even know about real typesetting systems, and that is a shame. This paper is intended to point out the benefits of using L<sup>A</sup>T<sub>E</sub>X as an application program for students. Further, it is more or less an answer to the article “TeX in Schools: Just Say No”, written by Konrad Neuwirth in 1990 [1], who claimed that TeX should not be used *as a programming language* in schools.

As I am still a student in Austria, the expressed opinions are related to the Austrian school system, but could be adjusted to other countries too. Moreover, this paper is meant to be valid for upper-secondary schools or higher educational institutions.

### 2 The benefits of L<sup>A</sup>T<sub>E</sub>X as an application program

Konrad Neuwirth’s article was written at a time when computers were still rare in Austrian schools. Since that time, much has changed—students are using computers every day, either at school or at home. Engineering schools have also started to teach programming languages like C or Java to every student, as they are an important skill in the modern world. Konrad Neuwirth considered using TeX as a programming language in schools, which is not the aim of this paper—although students are more confident with programming nowadays, so it would not be a big problem.

Nevertheless, L<sup>A</sup>T<sub>E</sub>X should be used by students as an application program only and should definitely not be more than that. Thus, it is important to guide beginners through the first steps, because they have to be aware that writing a document in L<sup>A</sup>T<sub>E</sub>X is not a fancy point-and-click adventure (the mouse is not used that often) they are used to. Once they have learned the basics, students are prepared to use L<sup>A</sup>T<sub>E</sub>X for their daily work.

There are many benefits of using L<sup>A</sup>T<sub>E</sub>X in schools, but most of them are also valid and essential for L<sup>A</sup>T<sub>E</sub>X itself. One of the most important points—especially for young people—is that L<sup>A</sup>T<sub>E</sub>X forces its users to *structure a document*. Students often find it very hard to structure things like their daily timetable, the priorities of tasks and—of course—documents. Normal word processors do not require a strict structure within a document while L<sup>A</sup>T<sub>E</sub>X obviously does. Structuring can help people to save considerable time, either at work, school or in their free time and therefore it is an important skill.

Second, students will learn to *concentrate on the content* of a document, not on its layout when using L<sup>A</sup>T<sub>E</sub>X. I personally do not completely agree on this point—although I am emphasizing it—as many people often find themselves searching for the best design for their L<sup>A</sup>T<sub>E</sub>X document, but for schools a completely different approach has to be taken. Teachers often use their own requirements for the student’s documents, such as homework and project reports. How about coding these requirements in L<sup>A</sup>T<sub>E</sub>X and passing the macros to the students instead of telling them how to format their document? When doing so, the students will definitely produce better content as they are able to concentrate on their topic, while the formatting stays the same for every document.

Another important fact is that L<sup>A</sup>T<sub>E</sub>X makes *group projects more productive* due to two features:

- file format
- subfiles

L<sup>A</sup>T<sub>E</sub>X’s text-based file format is not an advantage an average student would primarily think of, but it could save plenty of time when different people are working on the report of a group project or something similar. Text-based means that the file is pure, human-readable text with a specific encoding, nothing more. This feature ensures compatibility across systems and engine versions and could be even more important in larger groups. The commonly-used word processors do not always ensure this compatibility as they often have commercial and open-source versions as well as different versions of one product. Most of the word processors are somehow text-based too, as they use a zipped file tree with XML files to save a document, but this structure does not ensure compatibility in any way.

A second advantage of L<sup>A</sup>T<sub>E</sub>X for group projects is the use of several text parts located in subfiles, which can easily be included into a main document. Thus, each group member can work on one or more

text files, which are then included into the group's master file— $\text{\LaTeX}$  does the rest and keeps references correct.

### 2.1 $\text{\LaTeX}$ in non-engineering schools

All the points mentioned above have one thing in common: they are all about structure. As previously mentioned, students often find it hard to structure something. Things get even harder when people are not confronted with structuring every day, because their profession or education does not require it. Students of engineering schools have a bit of an advantage here, because their strict mathematical and logical education forces structuring more than the education of students at non-engineering schools. Both of them are experts in their profession, but cannot get used to  $\text{\LaTeX}$  in the same way.

Nevertheless,  $\text{\LaTeX}$  could also help people at non-engineering schools that do not need formulas and equations. It could help them producing *more readable pieces of text*, which is very important—especially when texts get very long. They are able to learn which fonts to choose to ensure the maximum reading convenience. Long and theory-laden texts could be reproduced using  $\text{\LaTeX}$  to make learning them more efficient and less tedious.

### 2.2 $\text{\LaTeX}$ for teachers

$\text{\LaTeX}$  could not only be a good tool for the students at a school, but also for the teachers. One main advantage of  $\text{\LaTeX}$  for teachers is the abovementioned possibility of creating and distributing *formatting templates (classes and packages)* which the students can then use to create documents with the desired look and appearance. In a scaled-down approach, teachers do not always need to write whole packages to get what they want. Simple macros could also be written and distributed to fulfill special tasks,

for example special-looking lists or anything similar. The key point of that all is again, compatibility. Formatting templates in  $\text{\LaTeX}$  are compatible with different systems and installations, whereas word processors often have their problems with that.

Another notable advantage of  $\text{\LaTeX}$ —when it is used by teachers—is that they can use the program to typeset their scripts in a better way. For example large scripts can be maintained more efficiently if a version control system is used. Further, the scripts can be typeset in an easily readable form, as mentioned in the previous section, to make them better understandable for the students. Some teachers only use their scripts in a digital form. In this case even animations within the script are possible and useful as they show technical drawings and processes in more detail.

## 3 Conclusion

All in all  $\text{\LaTeX}$  does a great job in schools. I have experienced the great possibilities of  $\text{\LaTeX}$  at my own school and I would definitely recommend the program to every student. Sure, at the beginning it is hard to get used to  $\text{\LaTeX}$ , but it is worth the effort as it greatly improves the working process of writing a document and therefore:

Just say yes!

## References

- [1] Konrad Neuwirth.  $\text{\TeX}$  in schools: Just say no. *TUGboat*, 12(1):171–174, March 1990. <http://tug.org/TUGboat/tb12-1/tb31kneuwirth.pdf>.

◊ Simon Michael Laube  
Wieselburg, Austria  
`simon dot laube (at) gmx dot at`

## Smoky letters

Linus Romer

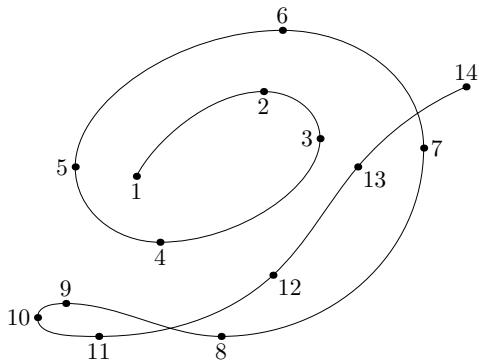
### Abstract

The wish for an individually designed thank-you card led to the idea of developing a “smoky” capital *D* that would look a bit different with each compilation but still elegant, such that it could be combined with a copperplate font to form the word *Danke* (German for *thanks*). This project was substantially facilitated by METAPOST and its random number generator.



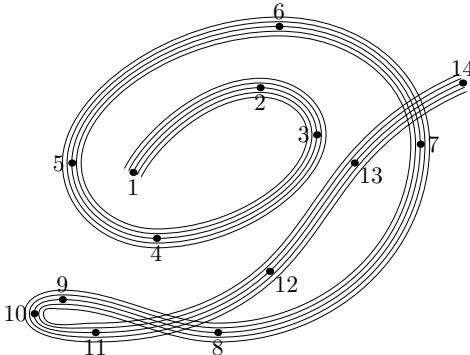
### From a single path to a bunch of paths

For a quick start, I read the coordinates of the most important points (mainly extrema) of the letter *D* in the *Calligra* font using the FontForge editor. These points were connected in a path like this:

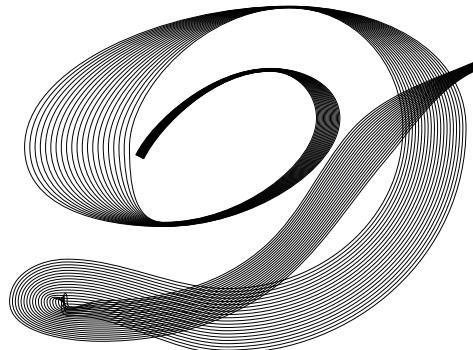


```
beginfig(0);
u:=1mm; % standard unit
pickup pencircle scaled .2pt; % pen size
z1=(21u,34u); z2=(48u,52u); z3=(60u,42u);
z4=(26u,20u); z5=(8u,36u); z6=(52u,65u);
z7=(82u,40u); z8=(39u,0); z9=(6u,7u);
z10=(0,4u); z11=(13u,0); z12=(50u,13u);
z13=(68u,36u); z14=(91u,53u);
draw z1..tension 1.5 and 1..z2{right}..
z3{down}..tension 1.5 and 1..
z4{left}..z5{up}..tension 1.4 and 1..
z6{right}..z7{down}..z8{left}..
z9{left}...z10{down}...z11{right}..
z12..z13..z14;
endfig;
```

Then, a replacement for the `draw` macro was needed, namely `smokydraw`. This new macro shifts the main points of the path and constructs a new path with the same directions and tensions as the original path:



In the final version of `smokydraw` the stem widths are additionally changed by using random numbers (`normaldeviate`).



```
def smokydraw expr p =
  save widths,k,n,smokypath;
  numeric widths,k,n;
  k=length p;
  n=10; % number of curves on each side
  for j=0 upto k: % set random widths
    widths[j]=5u*abs(normaldeviate);
  endfor
  path smokypath;
  for s=1,-1: % both sides of p
    for i=1 upto n: % curve index
      smokypath:=
        for j=0 upto k-1: % point index
          (point j of p shifted
           (dir(angle(direction j of p)+90)
            *i/n*widths[j]*s)){direction j of p}
          ..tension posttension j of p
          and pretension j+1 of p..
        endfor
        (point k of p shifted
         (dir(angle(direction k of p)+90)
          *i/n*widths[k]*s)){direction k of p};
      draw smokypath;
    endfor
  endfor
  draw p % original path (in the middle)
enddef;
```

`smokydraw` needs a predefined standard unit `u` and the macros `posttension` and `pretension` as described in *The METAFONTbook*. Enjoy!

◇ Linus Romer

Oberseeestrasse 7, Schmerikon, 8716, Switzerland  
linus.romer (at) gmx dot ch

## About the DK versions of Lucida

Charles Bigelow

Donald Knuth's informative and amusing letter, "A footnote about 'Oh, oh, zero!' [4], tells of his efforts in the late 1960s to reduce confusion between capital letter 'O' and zero in the typography of computing journals and books, when he recommended to the ACM (Association for Computing Machinery) and his publisher Addison-Wesley a new, squarish shape of capital 'O' to distinguish it from oval zero in fonts that simulated typewriter (fixed-width) text. After much correspondence, ACM did not follow his suggestion but Addison-Wesley did, commissioning a special squarish 'O' for the first volume of *The Art of Computer Programming*.

A dozen years later, Don designed a squarish capital 'O' for Computer Modern Typewriter, which he made with his newly developed Metafont system. His beloved squarish 'O' is now standard in his own books and in publications of others who use his Computer Modern typefaces.

He doesn't explicitly state in his "Footnote" (maybe it seemed too obvious to mention) that in effect he was proposing a new parameter of typographic distinction: "squarishness" versus "roundishness" of curves (my words) within a typeface.

### Structure of a typeface design

The graphical structure of a typeface design is built from a complex set of distinctive features, some of which apply within a face, and others of which apply between faces. Within a face, several features like round, straight, and diagonal, ascender or descender, dot or no dot distinguish individual letters from each other. Groups of letters are distinguished by a few features, including height, width, pointyness (which distinguishes punctuation (hence the name) from letters proper) and weirdness (for lack of a better word) for characters like @ # \$ % & that look somehow weirder than letters, are usually logographs symbolizing a word rather than a single sound, and for which people can't seem to agree on names or purported historical derivations. For instance, commercial "at" @ (plausibly derived from a contracted ligature of Latin "ad" meaning "at, to, toward" but poorly attested before the 16th century) is called "at" by most English speakers, but its name varies wonderfully from language to language, with meanings of ear, strudel, elephant trunk, monkey tail, and so on; number sign # (probably an abbreviation of Latin "numeris" meaning "number") is variously called "hash", "octothorpe", "pound", "sharp" and "number" in English. I expect that readers will enjoy noting

other names and suggesting alternative derivations.

When we read, we recognize all these kinds of distinctions almost unconsciously because they are part of our passive "vocabulary" of typographical signifiers. They help us understand the structuring of text. In type design, these distinctions are elements of a designer's active graphical vocabulary, the conceptual toolkit used to shape the look of information.

Graphical distinctions within and between capitals and lower-case, roman and italic typefaces have been devised, refined, and standardized since the early 15th century, when humanist scholar and scribe Poggio Bracciolini developed the humanist handwriting that eventually became canonical roman type. Thanks to Poggio, nearly all Latin text typefaces contain two alphabets: upper-case (capitals) and lower-case (small letters), and thanks to Poggio's friend, Niccolò Niccoli, who wrote a cursive variant of Poggio's hand, roman has been distinguished from italic.

Over time, the capitals gained semantic import; marking a significant distinction. A "Bill" is a person, but a "bill" is an invoice, a bird beak, or a cap brim. I can attest that something similar applies to "Chuck" and "chuck". The distinction between upper- and lower-case is therefore graphemic; the graphical difference signifies a difference in meaning. Interestingly, the patterns of capital usage differ between orthographies. In German orthography, initial capitals mark nouns and are correspondingly frequent, whereas in modern French, capitals are rather rare except at the beginnings of sentences and personal names. In English orthography, capital usage was fairly frequent in earlier centuries but has diminished since the 19th century.

**Size.** Capital letters are comparatively bigger, as denoted by their Latin-derived name "majuscule" (biggish) and lower-case letters smaller, in Latin "minuscule" (smallish). The Latin majuscule/minuscule distinction is rare among typographic writing systems, occurring first in Latin type, and later in Greek, Cyrillic, and Armenian types influenced by the Latin model but deriving their capital versus lower-case forms from different roots and by different means. The "case" distinction is not found in most other writing systems, including Hebrew, Arabic, Chinese, Korean, Japanese, or the several related systems used in India, such as Devanagari for Hindi and Sanskrit.

**Width.** The width of characters is another distinction. In proportionally spaced typefaces, capital 'O' is wider than zero, which is usually adequate for differentiation, but the width distinction is almost neutralized in monospaced (fixed-width) fonts, and that is where confusion between 'O' and zero most commonly occurs.

Marks. The dot over the minuscule letter ‘i’ appeared in the late middle ages as a light stroke over the letter ‘i’ in blackletter “textura” script, to mark the letter as separate from other so-called “minim” letters. The dot, along with later accents and diacritics, like those proposed in the 16th century by French typographer Geoffrey Tory, has been continued in most Euro-Latin typography, and there are occasional extensions when adapting Latin letters to other languages or clarifying distinctions, as in fonts for linguistics. Hence it seemed natural to add marks to distinguish capital letter ‘O’ from zero when confusion between the two cropped up in the early days of computing.

In my article “Oh, oh, zero!” [2], which Don Knuth “Footnoted”, I gave examples of various proposals from computer journals and typography journals for distinguishing ‘O’ and zero. These included the addition (or rarely, subtraction) of slashes, dots, loops, bars, and other twiddles to ‘O’ or to zero. Proposals from scientists, technologists, engineers, or mathematicians usually added distinguishing marks to the ‘O’, keeping their zero pristine in its symbolic mathematical emptiness. Proposals from humanists, artists, scholars and designers added distinguishing marks to the zero, keeping their capital letter ‘O’ by historical precedent and charismatic authority, citing the classical Roman inscriptions, or the famous story of the early Renaissance painter Giotto, as told by Giorgio Vasari. When an emissary from the Pope asked Giotto for a drawing to show his skill:

Giotto . . . took a sheet of paper, and with a brush dipped in red, fixing his arm firmly against his side to make a compass of it, with a turn of his hand he made an ‘O’ [tondo] so perfect in curve and contour that it was a marvel to see it.

Giorgio Vasari, *Lives of the most excellent painters, sculptors, and architects*

(Here I translate Vasari’s Italian word “tondo” as ‘O’ in English, but it can also be translated with justification as ‘circle’. Modern Italians do both when referring to Vasari’s story, sometimes saying ‘cerchio’ (circle) or ‘O’ instead of “tondo”, which incidentally also means “roman” when referring to a typeface.)

The skill of drawing a marvelous ‘O’ was not lost with Giotto. The late Fr. Edward Catich, who revived the painting and carving of Imperial Roman Inscriptional capitals, could paint a wonderfully round capital ‘O’ with two strokes of the brush, in keeping with his analysis of how the Trajan capitals were painted and carved back in 113 A.D. When Catich would demonstrate this to the admiring gasps of onlookers, he would grin and say, “Perfetto come la ‘O’ di Giotto.” Kris Holmes, who studied with

Catich briefly and with Catich’s pupil, Fr. Bob Palladino, can recount his other feats of brush-writing mastery, including writing a perfect Roman capital ‘R’ behind his back and upside down.

Contrast. A distinction called “contrast”, or modulation of thick and thin strokes, was used in “old-style” types from the 15th to the late 18th century, and in modern revivals, in which the figure zero was cut approximately at lower-case height but as an annulus, an unmodulated ring without thick-thin variation, distinguishing it from the lower-case ‘o’, which retained the varying line thickness of the humanist pen written letter. Contrast is also a major feature of the typeface genre called “modern”.

In his 1960s suggestion to Addison-Wesley, Don proposed a new way to distinguish the curve of the ‘O’ from that of the zero: “squarishness” versus “roundishness” (my words). Although, as he relates, Addison-Wesley followed his suggestion, it wasn’t a generally practical or enduring solution. In Metafont some ten years later, Don implemented his solution as a parameter he called “superness” (from Piet Hein’s term “superellipse”) [5]. Superness “controls the amount by which the curve differs from a true ellipse.” Thus, Don gave squarishness versus roundishness in typeface design a precise algebraic expression: the exponent of the general equation describing the ellipse.

The classic ellipse of the conic curves analyzed by Greek mathematicians is algebraically described by an equation of degree 2. In the early 19th century, the French mathematician Gabriel Lamé generalized the ellipse by varying the exponent. In particular, an exponent above 2.0 makes the corners of the ellipse smoothly bulge or inflate beyond the classical elliptical oval. In 1959, Danish polymath Piet Hein empirically determined that an ellipse of degree 2.5 was, for him, the most agreeable compromise between the ellipse and the rectangle, and he used the shape to design a traffic oval. Hein called the ellipse of degree 2.5 a “superellipse”. I believe that, to date, Metafont is the only type design tool that provides control over this parameter.

In “Oh, oh, zero!” [2], I also showed samples of recent digital fonts that distinguish zero and ‘O’ by various means, but I neglected to show a sample of Computer Modern Typewriter (shown here in fig. 1), although Karl Berry used CMTT in his “Production Notes” on the article [1], which shows instances of the squarish ‘O’. Knuth refers to this in his “Footnote”.

In this note, I wish to correct my oversight and also to relate my subsequent efforts to respond to Don’s appeal for a squarish ‘O’ for Lucida Console, a typeface that Kris Holmes and I designed in 1993, based on our Lucida Sans Typewriter of 1986.

### Squarish ‘O’s in Lucida

At the end of his *TUGboat* “Footnote”, Don wrote:

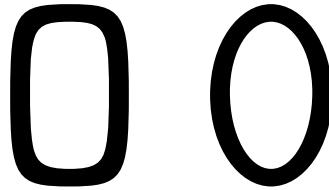
Alas, however, Chuck’s essay demonstrates that I’m still standing alone in this respect: None of the nine monospaced typefaces in his Fig. 9 have anything like an Oh that I would want to use. (Nowhere did I see a really satisfactory Oh in Chuck’s discussion until I came to Karl Berry’s production notes at the end, and Karl’s reference to *ZeroFontOT.otf*.) I herewith submit a humble request to have squarish O and Q available as alternates in the next edition of Lucida Console.

I could not ignore Don’s appeal, nor the challenge inherent in it, so I set about crafting a squarish ‘O’ for Lucida Console, but I began with Lucida Grande Mono, the most general case of three nearly identical designs: Lucida Sans Typewriter, which was designed to look like Lucida Sans but monospaced, Lucida Console, a version of Lucida Sans Typewriter with shortened capitals for the console/terminal window(s) of an operating system, and Lucida Grande Mono, which has the taller capitals of the original design, the larger character set of Lucida Console, and various small adjustments based on three decades of experience with the original design.

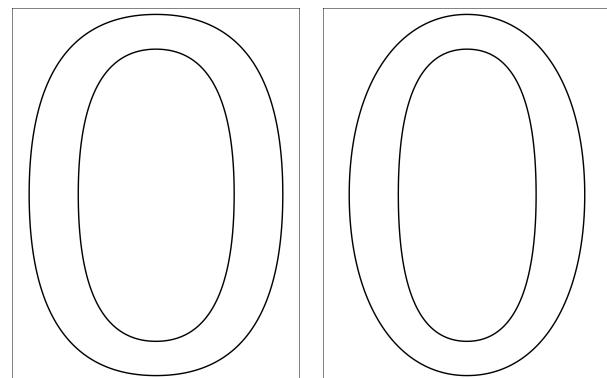
Of course, the task of making a simple squarish ‘O’ turned out to take much longer than I estimated. I figured it would take a few weeks, with interruptions and hiatuses, but it took several months. The practical difference between estimate and reality conformed to a Knuthian heuristic for estimating the length of time a project will take: make your best estimate in some time unit, add one, and jump to the next higher level of time measure.

Now that the fonts are at last done, Karl has asked me to write some words about the design process. I wish I could describe it as a clear, precise, and logical process but I must confess that, to borrow and invert a phrase used by Leslie Valiant for evolutionary “ecorithms” (contrasted with “algorithms”), I used a “probably approximately incorrect” method.

I started with the capital ‘O’ character from Lucida Grande Mono as the most general case, as described above. This is the same glyph as the original ‘O’ from Lucida Sans Typewriter (fig. 2 shows the outline of the ‘O’ and zero glyphs). It had been hand-drawn in 1986, digitized with Peter Karow’s Ikarus software as contour points on cubic Hermite splines. Those were converted to conic splines at the Imagen Corporation, using a contour representation developed by Vaughan Pratt at Stanford, and first converted to bitmap fonts and released in 1986 as Lucida Sans Typewriter. The Ikarus splines were later con-



**Figure 1:** Computer Modern Typewriter (10pt design size, enlarged): capital Oh (left) and zero (right).

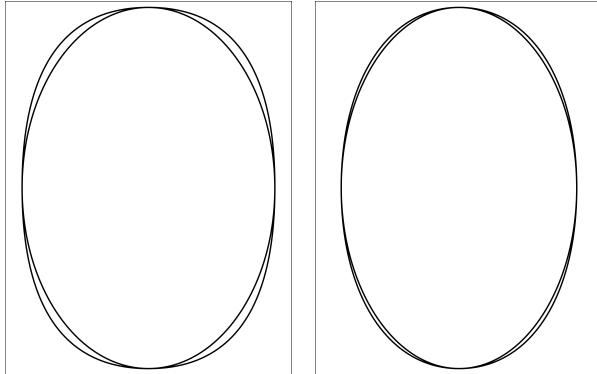


**Figure 2:** Hand-drawn capital Oh (left) and zero (right) in Lucida Grande Mono (originally Lucida Sans Typewriter).

verted to Sun Microsystems’ F3 general conic format developed by Jacobo Valdes and Eduardo Martinez (also based on Pratt’s conics). In 1990, we converted the Ikarus font to TrueType format for Microsoft, and some years later, to Bézier format for Adobe.

The original Lucida Sans Typewriter capital ‘O’ is elliptical-ish, somewhat more squarish than a true ellipse. The shape was drawn visually; its slightly greater width and squarishness helped distinguish it from the zero and gave it a little more interior area within the Procrustean confinement of the fixed-width typewriter cell. The zero is closer to a true ellipse (fig. 3).

At the time of the original design, we debated whether to put a slash in the zero to make a clear distinction between zero and ‘O’. We eventually did not add the slash because of various conflicts, the foremost of which was possible confusion with the Norwegian ‘Ø’ letter (which may have inspired the null set symbol with slash, according to mathematician André Weil). Also, some computer scientists and engineers—supposedly the main users of such a font—disdained an altered zero. Moreover, one of our goals for Lucida Sans Typewriter was to closely resemble proportionally spaced Lucida Sans, which of course did not have slashed zero. And, traditional “typewriter” fonts also did not have slashed zero.



**Figure 3:** Outer contours of above hand-drawn capital Oh (left) and zero (right), compared to true ellipses (inner contours) with the same major and minor axes.

To gain a sense of what Don wanted in a squarish ‘O’ shape, I examined the ‘O’ in his Computer Modern Typewriter. My task would have been much simpler if I had just imported his ‘O’s Bézier contours from a PostScript font (originally made by collective effort of Y&Y, Blue Sky Research, and Projective Solutions), scaled it to the proportions of Lucida Grande Mono, plunked it (along with accented variants) into the proper Unicode slots, and been done with it. But, some four decades ago, when Kris Holmes and I began working on type designs together, she said it is a mistake to look at other typefaces when designing a new one, because inevitably you can’t get the other designs out of your head, and that prevents you from creating something new from your own concepts. Also, it’s more interesting, and you learn more by starting afresh than by copying.

So, having looked at the squarish ‘O’ in Computer Modern Typefaces, I closed the book and put it back on the shelf. I then proceeded to craft a shape that looked squarish to my eye by starting from scratch. Designing a squarish ‘O’ for Lucida wasn’t simply a problem of making a squarish ‘O’ but of making a squarish ‘O’ that visually harmonized with everything else in the typeface, although there wouldn’t be any other letters like it except ‘Q’, or, in an extended character set, Greek Omicron and Theta, Cyrillic ‘O’, and the various accented ‘O’s.

In 1992 with Y&Y, developers of a PC-based version of *T<sub>E</sub>X*, we developed Lucida fonts in PostScript Type 1 format for use with *T<sub>E</sub>X*. For character contours, we tried to minimize the number of Bézier points, to keep font file sizes small, and the conversion from Ikarus format to PostScript Bézier put on-curve spline points at the *x*- and *y*-extremes of curves, in accordance with how we hand-digitized the

outlines. Having spline points at extremes facilitated the later hinting of the PostScript outlines by Y&Y. The capital ‘O’, then, comprised four Bézier curve segments with four on-curve spline knots where segments joined, and eight associated off-curve control points, or handles.

As using Bézier curve-based software for designing fonts directly on computers became available, another reason to minimize curve segments became evident—it was difficult for designers to smooth contours composed of many Bézier segments with spline knots and handles that had to be controlled by the unintuitive process of moving curve “handles” (control points) around in the plane and sometimes adding on-curve points. Cubic Bézier curves are tremendously flexible; in addition to seemingly well-behaved curves that look like (but are not exactly) classical conic sections, Bézier curves can make a vast number of loops, cusps, inflections, and other surprising shapes that are never needed in type design but which can confound the visual artist trying to control them. So, to get a handle on the curves (sorry), designers tend to be minimalists, using as few Bézier curves as possible if the curves must be manipulated by hand and eye.

A problem with the minimalist approach is that Bézier curves cannot exactly match true circles and ellipses. Arbitrarily closer approximations can be achieved by splitting Bézier arcs into shorter segments, but more segments involve more curve joins and handles, which thus increase the probability of making awkward joins, cusps, bulges, or dips, thereby complicating the task of the designer working with the curves. The minimization of Bézier segments may be a reason that many of the typefaces developed directly on computer screens seem to me to have subliminal similarity of curves. When I look at the shapes of many recent fonts, I have a sense of *déjà vu* all over again, as the late philosopher Yogi Berra is said to have said.

At any rate, because we had already reduced the contours of Lucida Sans Typewriter to a near-minimal number of Bézier curves when working with Y&Y, I could start with that version and manipulate the Bézier handles until the ‘O’ shape looked visibly squarish. This went through several iterations because I had to test the new ‘O’ shape in sample settings with the rest of the typeface. When the ‘O’ looked outstandingly squarish, it didn’t jibe with the rest of the characters in the font, but when it was rather roundish, it didn’t quite look different enough from the oval zero. As usual in type design, it wasn’t enough to achieve a “just noticeable difference” between the two characters. What was needed was a

“definitely noticeable difference”.

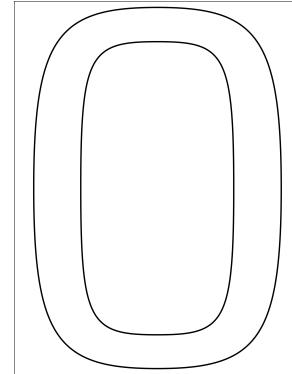
I then experimented with making the interior counter somewhat more squarish than the exterior contour. This gave the shape an illusory shoulder-padded look reminiscent of TV soap opera actress costumes of the 1980s.

When I had an ‘O’ that seemed adequately squarish, that wasn’t the end, because getting the isolated shape to look right was not enough; it also had to play well with others. Because the squarish ‘O’ was no longer in keeping with the ovality of most of the other letters, I had to re-fit its sidebearings—the spaces on each side. In a fixed-width font, side spacing usually involves compromises because letters like capital ‘I’ have lots of air around them, while ‘M’ and ‘W’ are packed in tight. The oval bowls of the original ‘O’ had been adjusted to seem roughly equal in spacing next to letters with straight sides like ‘H’ or with round sides, like ‘C’ or ‘D’.

Compared to the oval bowls of the original ‘O’, the sides of the squarish ‘O’ were almost upright, just slightly bowed, so they seemed closer to other characters, and hence their side spacing had to be increased to restore approximately equal letter spacing, but in the fixed-width cell, the letter also had to be narrowed to make room for the increased spacing. The squarish form also necessitated adjusting the weight of the character, because the straighter sides carried more weight, since they didn’t thin down to the thinner horizontal arches. To make the squarish ‘O’ seem the same weight as the original ‘O’ and to match the visual gray tone as the other capitals, I had to re-weight it, slightly shaving down the sides and arches. In a fixed-width font with a fairly strong stem weight like Lucida, this is another challenge.

If a fixed-width font is light in overall weight, it is easier to adjust visual spacing and weight, but Lucida Sans Typewriter (and hence Lucida Grande Mono and Lucida Console) has a relatively sturdy weight that makes it work well on back-lighted screens (which tend to erode the visual weight of a font), and also to work well in programming environments that use color to denote aspects of code. But, darker weight is harder to equalize because there is less white space available.

To get quantitative correlation of visual intuition, I rasterized the original ‘O’ and used photo software to count the percentage of black pixels within the cell, and did the same with the new squarish ‘O’. I then adjusted the outlines of the squarish ‘O’ sides and arches until the percentage of black pixels to total pixels was approximately the same as that of the original ‘O’, while keeping the visual look of the verticals in keeping with other capitals.



**Figure 4:** Hand-made (with Bézier splines) squarish Oh from Lucida Grande Mono (abandoned).

All that having been done, I made sixteen variants of the new, squarish ‘O’ and its siblings for the WGL character set of Lucida Grande Mono (which includes Unicode blocks Basic Latin, Latin-1, Latin Extended-A, part of Latin Extended-B, basic Greek, and basic Cyrillic). The new characters included ‘O’ versions with diacritics, OE digraphs, ‘Q’, Greek Omicron and Omicron-tonos, and Cyrillic ‘O’.

I didn’t make the Greek capital Theta squarish because, although traditionally similar in shape to Omicron, which is the ancestor of Latin ‘O’, Theta isn’t directly related to ‘O’, is a consonant not a vowel, isn’t confusable with an unadorned zero, and Greek mathematicians like Apollonius of Perga developed mathematics for traditional conics, not super-ellipses. Figuring I needed cultural advice on this, I asked the opinion of a Greek mathematician who uses Lucida math fonts, Antonis Tsolomitis, and he said that he didn’t think the Theta needed to match the squarish ‘O’.

After all those new ‘O’s and related characters were installed in the regular weight font, I made bold versions following the same process. The bolder weight made harmonization and fitting of the squarish ‘O’ with the rest of the capitals an even more elaborate process of refinement, because as weight increases in a fixed-width cell, the character has to be narrowed more in order to have adequate side spacing. Next, I obliqued both regular and bold versions to the same angle (11.3 degrees) and added them to the italic styles.

After the four Lucida Grande Mono faces were done, I then scaled down the squarish ‘O’s in the *y* dimension to make all those characters all over again for the four Console versions. All in all, the two font families, Lucida Grande Mono and Lucida Console, had 128 new squarish ‘O’-like characters (fig. 4).

(By this time, I wished Don had “thinked different” (to convert an advertising phrase from Apple from imperative to past tense) and opposed the common mathematician-scientist preference for modifying the ‘O’, and had instead requested a modified zero. My task would have required only 8 new characters instead of 128, saving four binary orders of magnitude. And, if other scientists later asked for variant ‘O’s instead of zero, I could have resisted by citing Knuth’s great authority.)

After testing the final fonts for any last-minute problems, I generated OpenType fonts and sent them to Karl Berry at TUG, who enlisted Michael Sharpe to add OpenType tables to allow switching between the default Lucida Grande Mono and Lucida Console versions, with slashed zero and oval ‘O’, and the new “DK” version with open zero and squarish ‘O’.

I wish I could say that was the end of the process. But, it was not. It was only the end of the first, and ultimately discarded, phase.

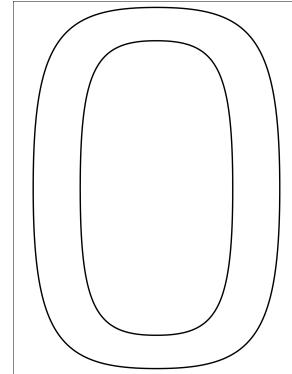
### Superelliptical-pi ‘O’s in Lucida Grande Mono

After Karl and Michael received the fonts and made a test OpenType version, I began to have second thoughts. I should say, “2nd order” thoughts.

I began to think that making a squarish ‘O’ by visual intuition wasn’t the optimal way to make a squarish ‘O’, at least not for Don Knuth. I figured I needed a more mathematical approach, although I am not a mathematician.

Hence, I investigated the superellipse numerically as well as visually. In Lamé’s algebraic generalization of the ellipse, the exponent of Don’s Computer Modern Typewriter squarish ‘O’ is 4, which Don presumably found clearly distinguishable from more or less elliptical zero, but by visual inspection, it was too squarish to harmonize with the fitting, weight, and look of the rest of Lucida Grande Mono.

Piet Hein’s superellipse has an exponent of 2.5, instead of the classical 2.0, and has been popular. Hein empirically settled on 2.5 as the exponent that gave the most satisfying compromise between the rectangle and the ellipse. That may be, but he was assessing the aesthetics of the shape in isolation, for instance, in his famous traffic round-about. From a long-ago infatuation with the superellipse, I have a small metal “super egg”, a superellipsoid of revolution of a superellipse. It has the charming ability to stand stably on one end, unlike a hen’s egg (barring Columbus’s demonstration). Also, I have a porcelain superellipse dish, suitable for baking a pie. So, I was personally acquainted with the superellipse in household objects.



**Figure 5:** Lucida Grande Mono DK Oh superelliptical-pi, outer and inner contours.

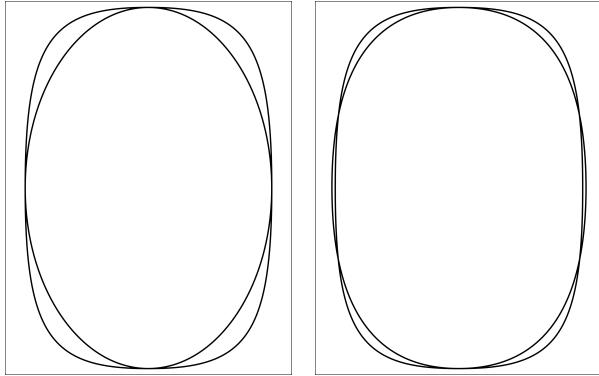
For a typeface design, the squarish ‘O’ can’t be a mere compromise between ellipse and rectangle. It has to distinguish ‘O’ from zero unambiguously. Not by a mere “just-noticeable” difference, nor a “just-preferable” difference, but an “obviously visible” difference.

So, I figured I needed to find a superellipse with an exponent somewhere between Hein’s 2.5 (not squarish enough) and Knuth’s 4 (too squarish for Lucida). So next I tried the so-called “natural superellipse” with an exponent of the natural logarithm base  $e$  (approximately 2.718). It was more squarish but still pleasing, and mathematically,  $e$  is transcendental and irrational, which, based on such cool names, I thought should be important qualities, although I can’t explain exactly why. Alas,  $e$  was still not squarish enough to be unambiguous.

Nevertheless, I didn’t give up, taking heart from Mark Twain’s observation in *A Tramp Abroad*: “A round man cannot be expected to fit in a square hole right away. He must have time to modify his shape.”

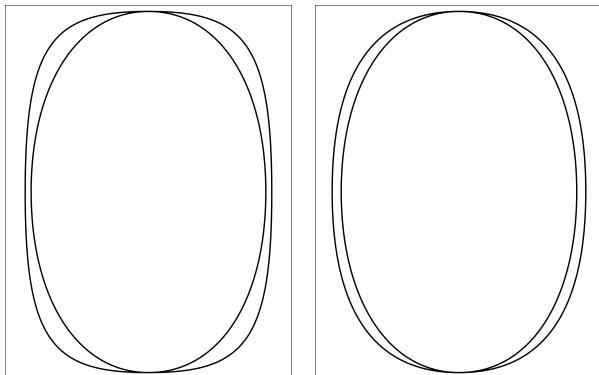
So I tried degree 3.0. Better, but still not quite squarish enough. So then I tried an exponent of 3.5, but that made the superelliptical ‘O’ look slightly too squarish, because the near-vertical sides and near-horizontal top and bottom didn’t rasterize at low resolutions with enough curvature to please me. At low-res, nearly but not quite straight curves often rasterize with infelicitous bit patterns. But, I was converging on something between 3.0 and 3.5. Perhaps 3.25 should have been next, but having tried  $e$ , I thought another transcendental irrational number might offer mathematical elegance, even if I couldn’t explain it. So I chose  $\pi$  (see fig. 5), the best-known transcendental number, which had been discovered by the Greeks in ancient times, and which is intimately linked to the circle.

With an OpenType font em only 1000 units in



**Figure 6:** Left: Lucida Grande Mono DK Oh superelliptical-pi outer contour, compared to true ellipse with same major and minor axes.

Right: Lucida Grande Mono DK superelliptical-pi capital Oh outer contour, with standard Lucida Grande Mono hand-drawn Oh outer contour intersecting. The superelliptical-pi contour is narrower, for more harmonious letter spacing and weighting.



**Figure 7:** Left: outer contour of superelliptical-pi capital Oh from Lucida Grande Mono DK (outer), with outer contour of zero (inner) for comparison. Right: hand-drawn capital Oh contour with zero inside. Zero is narrower and has lesser shoulders.

height, in which the actual ‘O’ height was 759 units, and a fixed width cell only 603 units in width per character, in which the ‘O’ was only 519 units wide, a long decimal expansion of  $\pi$  was not needed, so I settled for an approximate exponent of 3.142. And even that was more precise than I was able to achieve in practice.

I used a superellipse calculator to plot a bitmap approximation, brought that into a font development tool as a background image, and hand-fit Bézier splines along the curves. As I worked on adjusting the shape, I found that, with the minimum number of four Bézier segments, I could not exactly model the plotted superellipse of exponent  $\pi$ . I wondered if

Bézier approximations to superellipses had the same sort of slight error as approximations to classic circles and ellipses, so I asked Berthold Horn, with whom we had worked so well on the original PostScript Type 1 versions of Lucida. He affirmed the error problem and sent me helpful plots showing the small differences. He suggested dividing the Bézier curves into smaller segments for greater accuracy, but, as I explain above, I resisted that suggestion because of the user-interface problem of messing around with the off-curve handles on on-curve spline joins of multiple curves. This is why I describe my process as “probably approximately incorrect” (fig. 6, left).

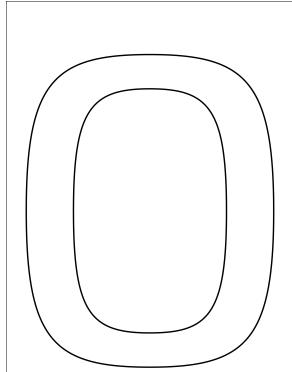
Nevertheless, considering the usual resolutions at which the letter would be rendered in text sizes on computer screens and printout, and the limitations of human visual acuity, the approximation seemed essentially as good as precision. For Lucida Grande Mono in 10 point text at 300 dots per inch on a laser printer, the ‘O’ height would be roughly 30 bilevel pixels tall, around 34 grayscale pixels on an iPhone Retina screen, around 42 grayscale pixels on the iPhone 6s Plus, and around 60 pixels on a 600 dpi printer. At those sizes and resolutions, tiny errors in approximating  $\pi$  seem negligible.

Figure 6 (right) compares the outer contour of the superelliptical-pi capital Oh for the DK font to the outer contour of the standard Lucida Grande Mono Oh. Figure 7 compares, on the left, the outer contours of the superelliptical-pi Oh with the zero from the same font, and, on the right, the hand-drawn original Oh with the zero.

#### Superelliptical-pi ‘O’s in Lucida Console

The superelliptical ‘O’ may prove to be only a quaint curiosity in Lucida Grande Mono, but in Lucida Console, it resolves an important problem.

Lucida Console has shortened capitals to adapt it to the graphical shortcomings of a terminal window in Windows NT. It has functioned well for that purpose for more than 20 years, and has a distinctive look that people have adopted for general purposes, not just terminal windows. However, current programming styles like CamelCase and PascalCase (also called BumpyCaps, mixedCase, etc.) use compounded words or phrases in which the separate parts are signified by capitals. When the capital letters are shortened, as in Lucida Console, some pairs are harder to distinguish, especially capital ‘O’ and lower-case ‘o’. The  $\pi$  superellipse in the DK version of Lucida Console (fig. 8) solves the particular problem of distinguishing capital ‘O’ from zero, taller and more oval than the shortened ‘O’, and also distinguishes capital ‘O’ from lower-case ‘o’, which



**Figure 8:** Lucida Console DK Oh, superelliptical-pi contours (derived from the Grande Mono DK Oh).

is shorter and more circular.

After choosing  $\pi$  for the new exponent of the superelliptical ‘O’, I threw away the previous “shoulder pad” version and rebuilt all 128 characters but also added Theta, feeling that I should not deprive our Greek friends of the innovation (after all, they invented the forms of our capital alphabet), so there were 144 new characters to make. A gross of ‘O’s!

In summary, this is an extension to monospaced font families, beyond Computer Modern Typewriter, of Don Knuth’s innovation of superellipticity as a distinctive feature within a typeface design. A new way to solve the venerable and perennial ‘O’ versus zero problem.

I feel it would have been more accurate to have rendered the letters with the greater precision of Metafont than with the font tools I used, which involved hand-fitting Bézier splines. Contemporary WYSIWYG font design tools offer more user-friendly interfaces (although far from truly congenial) but less precision than Metafont, although there is no real need for such a trade-off. If desire for superelliptical figures gained currency, we might hope that in the future, font tool developers would integrate a superellipse function into their software, such as FontLab, Glyphs or Robofont, some of which already offer an ellipse-drawing function. Knuth’s source code is publicly available, and published in *Metafont: The Program*. Probably, most font designers will not need a precise superellipse to solve the special problem of ‘O’ versus zero, but adjustable superellipticity could be useful in creating typefaces within the genre of superelliptical styles, as described further below.

Knuth’s innovative parameterization of elliptical versus superelliptical within a typeface to distinguish ‘O’ and zero is not, however, the first instance of squarish shapes in text typefaces. Type designers have shown a feeling for such forms, even without the aid of algebra.

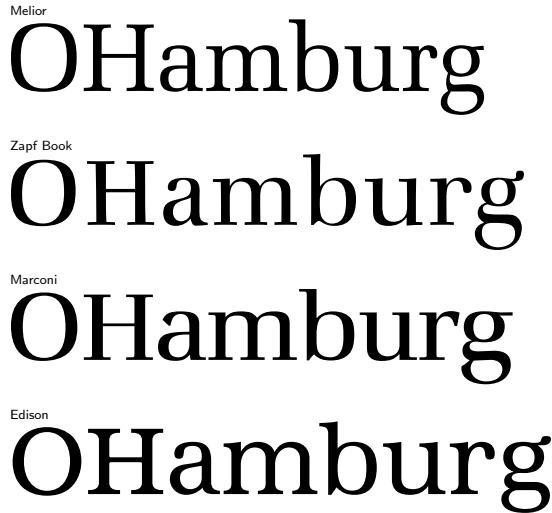
### Historical designs with squarish shapes

The late Hermann Zapf designed Melior, released in 1952 as a news text face, with superelliptical forms. Zapf employed several techniques to craft Melior [6]—his elegant calligraphic shaping of letters with a slight superelliptical trait (harder to achieve by hand than in traditional calligraphy), his preternatural skill in drawing letters precisely at small sizes, and his ingenuity in combining traits of transitional and modern designs. Although initially intended for newspaper text, Melior achieved wider usage in magazines as well as in advertising typography and display, where its distinctive look was at once both modern and classical, appealing to typographers in search of a new look.

Aldo Novarese and Alessandro Butti’s Microgramma titling face of 1952 (some sources say 1951) also has a distinctive “squarish” look, although some letter shapes appear to owe more to the modernist concept that informed Marcel Breuer’s bent tubing furniture than to the superellipse per se. As a display face, Microgramma lacked lower-case, but Novarese’s Eurostile of 1962 extended the design concept to include lower-case, and provided a lower-case for Microgramma in its later releases. Microgramma and Eurostile were, and still are, popular in titling and logos supposed to evoke the future. This has often puzzled me because in nearly every movie I have ever seen about the future, nobody is reading, whether classical or superelliptical fonts. Everyone is too busy blasting with ray-guns or vaulting into hyper-space to relax and read a good book.

Of all designers, Zapf has explored superelliptical forms most extensively (fig. 9). After Melior, he created Hunt Roman, a private press face for the Hunt Botanical Library at Carnegie Mellon University. Developed in association with Jack Stauffacher, then the book designer for the library, Hunt shows subtle traces of Melior-like superelliptical forms. As a private press face, it was produced only in metal. Comenius is a later, commercial phototype relative of Hunt, which also shows hints of superellipticity. Two of Zapf’s type families for ITC are further explorations of the concept. Zapf Book has formal, Walbaum-like “Modern” high-contrast serifed forms, while Zapf International gives a flowing, informal, hand-lettered look to active but slightly superelliptical shapes.

Zapf’s experiments with superelliptical designs continued with three of the earliest original digital typefaces that he designed for Dr.-Ing. Rudolf Hell’s “Digiset” digital typesetters in the 1970s and early 1980s. Marconi, intended for newspaper headlines



**Figure 9:** Some of Hermann Zapf’s “squarish” designs, all typeset at design size 36 pt.

and subheads, is a severely formal display family that combines Bodoni-like contrast, traditionally popular in newspaper headings, with superelliptical shapes. Its text companion, Edison [3], designed to save on newsprint costs while maintaining open contours that won’t clog or fill with news ink on high-speed presses, has a stunningly big x-height for a serifed design, nearly 53% of the body. At 8 point, Edison looks as big as Times Roman at 12 point. Although nearly 40 years old, it looks surprisingly new even today.

Zapf’s last typeface for Hell was Aurelia, released in 1983. Aurelia is a fascinating application of subtle superellipticity to the Venetian Humanist genre, essentially the first successful typographic roman type. Thus, Zapf combined some of the newest concepts in digital type design with some of the oldest in metal type. Inspired by the admirable humanist typeface in books printed by Nicolas Jenson in Venice in the 1470s, Zapf rendered Jenson’s definitive roman with a distinctively calligraphic touch first seen in Palatino, and with a hint of the superellipse first seen in Melior.

### Availability

The Lucida DK fonts are included in the complete Lucida OpenType font set, available through the T<sub>E</sub>X Users Group: <http://tug.org/lucida>. They are also available on their own to TUG members. If there is demand, they could be made available through B&H’s web site as well: <http://lucidafonts.com> (which provides many Lucida variants of all kinds not available elsewhere).

To conclude, here is one last example, showing the four variants each of the original Lucida Sans

Typewriter with the new Lucida Grande Mono DK and Lucida Console DK, all typeset uniformly at a nominal size of (approximately) 8 pt.

```

ABOQ xyz 012 LucidaSansTypewriterOT
ABOQ xyz 012 LucidaSansTypewriterOT-Oblique
ABOQ xyz 012 LucidaSansTypewriterOT-Bold
ABOQ xyz 012 LucidaSansTypewriterOT-BoldOblique
ABOQ xyz 012 LucidaGrandeMonoDK
ABOQ xyz 012 LucidaGrandeMonoDK-Italic
ABOQ xyz 012 LucidaGrandeMonoDK-Bold
ABOQ xyz 012 LucidaGrandeMonoDK-BoldItalic
ABOQ xyz 012 LucidaConsoleDK
ABOQ xyz 012 LucidaConsoleDK-Italic
ABOQ xyz 012 LucidaConsoleDK-Bold
ABOQ xyz 012 LucidaConsoleDK-BoldItalic

```

### References

- [1] Berry, Karl. Production notes. *TUGboat* 34:2, pp. 181–181, 2013. <http://tug.org/TUGboat/tb34-2/tb107prod.pdf>.
- [2] Bigelow, Charles. Oh, oh, zero! *TUGboat* 34:2, pp. 168–181, 2013. <http://tug.org/TUGboat/tb34-2/tb107bigelow-zero.pdf>.
- [3] FontShop. Edison (LT). <https://www.fontshop.com/families/edison-lt#info>.
- [4] Knuth, Donald E. A footnote about ‘Oh, oh, zero!’, *TUGboat* 35:3, pp. 232–234, 2014. <http://tug.org/TUGboat/tb34-2/tb111knut-zero.pdf>.
- [5] Knuth, Donald E. *The METAFONTbook*, volume C of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986. (Notes on superellipticity on pp. 126ff.)
- [6] Zapf, Hermann. *Alphabet Stories: A Chronicle of Technical Developments*. RIT Cary Graphic Arts Press, Rochester, 2007. ISBN 978-1-933360-29-4. (Notes on superellipticity on pp. 30–31, 114–115.) <http://ritpress.rit.edu/publications/books/alphabet-stories.html>.

◆ Charles Bigelow  
<http://lucidafonts.com>



Superelliptical Apple pie, baked and photographed by Kris Holmes (co-designer of Lucida). Superellipse baking dish and super egg designed by Piet Hein. (In addition to apples, the pie filling contains a bit of quince and a hint of poncirus.)

## History of cookbooks

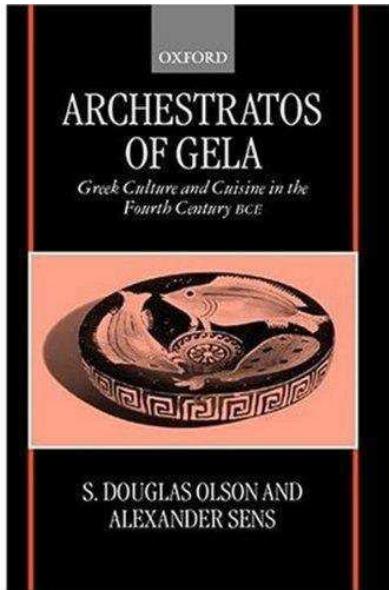
Taco Hoekwater

### Introduction

Cookbooks as we know them, with detailed instructions, ingredient list, and illustrations, are a fairly modern invention. This paper presents some famous cookbooks from history, starting in ancient Greece and ending with Internet-based modern approaches.

### The Life of Luxury

Archestratus was a Greek writer and traveller who lived in the 4th century BCE. Coming from Sicily (then a Greek colony) he travelled throughout the Mediterranean. He wrote a poem called ‘Hedypatheia’ (meaning ‘Pleasant Living’ or ‘Life of Luxury’). The original of the poem is lost, but luckily parts of it were quoted in another ancient work, and so some 60 verses are still known.



Here is a small sample:

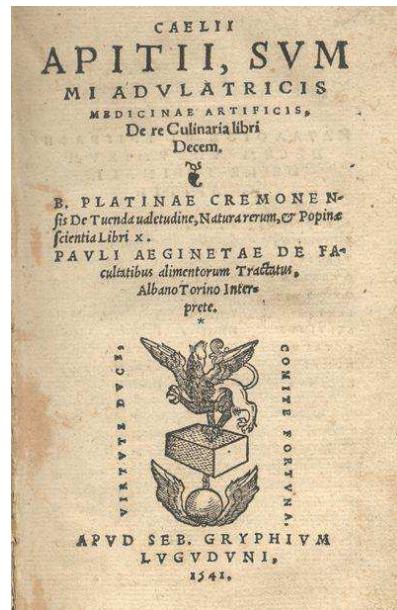
But I say to hell with saperde, a Pontic dish,  
And those who praise it. For few people  
Know which food is wretched and which is  
excellent.

But get a mackerel on the third day, before it  
goes into salt water  
Within a transport jar as a piece of recently  
cured, half-salted fish.  
And if you come to the holy city of famous  
Byzantium,  
I urge you again to eat a steak of peak-season  
tuna; for it is very good and soft.

Archestratus, fragment 39  
Olson and Sens translation

### Apicius, a.k.a. ‘De re coquinaria’

Whereas ‘Life of Luxury’ seems to have been more of a travel guide, the Roman recipe collection known as ‘Apicius’ was intended to be used while cooking.



Compiled around the start of the 5th century, it is a collection of ten books on various topics related to food and cooking. It has actual recipes, although not quite the way we are used to them. An example:

ANOTHER LAMB STEW — put kid or lamb in the stew pot with chopped onion and coriander. crush pepper, lovage, cumin, and cook with broth oil and wine. put in a dish and tie with roux.

Apicius, translation from Project Gutenberg

This recipe is concise almost to the point of uselessness, but that is a common problem with historical cookbooks: for most of their history, cookbooks were written by professional cooks *for* professional cooks (working for royalty and popes). Helpful information for amateur cooks like cooking times and ingredient amounts is omitted.

### Book of Dishes — al-Warrāq

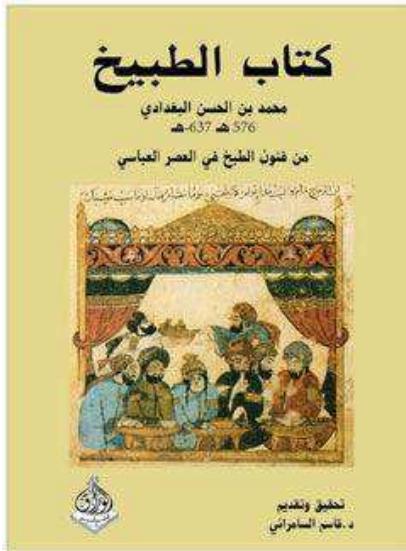
After the disintegration of the Roman empire, European interest in cookbooks became nearly nonexistent for centuries. As the cookbooks of the time were very much a ‘haute cuisine’ affair, a certain level of cultural prosperity was a prerequisite to new books being written. This was the case in the Arabic world, and two famous books come from that background.

First there is Kitab al-Tabikh (‘The Book of Dishes’), composed in the 10th century by Ibn Sayyār al-Warrāq. Some of the recipes in this collection are

as terse as the earlier example, but most are fairly elaborate, e.g., mentioning amounts for ingredient spices. In 2007, Brill published an English translation by Nawal Nasrallah under the title ‘Annals of the Caliphs’ Kitchens—Ibn Sayyār al-Warrāq’s Tenth-Century Baghdadi Cookbook’.

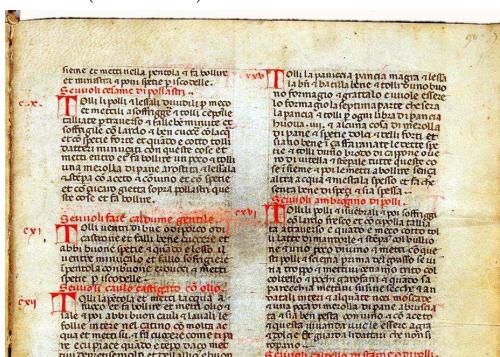
### Book of Dishes — al-Baghdadi

The second Arabic ‘Book of Dishes’ was compiled by Muḥammad bin Ḥasan al-Baghdadi, in 1226. Besides that information and the cover image below, I could not find much information about this book. I suspect that is mostly because the 2007 edition of the *other* Book of Dishes pops up in every Internet search using English language text ... and I do not understand enough Arabic to get around that problem.



### Liber de Coquina

At the end of the ‘middle ages’, European culture once again reached a high enough level that there was interest in cookbooks. One of the first was a collection from the early 14th century named ‘Liber de Coquina’. It has two parts: ‘Tractatus’ (part 1) and ‘Liber de Coquina’ (part 2). Both parts are written in (medieval) latin.



It is interesting that while the text in the Arabic collections was often quite detailed, this is not the case in the European manuscripts. While much attention was given to the look of the manuscripts, the recipes themselves are very terse.

### Le Viandier

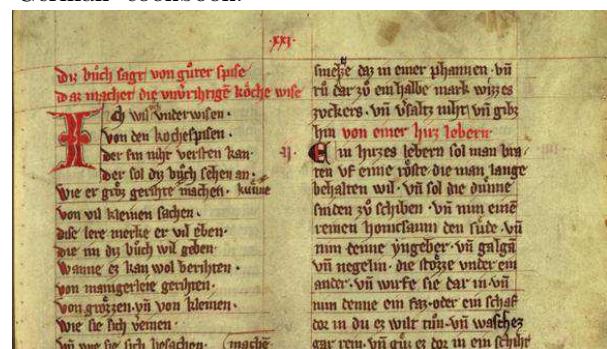
This book is generally considered to be the start of ‘French cuisine’. It was compiled in the early 14th century by a French author with chef Guillaume Tirel. Note the use of ‘compiled’ in the previous sentence: plagiarism was quite normal in these times. In fact the first known (but incomplete) manuscript containing this collection is older than Tirel.

## Le Viandier



### Das Buoch von guoter Spise

Much like ‘Le Viandier’ is considered the first ‘French’ cookbook, ‘Das Buoch von guoter Spise’ is the first ‘German’ cookbook.



A sample recipe (for apple sauce):

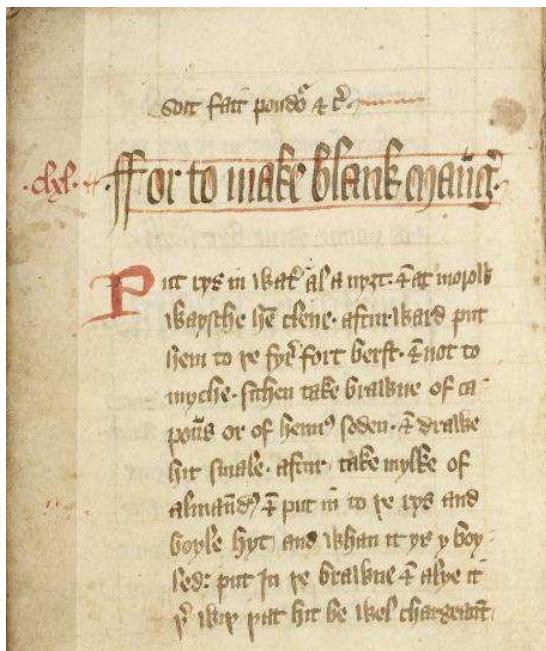
#### 69. Ein apfelmus

Wilt du machen ein apfelmus. so nim  
schöne epfele und schele sie. und snide sie in

ein kalt wazzer. und süde sie in einem haufen.  
und menge sie mit wine und mit smalze und  
ze slahe eyer mit wiz und mit al. und tu daz  
dor zu. und daz ist gar ein gut fülle. und  
versaltz niht.

## Forme of Cury

With French and German progenitors, there should be an English one as well! The ‘Forme of Cury’ in a compilation by ‘the chief Master Cooks of King Richard II’. The text is in Middle English and it dates to the end of the 14th century.

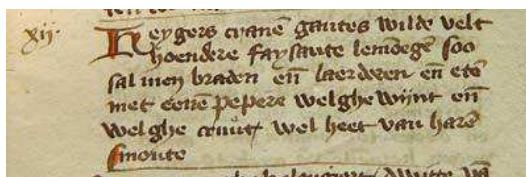


## For to Make Blank Maunger

Put rys in water al a nyzt and at morowe  
waisshe hem clene, afterward put  
hem to þe fyre fort berst & not to  
myche. ssithen take brawn of Ca  
pouns, or of hennes. soden & drawe  
it smale. after take mylke of  
Almandes. and put in to be Ryys &  
boile it. and whan it is yboi  
led put in þe brawn & alye it  
þerwith. bat it be wel chargeaunt

## Wel ende edelijke spijse

This is a Dutch cookbook from the second half of the 15th century.



Reygers cranen gantes wilde velt  
hoendere faysante lemmoegen soo  
sal men braden ende laerderen ende eten  
met eenen pepere wel ghewijnt ende  
wel ghecruut wel heet van haren  
smoute

English translation:

Herons, cranes, geese, wild partridges, pheasants, pheasants (a variety)

Roast them and lard them, and eat them with a pepper [sauce] with enough wine and spices, very hot from their fat.

De honesta voluptate et valetudine

This Italian book ('On honourable pleasure and health') from 1474 has the honour of being the first 'printed' cookbook. The publisher/composer is Bartolomeo Sacchi (a.k.a. Platina), but it is mostly based on earlier work by Maestro Martino of Como. It became widely popular and had a large influence on the Italian cooking tradition.



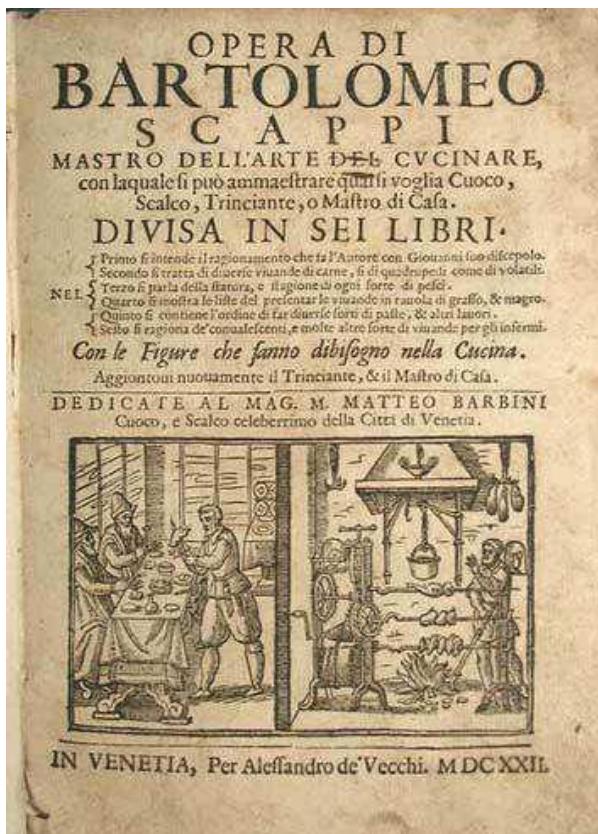
## Een notabel boecxken van cokervyen

A book called ‘A notable little cookery book’ was the first printed Dutch cookbook (not a very important book in the great scheme of things, but hey, I am Dutch). Printed in 1514 in Brussels by Thomas vander Noot, who may or may not also be the author.



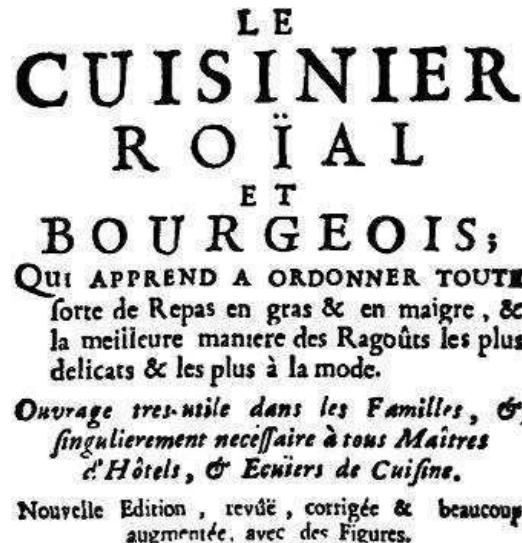
### Opera dell'arte del cucinare

Bartolomeo Scappi was the Italian Renaissance chef for popes Pius IV and V. The 'Works of Art of Cooking' is a masterpiece of six books containing more than a thousand recipes as well as explanations of techniques and giving helpful hints about all aspects of cooking. A notable and popular feature of his books were the beautiful illustrations.



### Le Cuisinier roïal et bourgeois

After the advent of printing (and generally, the end of the Middle Ages), there was a growing market for cookbooks. An important example from this period is 'The royal and bourgeois cook' by François Massialot. Published in 1691, it was the first cookbook to contain an alphabetic recipe list. Until this, recipes were typically only grouped in categories, without any means of quickly finding a particular recipe.

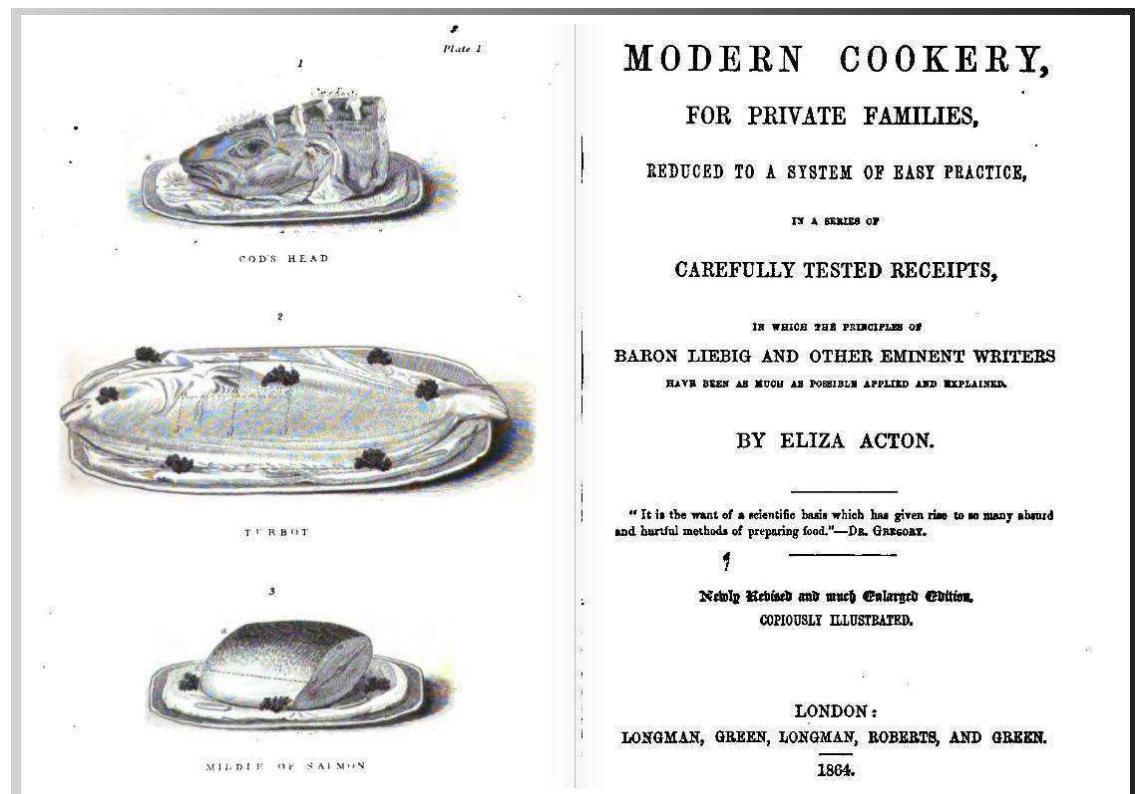
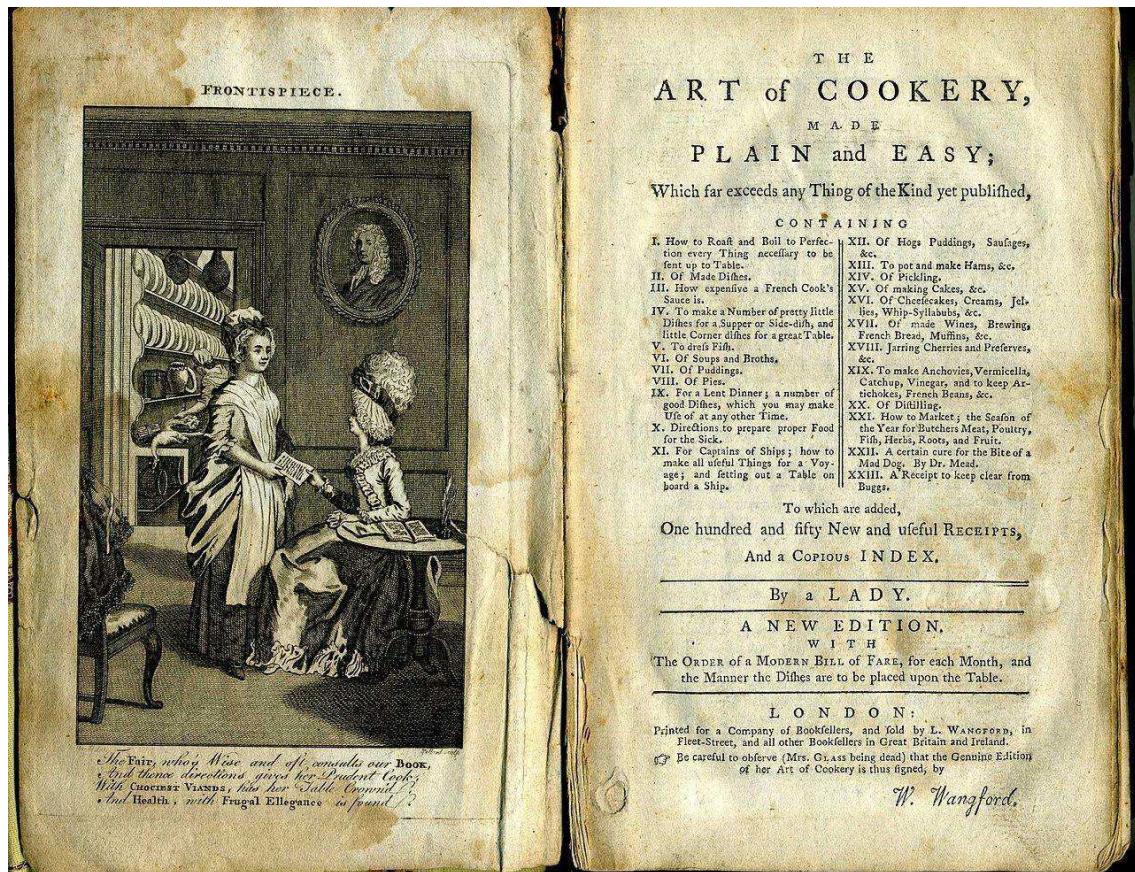


**A PARIS,**  
Chez CLAUDE PRUDHOMME, au Palais, au sixième  
Pilier de la Grand'Sale, vis-à-vis la Montée de la  
Cour des Aides, à la Bonne-Foi couronnée.

M. DCCV.  
**AVEC PRIVILEGE DU ROT.**

### The Art of Cookery Made Plain and Easy

All the cookbooks listed so far were aimed at professional chefs. One of the most famous English cookbooks from the 18th century changed that. Hannah Glasse wrote recipes specifically for the servant cooks of her well-to-do buyers (the servants themselves could probably not afford her book). First published in 1747, 'The Art of Cookery Made Plain and Easy' was a big success. Besides the simple language, she also worked to be practical (read: economical) in the choice of ingredients. All in all, the book became very popular in the North American colonies.



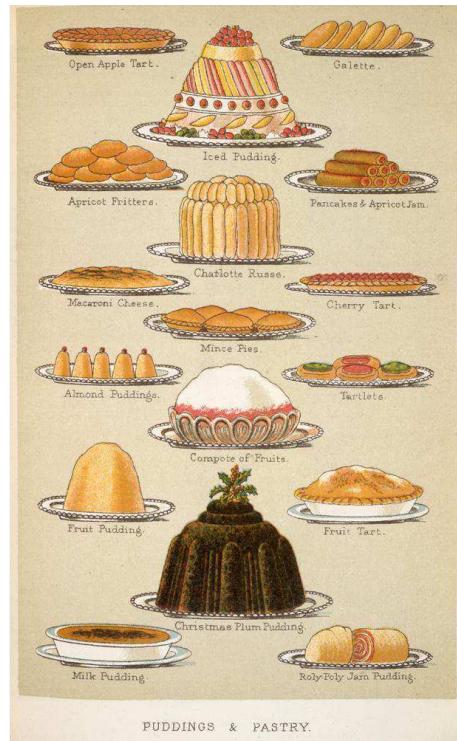
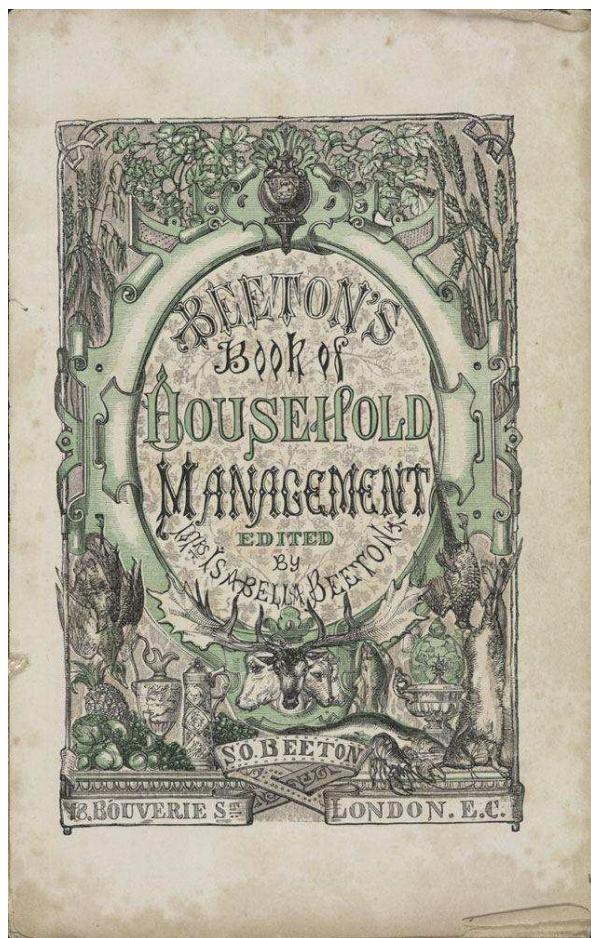
### Modern Cookery for Private Families

Hannah Glasse may have been more accessible than previous cookbook authors, but she was still a chef writing for other chefs—less-educated chefs, but still cooking professionals. Eliza Acton's book steps away from that. 'Modern Cookery for Private Families' was published in 1845, well after the industrial revolution, and was aimed specifically at housewives. This audience change necessitated the inclusion of exact quantities and cooking times, and it thus became one of the first 'modern' cookbooks.

(picture on previous page, bottom)

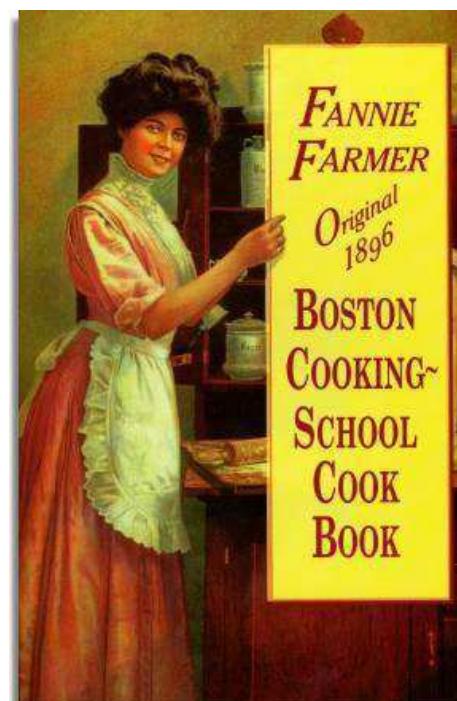
### Mrs Beeton's Book of Household Management

While Eliza Acton's book contained true innovations, its publishing house seems to have missed out on the marketing side of things ... Because Isabella Beeton's 'Mrs Beeton's Book of Household Management' was to become known as *the* Victorian cookbook. Published in 1861, it is essentially a collection of plagiarized recipes (including many recipes from Eliza Acton's book). What it did have: excellent illustrations, and plenty of them.



### Boston Cooking-School Cook Book

Fannie Farmer's cookbook is the first American contribution to this list. It was published in 1896, by the school principal of the Boston Cooking School. The keyword for this book: standardization. The modern American measuring system of cups and spoonfuls was introduced in this book.



It was also the first widely-known book to use a bullet list presentation for the ingredients, completely separate from the processing instructions that followed below that list.

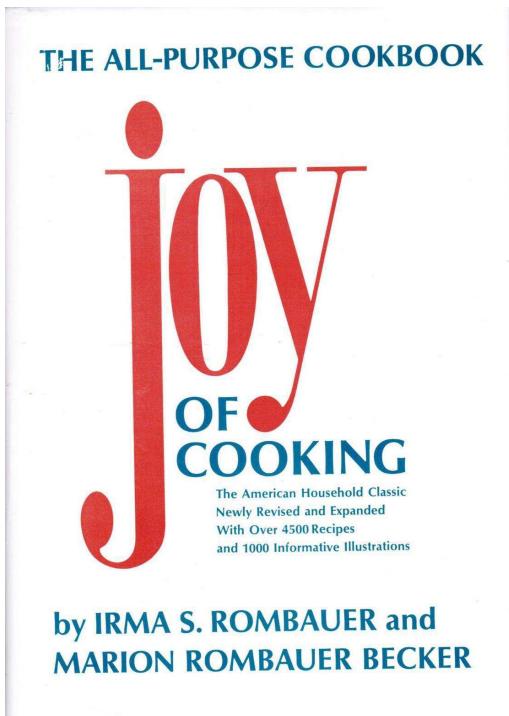
#### Brownies.

$\frac{1}{2}$ cup butter.	1 egg well beaten.
$\frac{1}{2}$ cup powdered sugar.	$\frac{1}{2}$ cup bread flour.
$\frac{1}{2}$ cup Porto Rico molasses.	1 cup pecan meat cut in pieces.

Mix ingredients in order given. Bake in small, shallow fancy cake tins, garnishing top of each cake with one-half pecan.

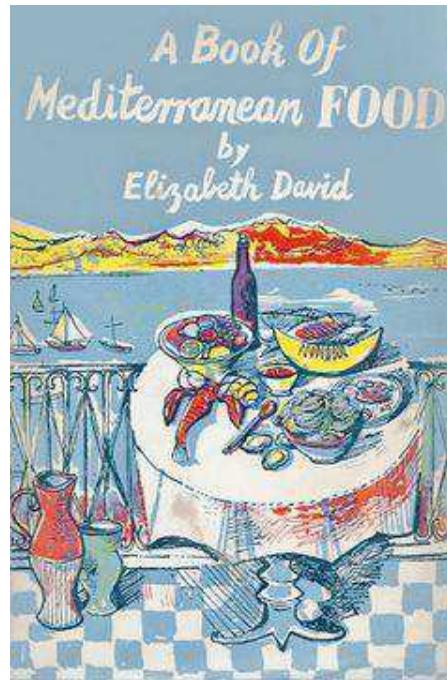
#### The Joy of Cooking

Entering the 20th century, Irma S. Rombauer's 'The Joy of Cooking' was published in 1931. This is the first book of all those in this article which I was familiar with even before I got interested in the history of cookbooks. In particular, I remember this cover (of a seventies edition) from the English language section of book sales that I visited as a small boy.



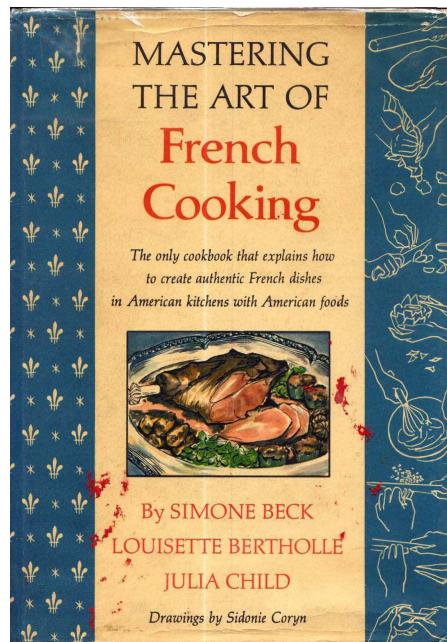
#### A Book of Mediterranean Food

This book is perhaps a bit of a surprise. It is by Elizabeth David, published in 1950, after her return from the Mediterranean to England a few years after WWII. Interesting points about this book: it had 'mood pictures' (black and white engravings of Mediterranean scenes) and targeted a specific 'foreign' food culture exclusively.



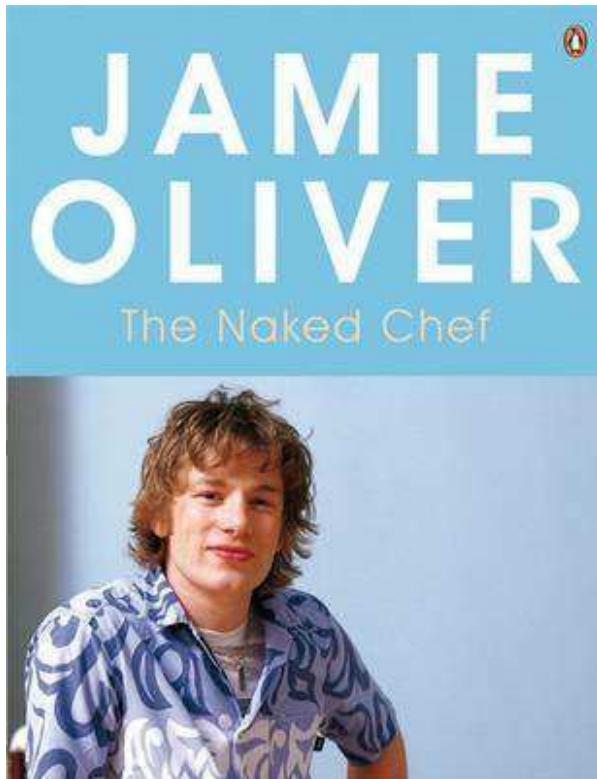
#### Mastering the Art of French Cooking

As the preface states, this book could also be called 'French Cooking from the American Supermarket.' First published in 1961, Simone Beck, Louisette Bertholle, and Julia Child brought French cuisine to the American audience. The recipes in this cookbook are not the easiest, but nevertheless it became very popular in the USA, not in the least thanks to Julia Child's television series 'The French Chef', first aired in 1963. Recently, interest soared again after the release of the movie 'Julie & Julia'.



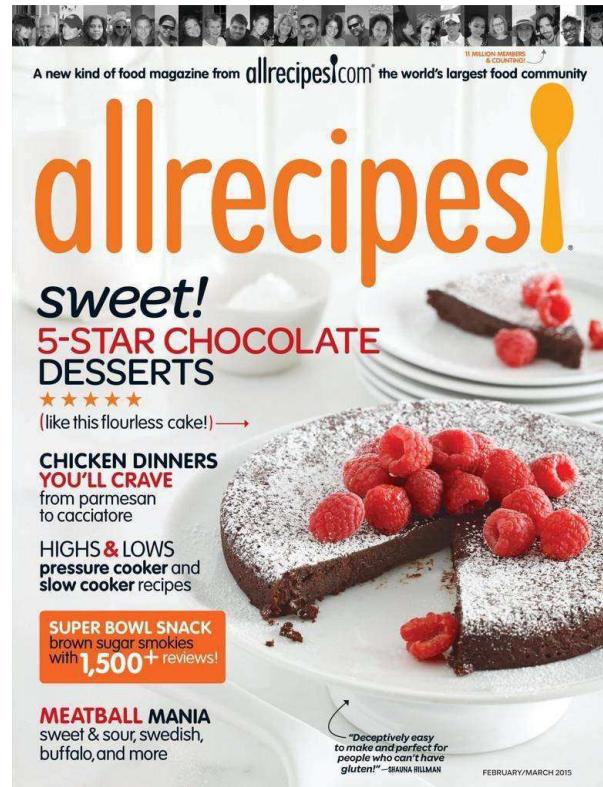
### The Naked Chef

The latter part of the 20th century had many nice cookbooks, as cheaper printing costs allowed for glossy books with full colour images at a reasonable price. But nothing had as much impact as Jamie Oliver's 'The Naked Chef', which came out in 1999. The BBC television programme of the same name made Jamie Oliver a celebrity overnight, and he has now written over a dozen well-selling cookbooks.



### Allrecipes.com

Modern people do not use cookbooks any more. No, they use the bookstand in the kitchen to hold an iPad while looking at [allrecipes.com](http://allrecipes.com) for a recipe. Sometimes with a video to explain the process. Originally just a web site, Allrecipes is now also a magazine, a YouTube channel, a set of online helper apps (such as a kitchen timer), and a set of mobile applications.



### Recipe Fiddle

If you prefer to have a cookbook generated *for* you, you can try 'Recipe Fiddle' by David Jarvis. This is a web site ([recipefiddle.com](http://recipefiddle.com)) that can generate a personalized cookbook for you, using ConTEXt to do the typesetting. Currently in beta.

### Recipe Fiddle



Create personalized recipe books in minutes.

[Watch Tutorial](#)

- ◊ Taco Hoekwater  
taco (at) elvenkind dot com

## Typographers' Inn

Peter Flynn

### The usability of digital typography

I have been asked to contribute to some research on this topic, so I have been trying to identify what there is, if anything, in specifically *digital* typography for which the usability factors are not already present in non-digital typography (hand-set type, hot-metal, analog film, etc.).

The traditional view of most typographic design is that it should keep out of the way and be invisible; anything which comes between reader and message should assist the passage of information, not hinder it. The moment the reader starts admiring the layout or the typeface instead of reading the text is the moment you have failed to convey the author's message. Present company excepted, of course. Adrian Frutiger (whose recent death is remembered on p. 182 f.) once said, 'The whole point with type is for you not to be aware it is there. If you remember the shape of a spoon with which you just ate some soup, then the spoon had a poor shape.' [1]

I have summarized what appear to be five main areas here, and I would be very interested to know if they correspond with the feelings of other TeX users.

**Design** Digital typography (well, pretty much *all* typography nowadays, apart from the metal type community) allows greater positional freedom than is possible in metal, so any measure of usability needs to consider the effects of this in design and layout. We also need to remember that Letraset (rub-down lettering) allowed much of the same freedom, and that largely preceded the switch to digital methods.

**Typefaces** The availability of typefaces has changed as well, so the choices are expanding on two fronts: faces that were no longer available are being digitized for modern use; and easy access to typeface design software by anyone with a computer has led to an explosion of new fonts, not all of them necessarily very usable. Within fonts, previously inaccessible or hard-to-obtain characters are now more widely available.

**Users' experiences of fonts** The level of user experience (UX) of software that uses fonts is now extensive. It started with word-processing, when a 'digital font' was just a dot-pattern for a nine-nozzle ink-jet printer or a removable typehead on a daisy-wheel printer. Now, virtually every application has configurable fonts, both for its own interface and for the output it produces,

and users have become accustomed to this flexibility and have come to expect it.

**Workflow** The production workflow has changed, both at the editorial and the design and composition stages, with the digital file being re-used between edit cycles instead of having to be re-keyed, or the (metal) type having to be kept standing. Changes are more easily made, both to the copy and the layout.

**Flexibility** Finally, there has been a shift in the degree to which additional features can be added. A book can become a web site, and a web site can be reformatted as a book (or ebook); and an article can be reproduced in many different guises, with the text staying unaltered, but the design and layout changing dramatically. Links can be revealed or hidden, although we still can't yet click on paper.

This may all sound like pie in the sky, as we still labor under the burden of evil file formats, incompatible equipment, or heavy-handed corporate or personal predilections. But I would hope that what we do, especially with TeX, and the way that we do it, makes the documents we produce usable for our readers. There are regulatory issues, too, such as the font requirements for pharmaceutical labelling information, and signage of various kinds.

There has been a recent discussion, on a professional usability mailing list, about research into performance increases related to 'information hierarchies'. I initially assumed this meant document structuring (chapter/section/subsection), but the term actually referred to font changes which imply degrees of importance (headings, emphasis, etc.). The discussion turned on studies which have found that certain typographical treatments may actually confuse readers and lead to lower comprehension. More on this another time.

If you have experienced usability changes as a result of typography, please let me know.

### Hierarchy and balance

On a similar note, I have been implementing several document classes for technical documentation. These are mostly for clients' internal use, but some are for white papers, the articles published by R&D departments highlighting such findings as they can safely release without damaging their competitive edge.

They need to reflect the corporate identity, of course, which can be anything from non-existent to extremely complex, but they also need to be slightly different from the rest of the organization's documents, as they will get exposure to a different set of people from most other documents. In most cases

the layout is specified by a designer, and it has been interesting to compare the styles.

There are usually three layers to an information hierarchy when instantiated typographically: major, minor, and body. The major level is usually for the titling, the minor for labelling the components which make up the body, and the body for the normal text. The boundaries sometimes blur: the face used for the minor level might also be used at subtitle level, but in general they are kept apart.

Figure 1 shows an example of this kind of design (adapted and anonymized from the originals).



**Figure 1:** Example of typographic hierarchy

The major and minor level (unusually) both use Raleway, which happens to be the corporate name-style, with the oddity that they always put the ‘Inc’ italicized in the body serif face. The title, subtitle, and date are in Raleway Bold.

The minor level headings (all the sectioning) are in Raleway Regular, as is the identity block at top right, except the corporate tagline is bold.

The body is in Warnock, spaced widely, with a larger size spaced even more widely for the summary. The side-notes, however, are in Raleway, to distinguish them more clearly from the body copy, as there is no rule between the columns.

The strong vertical stress given by the separation of the page into two unequal columns (text and wide margin), is balanced between the banner (the identity block), the title (for the date), and the body. Keeping the sans face for this right-hand material helps make it clear that it is distinct from the body.

I think they get away with it here, as the sans is the same face as the corporate name-style, and their globe logo is non-typographic. The two oddities (the italicized *Inc* and the page number centered in the right-hand column) are ‘features’ taken from the corporate style of the same company used in other areas.

Implementing this kind of layout in L<sup>A</sup>T<sub>E</sub>X requires a wide right-hand margin specified with the `geometry` package, and a `\maketitle` able to use it. The remainder of the document largely follows standard article conventions.

## Afterthought

Lots of people seem to want to use T<sub>E</sub>X on their Android devices, which I mentioned in an earlier column; and lots of people also use Emacs for T<sub>E</sub>X. A version of Emacs for Android put out some years ago failed on a broken terminal application, and no longer works because of Android’s new position-independent executable requirements.

David Megginson, whom some of you may know from his XML activities, has fixed this, recompiled the binary, and made it available on GitHub with instructions at <http://quoderat.megginson.com/2015/05/26/update-running-emacs-in-android-1-lollipop>.

I installed this on my Galaxy Note 4 during the XML Summer School recently, much to the amusement of my non-Emacs-using colleagues, and it seems to work just fine. I haven’t managed to bind it to a T<sub>E</sub>X executable yet, as I don’t know if the one that comes with the L<sup>A</sup>T<sub>E</sub>X Editor app is accessible, but watch this space.

## References

- [1] Monotype GmbH. Typography as the highest form of visual communication: A talk with Adrian Frutiger. *Linotype Font Feature*, July 2014. <http://www.linotype.com/de/2316/portrait.html>.

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

# L<sup>A</sup>T<sub>E</sub>X News

Issue 22, January 2015

## New L<sup>A</sup>T<sub>E</sub>X 2 <sub>$\varepsilon$</sub> bug-fix policy

### Introduction

For some years we have supplied bug fixes to the L<sup>A</sup>T<sub>E</sub>X 2 <sub>$\varepsilon$</sub>  kernel via the `fixltx2e` package. This kept the kernel stable, but at the expense of meaning that most users did not benefit from bug fixes, and that some compromises which were made to save space in the machines of the time are still affecting most users today.

In this release we have started a new update policy. All the fixes previously available via `fixltx2e` are now enabled *by default* in the format, as are some further extensions for extended T<sub>E</sub>X engines, ε-T<sub>E</sub>X, X<sub>Q</sub>T<sub>E</sub>X and LuaT<sub>E</sub>X. Compatibility and stability are still important considerations, and while most users will not notice these improvements, or will want to benefit from them, a new `latexrelease` package is provided that will revert all the changes and re-instate the definitions from earlier releases. The package can also be used with older releases to effectively *update* the kernel to be equivalent to this 2015 release.

A new document, `latexchanges`, is distributed with the release that documents all the changes to documented commands since the 2014 L<sup>A</sup>T<sub>E</sub>X release, and will be updated in future releases if further changes have been made.

### The `latexrelease` package

As noted above a new package is available to manage differences between L<sup>A</sup>T<sub>E</sub>X releases. If you wish to revert all changes back to the definitions as they were in previous releases you may start your document requesting the L<sup>A</sup>T<sub>E</sub>X release from May 2014:

```
\RequirePackage[2014/05/01]{latexrelease}
\documentclass{article}
```

Conversely if you start a large project now and want to protect yourself against possible future changes, you may start your document

```
\RequirePackage[2015/01/01]{latexrelease}
\documentclass{article}
```

Then the version of `latexrelease` distributed with any future L<sup>A</sup>T<sub>E</sub>X release will revert any changes made in that format, and revert to the definitions as they were at the beginning of 2015.

If you wish to share a document using the latest features with a user restricted to using an older format, you may use the form above and make the `latexrelease`

package available on the older installation. The package will then update the format definitions as needed to enable the older format to work as if dated on the date specified in the package option.

### The `\IncludeInRelease` command

The mechanism used in the `latexrelease` package is available for use in package code. If in your `zzz` package you have

```
\RequirePackage{latexrelease}
\IncludeInRelease{2015/06/01}
  {\zzz}{\zzz definition}
  \def\zzz.....new code
\EndIncludeInRelease
\IncludeInRelease{0000/00/00}
  {\zzz}{\zzz definition}
  \def\zzz.....original
\EndIncludeInRelease
```

then in a document using a format dated 2015/06/01 or later, the “new code” will be used, and for documents being processed with an older format, the “original” code will be used. Note the format date here may be the original format date as shown at the start of every L<sup>A</sup>T<sub>E</sub>X run, or a format date specified as a package option to the `latexrelease` package.

So if the document has

```
\RequirePackage[2014/05/01]{latexrelease}
\documentclass{article}
\usepackage{zzz}
```

then it will use the *original* definition of `\zzz` even if processed with the current format, as the format acts as if dated 2014/05/01.

### Limitations of the approach

The new concept provides full backward and forward compatibility for the L<sup>A</sup>T<sub>E</sub>X format, i.e., with the help of a current `latexrelease` package the kernel can emulate all released formats (starting with 2014/06/01<sup>1</sup>).

However, this is not necessarily true for all packages. Only if a package makes use of the `\IncludeInRelease` functionality will it adjust to the requested L<sup>A</sup>T<sub>E</sub>X release date. Initially this will only be true for a few selected packages and in general it may not even be

<sup>1</sup>Patching an older format most likely works too, given that the changes in the past have been minimal, though this isn’t guaranteed and hasn’t been tested.

advisable for packages that have their own well-established release cycles and methods.

Thus, to regenerate a document with 100% compatible behavior it will still be necessary to archive it together with all its inputs, for example, by archiving the base distribution trees (and any modifications made). However, the fact that a document requests a specific L<sup>A</sup>T<sub>E</sub>X release date should help identifying what release tree to use to achieve perfect accuracy.

### *Updates to the kernel*

#### *Updates incorporated from fixltx2e*

The detailed list of changes incorporated from fixltx2e is available in the new `latexchange` document that is distributed with this release. The main changes are that 2-column floats are kept in sequence with one column floats, corrections are made to the `\mark` system to ensure correct page headings in 2-column documents, several additional commands are made robust.

#### *$\varepsilon$ -T<sub>E</sub>X register allocation*

L<sup>A</sup>T<sub>E</sub>X has traditionally used allocation routines inherited from plain T<sub>E</sub>X that allocated registers in the range 0–255. Almost all distributions have for some years used  $\varepsilon$ -T<sub>E</sub>X based formats (or X<sub>E</sub>T<sub>E</sub>X or LuaT<sub>E</sub>X) which have  $2^{15}$  registers of each type ( $2^{16}$  in the case of LuaT<sub>E</sub>X). The `etex` package has been available to provided an allocation mechanism for these extended registers but now the format will by default allocate in a range suitable for the engine being used. The new allocation mechanism is different than the `etex` package mechanism, and supports LuaT<sub>E</sub>X's full range and an allocation mechanism for L<sup>A</sup>T<sub>E</sub>X floats as described below.

On  $\varepsilon$ -T<sub>E</sub>X based engines, an additional command, `\newmarks` is available (as with the `etex` package) that allocates extended  $\varepsilon$ -T<sub>E</sub>X marks, and similarly if X<sub>E</sub>T<sub>E</sub>X is detected a new command `\newXeTeXintercharclass` is available, this is similar to the command previously defined in the `xelatex.ini` file used to build the `xelatex` format.

#### *Additional L<sup>A</sup>T<sub>E</sub>X float storage*

L<sup>A</sup>T<sub>E</sub>X's float placement algorithm needs to store floats (figures and tables) until it finds a suitable page to output them. It allocates 18 registers for this storage, but this can often be insufficient. The contributed `morefloats` package has been available to extend this list; however, it also only allocates from the standard range 0–255 so cannot take advantage of the extended registers. The new allocation mechanism in this release

incorporates a new command `\extrafloats`. If you get the error: `Too many unprocessed floats`, then you can add (say) `\extrafloats{500}` to the document preamble to make many more boxes available to hold floats.

#### *Built-in support for Unicode engines*

The kernel sources now detect the engine being used and adjust definitions accordingly, this reduces the need for the “.ini” files used to make the formats to patch definitions defined in `latex.ltx`.

As noted above the format now includes extended allocation routines.

The distribution includes a file `unicode-letters.def` derived from the Unicode Consortium's Unicode Character Data files that details the upper and lower case transformation data for the full Unicode range. This is used to set the `lccode` and `uccode` values if a Unicode engine is being used, rather than the values derived from the T1 font encoding which are used with 8-bit engines.

Finally `\typein` is modified if LuaT<sub>E</sub>X is detected such that it works with this engine.

### *l3build*

This release has been tested and built using a new build system implemented in Lua, intended to be run on the `texlua` interpreter distributed with modern T<sub>E</sub>X distributions. It is already separately available from CTAN. This replaces earlier build systems (based at various times on `make`, `cons`, and Windows `bat` files). It allows the sources to be tested and packaged on a range of platforms (within the team, OS X, Windows, Linux and Cygwin platforms are used). It also allows the format to be tested on X<sub>E</sub>T<sub>E</sub>X and LuaT<sub>E</sub>X as well as the standard pdft<sub>E</sub>X/ $\varepsilon$ -T<sub>E</sub>X engines.

### *Hyperlinked documentation and TDS zip files*

As well as updating the build system, the team have looked again at exactly what gets released to CTAN. Taking inspiration from Heiko Oberdiek's `latex-tds` bundle, the PDF documentation provided now includes hyperlinks where appropriate. This has been done without modifying the sources such that users without `hyperref` available can still typeset the documentation using only the core distribution. At the same time, the release now includes ready-to-install TDS-style zip files. This will be of principal interest to T<sub>E</sub>X system maintainers, but end users with older machines who wish to manually update L<sup>A</sup>T<sub>E</sub>X will also benefit.

# LATEX News

Issue 23, October 2015

## Contents

### Enhanced support for LuaTEX

Names of LuaTEX primitive commands . . . . .	1
T <sub>E</sub> X commands for allocation in LuaTEX . . . . .	2
Predefined Lua functions . . . . .	2
Support for older releases and plain T <sub>E</sub> X . . . . .	2
Additional LuaTEX support packages . . . . .	2

### More floats and inserts

### Updated Unicode data

### Support for comma accent

### Extended inputenc

### Pre-release releases

### Updates in tools

### Enhanced support for LuaTEX

As noted in LATEX News 22, the 2015/01/01 release of LATEX introduced built-in support for extended T<sub>E</sub>X systems.

The range of allocated register numbers (for example, for count registers) is now set according to the underlying engine capabilities to 256, 32768 or 65536. Additional allocators were also added for the facilities added by ε-T<sub>E</sub>X (\newmark) and X<sub>E</sub>T<sub>E</sub>X (\newXeTeXintercharclass). At that time, however, the work to incorporate additional allocators for LuaTEX was not ready for distribution.

The main feature of this release is that by default it includes allocators for LuaTEX-provided features, such as Lua functions, bytecode registers, catcode tables and Lua callbacks. Previously these features have been provided by the contributed luatex (Heiko Oberdiek) and luatexbase (Élie Roux, Manuel Pégourié-Gonnard and Philipp Gesang) packages. However, just as noted with the etex package in the previous release, it is better if allocation is handled by the format to avoid problems with conflicts between different allocation schemes, or definitions made before a package-defined allocation scheme is enabled.

The facilities incorporated into the format with this release, and described below, are closely modelled on the luatexbase package and we thank the authors, and

especially Élie Roux, for help in arranging this transition.

The implementation of these LuaTEX features has been redesigned to match the allocation system introduced in the 2015/01/01 LATEX release, and there are some other differences from the previous luatexbase package. However, as noted below, luatexbase is being updated in line with this LATEX release to provide the previous interface as a wrapper around the new implementation, so we expect the majority of documents using luatexbase to work without change.

#### *Names of LuaTEX primitive commands*

Names of LuaTEX primitive commands . . . . .	2
The 2015/01/01 LATEX release for the first time initialised LuaTEX in latex.ltx if LuaTEX is being used. Following the convention used in the contributed luatex.ini file used to set up the format for earlier releases, most LuaTEX-specific primitives were defined with names prefixed by luatex. This was designed to minimize name clashes but had the disadvantage that names did not match the LuaTEX manual, or the names used in other formats, and produced some awkward command names such as \luatexluafunction. From this release the names are enabled without the luatex prefix.	2
Following the convention used in the contributed luatex.ini file used to set up the format for earlier releases, most LuaTEX-specific primitives were defined with names prefixed by luatex. This was designed to minimize name clashes but had the disadvantage that names did not match the LuaTEX manual, or the names used in other formats, and produced some awkward command names such as \luatexluafunction. From this release the names are enabled without the luatex prefix.	2

In practice this change should not affect many documents; relatively few packages access the primitive commands, and many of those are already set up to work with prefixed or unprefixed names, so that they work with multiple formats.

For package writers, if you want to ensure that your code works with this and earlier releases, use unprefixed names in the package and ensure that they are defined by using code such as:

```
\directlua{tex.enableprimitives("",  
    tex.extraprimitives(  
        "omega", "aleph", "luatex"))}
```

Conversely if your document uses a package relying on prefixed names then you can add:

```
\directlua{tex.enableprimitives("luatex",  
    tex.extraprimitives(  
        "omega", "aleph", "luatex"))}
```

to your document.

Note the compatibility layer offered by the luatexbase package described below makes several commands available under both names.

As always, this change can be reverted using:  
`\RequirePackage[2015/01/01]{latexrelease}`  
 at the start of the document.

#### *TeX commands for allocation in LuaTeX*

For detailed descriptions of the new allocation commands see the documented sources in `ltluatex.dtx` or chapter N of `source2e`; however, the following new allocation commands are defined by default in LuaTeX: `\newattribute`, `\newcatcodetable`, `\newluafunction` and `\newwhatsit`. In addition, the commands `\setattribute` and `\unsetattribute` are defined to set and unset Lua attributes (integer values similar to counters, but attached to nodes). Finally several catcode tables are predefined: `\catcodetable@initex`, `\catcodetable@string`, `\catcodetable@latex`, `\catcodetable@atletter`.

#### *Predefined Lua functions*

If used with LuaTeX, L<sup>A</sup>T<sub>E</sub>X will initialise a Lua table, `luatexbase`, with functions supporting allocation and also the registering of Lua callback functions.

#### *Support for older releases and plain TeX*

The LuaTeX allocation functionality made available in this release is also available in plain TeX and older L<sup>A</sup>T<sub>E</sub>X releases in the files `ltluatex.tex` and `ltluatex.lua` which may be used simply by including the TeX file: `\input{ltluatex}`. An alternative for old L<sup>A</sup>T<sub>E</sub>X releases is to use:

`\RequirePackage[2015/10/01]{latexrelease}`  
 which will update the kernel to the current release, including LuaTeX support.

#### *Additional LuaTeX support packages*

In addition to the base L<sup>A</sup>T<sub>E</sub>X release two packages have been contributed to the `contrib` area on CTAN. The `ctablestack` package offers some commands to help package writers control the LuaTeX `catcodetable` functionality, and the `luatexbase` package replaces the previously available package of the same name, providing a compatible interface but implemented over the `ltluatex` code.

#### *More floats and inserts*

If ε-T<sub>E</sub>X is available, the number of registers allocated in the format to hold floats such as figures is increased from 18 to 52.

The extended allocation system introduced in 2015/01/01 means that in most cases it is no longer necessary to load the `etex` package. Many classes and packages that previously loaded this package no longer do so. Unfortunately in some circumstances where a package or class previously used the `etex` `\reserveinserts` command, it is possible for a document that previously worked to generate an error

“no room for a new insert”. In practice this error can always be avoided by declaring inserts earlier, before the registers below 256 are all allocated. However, it is better not to require packages to be re-ordered and in some cases the re-ordering is complicated due to delayed allocations in `\AtBeginDocument`.

In this release, a new implementation of `\newinsert` is used which allocates inserts from the previously allocated float lists once the classical register allocation has run out. This allows an extra 52 (or in LuaTeX, 64 thousand) insert allocations which is more than enough for practical documents (by default, L<sup>A</sup>T<sub>E</sub>X only uses two insert allocations).

#### *Updated Unicode data*

The file `unicode-letters.def` recording catcodes, upper and lower case mappings and other properties for Unicode characters has been regenerated using the data files from Unicode 8.0.0.

#### *Support for comma accent*

The command `\textcommabelow` has been added to the format. This is mainly used for the Romanian letters S<sub>s</sub>T<sub>t</sub>. This was requested in `latex/4414` in the L<sup>A</sup>T<sub>E</sub>X bug tracker.

#### *Extended inputenc*

The `utf8` option for `inputenc` has been extended to support the letters s and t with comma accent, U+0218–U+021b. Similarly circumflex w and y U+0174–U+0177 are defined. Also U+00a0 and U+00ad are declared by default, and defined to be `\nobreakspace` and `\-` respectively.

The error message given on undefined UTF-8 input characters now displays the Unicode number in U+hex format in addition to showing the character.

#### *Pre-release releases*

The patch level mechanism has been used previously to identify L<sup>A</sup>T<sub>E</sub>X releases that have small patches applied to the main release, without changing the main format date.

The mechanism has now been extended to allow identification of pre-release versions of the software (which may or may not be released via CTAN) but can be identified with a banner such as  
`LaTeX2e <2015/10/01> pre-release-1`  
 Internally this is identified as a patch release with a negative patch level.

#### *Updates in tools*

The `multicol` package has been updated to fix the interaction with “here” floats that land on the same page as the start or end of a `multicols` environment.

## Introduction to list structures in L<sup>A</sup>T<sub>E</sub>X

Thomas Thurnherr

### Abstract

Lists are frequently used structures in documents as well as presentations. L<sup>A</sup>T<sub>E</sub>X distinguishes three types of lists: bulleted list, ordered list, and descriptive list. In this article, I introduce these three types of lists, describe basic manipulations, and provide information about packages that expand on standard list structures by adding extra flexibility.

### 1 Introduction

L<sup>A</sup>T<sub>E</sub>X distinguishes between three types of lists: bulleted list, ordered list, and descriptive list. The bulleted list, where the order of elements is not important, is called `itemize`. On the other hand, ordered lists are termed `enumerate`, as their elements are numbered. Lastly, `description` is a descriptive list, which generally describes words or phrases. Usage of the three list types is similar; they are implemented as environments and elements are added via the `\item` command.

```
\begin{list-type}
  \item text
  \item text
\end{list-type}
```

### 2 Bulleted/unordered list

A bulleted list is a list where the ordering of elements does not matter. The `itemize` environment creates an unordered list and elements are added with `item`. Here is an example:

```
\begin{itemize}
  \item A bulleted item
  \item Another bulleted item
  \item And another bulleted item
\end{itemize}
```

↓

- A bulleted item
- Another bulleted item
- And another bulleted item

The default label for unordered lists is a bullet (•). I will show later (section 6) how to replace the label with another symbol.

### 3 Numbered/ordered list

The behavior of an ordered list is similar to the unordered list. The only difference is that the label is a number or letter from the alphabet, which is

incremented for every element. By default, the label is an arabic number followed by a dot. Again, I will show later how to make changes to the way the label is typeset.

```
\begin{enumerate}
  \item An ordered item
  \item Another ordered item
  \item And another ordered item
\end{enumerate}
```

↓

1. An ordered item
2. Another ordered item
3. And another ordered item

L<sup>A</sup>T<sub>E</sub>X uses a `counter` (called `enumi`) to keep track of the number of elements in an ordered list. Therefore, list items can be referenced within as well as outside the list. An example is given below to illustrate how to cross-reference a list element.

```
\begin{enumerate}
  \item An ordered item\label{itm:myList}
  \item A reference to item \ref{itm:myList}
\end{enumerate}
```

↓

1. An ordered item
2. A reference to item 1

### 4 Descriptive lists

Unlike `itemize` and `enumerate`, a descriptive list, or `description`, does not have a label. Instead, a word or phrase is used, which is passed to `item` as an optional argument. This is shown in the following:

```
\begin{description}
  \item[First] A descriptive item
  \item[Second] Another descriptive item
  \item[Third] And another descriptive item
\end{description}
```

↓

- First** A descriptive item  
**Second** Another descriptive item  
**Third** And another descriptive item

### 5 Nested lists

Lists can be nested by placing a list environment inside another list environment. Different list types may be combined, as illustrated in the example below. When the same list types are nested, L<sup>A</sup>T<sub>E</sub>X uses different, predefined labels for each level of nesting. The default maximum level of nesting for each type

of list is four. If more than four lists of the same type are nested, L<sup>A</sup>T<sub>E</sub>X throws the error: “Too deeply nested”.

```
\begin{enumerate}
  \item An ordered list item
  \begin{enumerate}
    \item A nested ordered list item
    \begin{itemize}
      \item A nested unordered list item
    \end{itemize}
    \item Another nested ordered list item
  \end{enumerate}
  \item Another ordered list item
\end{enumerate}
```

↓

1. An ordered list item
  - (a) A nested ordered list item
    - A nested unordered list item
  - (b) Another nested ordered list item
2. Another ordered list item

## 6 List manipulations

The appearance of a list is alterable. For example, one might want a different label from the default, or to add/remove space between items. There are several packages which implement macros for list manipulations. Some of the most commonly used packages are: `enumerate` [1], `enumitem` [2], and `mdwlist` [4].

Here, I will focus on `enumitem` as it provides the most comprehensive set of macros to manipulate list structures and comes with extensive documentation. The commands here assume that this package is loaded in the preamble:

```
\usepackage{enumitem}
```

### 6.1 Changing the label

The label can be changed using the optional environment parameter `label`. In the example below I change the label of an unordered list, which is a bullet by default, to a diamond. Some symbols frequently used as labels for unordered lists are given in table 1 (left column).

```
\begin{itemize}[label=$\diamond$]
  \item A diamond-labelled item
  \item Another diamond-labelled item
\end{itemize}
```

↓

- ◊ A diamond-labelled item
- ◊ Another diamond-labelled item

**Table 1:** Label options for `itemize` and `enumerate`.

<code>itemize</code>		<code>enumerate</code>	
label	code	label	code
•	\$\bullet\$	1,2,...	\arabic*
—	\$-\$	i,ii,...	\roman*
.	\$\cdot\$	I,II,...	\Roman*
*, *	\$*\$, \$\star\$	a,b,...	\alph*
◊	\$\diamond\$	A,B,...	\Alpha*

Analogous to an unordered list, the label may be changed in a numbered list. Again, available options are listed in table 1 (right column). In the example below, I change the default label (`arabic`) to `roman` labelling. Moreover, the number may be combined with parentheses or any punctuation symbol.

```
\begin{enumerate}[label=(\roman*)]
  \item A roman-numeralled item
  \item Another roman-numeralled item
\end{enumerate}
```

↓

- (i) A roman-numeralled item
- (ii) Another roman-numeralled item

### 6.2 Resume numbering from previous ordered list

The `enumitem` package provides the option `resume`, which resumes numbering from the preceding ordered list in a new ordered list.

```
Ordered list:
\begin{enumerate}
  \item Ordered list item
\end{enumerate}
Resume previous ordered list:
\begin{enumerate}[resume]
  \item Resumed list item
\end{enumerate}
```

↓

- Ordered list:
1. Ordered list item
- Resume previous ordered list:
2. Resumed list item

### 6.3 Vertical space

`enumitem` implements several parameters to control vertical spacing outside and inside list structures. The parameters are summarized below:

**topsep** Whitespace above and below list

**partopsep** Extra whitespace when list starts new paragraph  
**itemsep** Spacing between elements in list  
**parsep** Spacing between paragraphs in list  
**noitemsep** Sets **itemsep** and **parsep** to 0pt  
**nosep** Removes vertical space completely

Except for the last two options, these parameters are used as key–value pairs, where the value specifies the amount of whitespace.

```
\begin{enumerate}[topsep=10pt]
  \item A numbered item
  \item Another numbered item
\end{enumerate}
```



1. A numbered item
2. Another numbered item

## 6.4 Horizontal space

The following parameters control horizontal spacing outside and inside list structures:

**leftmargin** Limits the list to the left  
**rightmargin** Limits the list to the right  
**listparindent** Paragraph indent within a list  
**labelsep** Separation between label and body  
**itemindent** Item indent (first line)

Both vertical and horizontal spacing may be controlled globally using the `\setlist` command. Changes can be applied to all list types simultaneously or restricted to a specific list type via an optional list type parameter. In the example below, the first command is limited to numbered lists, whereas the second command changes all three list types simultaneously.

```
\setlist[enumerate]{nosep}
\setlist{topsep=0pt, itemsep=0pt}
```

## 7 Inline list

The **paralist** package [5] provides macros for inline lists, where all elements of a list are displayed within the same paragraph. The package implements three alternative list environments called: **inparaitem**, **inparaenum**, and **inparadesc**. Their usage is similar to the standard list environments.

The **paralist** package implements environments for inline lists. These are:  
`\begin{inparaenum}`  
 `\item inparaitem,`

```
\item inparaenum, and
\item inparadesc.
\end{inparaenum}
```



The **paralist** package implements environments for inline lists. These are: 1. **inparaitem**, 2. **inparaenum**, and 3. **inparadesc**.

## 8 Reverse numbered list

The **etaremune** package [3] implements an environment with the same name as the package to reverse the numbers of elements in an ordered list. Here is an example:

The three most popular movies in IMDB are, starting from the third:  
`\begin{etaremune}`  
 `\item The Godfather: Part II (1974)`  
 `\item The Godfather (1972)`  
 `\item The Shawshank Redemption (1994)`  
`\end{etaremune}`



The three most popular movies in IMDB are, starting from the third:  

3. The Godfather: Part II (1974)
2. The Godfather (1972)
1. The Shawshank Redemption (1994)

## References

- [1] **enumerate**—enumerate with redefinable labels. <http://ctan.org/pkg/enumerate>. Accessed: 2015-09-14.
- [2] **enumitem**—control layout of itemize, enumerate, description. <http://ctan.org/pkg/enumitem>. Accessed: 2015-09-14.
- [3] **etaremune**—reverse-counting enumerate environment. <http://ctan.org/pkg/etaremune>. Accessed: 2015-09-14.
- [4] **mdwlist**—miscellaneous list-related commands. <http://ctan.org/pkg/mdwlist>. Accessed: 2015-09-18.
- [5] **paralist**—enumerate and itemize within paragraphs. <http://ctan.org/pkg/paralist>. Accessed: 2015-09-14.

◊ Thomas Thurnherr  
 thomas.thurnherr (at) gmail dot com  
<http://texblog.org>

---

**gradstudentresume: A document class for graduate student CVs**

Anagha Kumar

### Abstract

Despite casting a rather wide net for a template for academic CVs, I was unable to find one that caught my fancy. I therefore chose to write my own document class. This paper aims to introduce this document class as well as provide some practical tips on writing a document class in L<sup>A</sup>T<sub>E</sub>X.

### 1 Introduction

Despite the widespread usage of L<sup>A</sup>T<sub>E</sub>X among mathematicians, statisticians, economists, computer scientists, and others with a quantitative bent, the community of L<sup>A</sup>T<sub>E</sub>X contributors is relatively small. This was borne out during my quest for a suitable CV template. After being unable to find one to my liking, I felt compelled to write a document class of my own and upload it onto CTAN. Since I am a ten-finger clinger to the notion that contributions should be accompanied by appropriate documentation, I decided to submit a paper detailing both the usage of the document class and offering hints on how to write a document class of one's own.

My document class is called `gradstudentresume` and serves as a template for graduate students writing an academic CV. The package is available at <http://ctan.org/pkg/gradstudentresume>.

### 2 Constructing the document class

Writing an academic CV can be cumbersome due both to the length of such CVs and the repetitive nature of the material. Graduate students often find themselves struggling with constantly formatting line after line. As such, this document class aimed to define environments and commands to make such tasks less repetitive. While we do not need to go over every line in the `.cls` file, some features warrant discussion.

First, despite the availability of the `\smallskip`, `\medskip`, and `\largeskip` commands, I believe that it is important to exercise precise control over the output of these documents. So, the `gradstudentresume` document class contains new commands (namely `\smallspace`, `\medspace`, and `\largeSpace`) that leave spaces of 1 mm, 3 mm, and 5 mm respectively. Similarly, I defined a new environment `wrapped` which decreases the separation between items in a list. Ordinarily, one would simply use the `itemize` environment but defining one's own environment allows one to exercise far more control over the layout of the desired document.

ANAGHA KUMAR  
Address: 1711 35th Street NW, Apartment 23, Washington DC 20007  
Telephone: 012-345-6789  
Email: anaghakumar2405@gmail.com

#### EDUCATION

B.A., Mathematics (Honors) <i>Bryn Mawr College</i> Graduation Honors: Magna Cum Laude	May 2011 Bryn Mawr, PA
--	---------------------------

M.S., Biostatistics <i>The Perelman School of Medicine at the University of Pennsylvania</i> Thesis: Lymphedema Assessment of the Breast, Arm, and Torso Advisor: Dr. Andrea Troxel	May 2013 Philadelphia, PA
--	------------------------------

M.A., Statistics <i>The Wharton School of the University of Pennsylvania</i> Thesis: Statistical Analyses Using Dynamic Documents Advisor: Dr. Dylan Small	May 2015 Philadelphia, PA
---	------------------------------

#### WORK EXPERIENCE

Statistician - MedStar Health Research Institute <i>Joint appointment as Statistician at the Georgetown University Hospital</i>	2015–present
--	--------------

- Responsible for the design and analysis of statistically valid clinical trials and observational studies.
- Authored the statistical portions of manuscripts, grants, and IRB submissions.

#### PUBLICATIONS

- Brown J.C., Kumar, A., Cheville A.L., Tchou J.C., Troxel A.B., Harris S.R., and Schmitz K.H. (2015). Association between lymphedema self-care adherence and lymphedema outcomes among women with breast cancer-related lymphedema. *Am J Phys Med Rehabil.*, 94 (4), 288–96.

#### PRESENTATIONS

CONTRIBUTED <b>Race or Resource?: Differences in Lymphedema Severity among Obese Breast Cancer Survivors</b> • Trans-disciplinary Research on Energetics & Cancer Scientific Meeting (Poster) <i>Fred Hutchinson Cancer Research Center</i>	September 2015 Seattle, WA
--	-------------------------------

#### TEACHING

TEACHING ASSISTANT <b>Fundamentals of Biostatistics</b> <i>Professor: Dr. Jason Roy (University of Pennsylvania Department of Biostatistics)</i>	Spring 2013
--	-------------

- Full semester course for 30 medical fellows and residents.
- Graded homework assignments, assisted students during office hours, and served as a Stata lab instructor.

1

**Figure 1:** The first page of my CV, as an example.

Next, the document class contains a new environment called `info` for name, address, etc. Here is its definition:

```
\newenvironment{info}[4]
{\begin{center}
 \Large \textsc{\#1} \\
 \normalsize \textit{Address}: \#2 \\
 \textit{Telephone}: \#3 \\
 \textit{Email}: \#4
\end{center}}
{\large\space}
```

This is an environment with four parameters and centered text. The reader should take care to follow the L<sup>A</sup>T<sub>E</sub>X syntax of specifying the number of parameters in square brackets after the `newenvironment` command and then preceding parameter numbers with # in the body of the defined environment.

This document class also contains several new commands such as `\sectionrule` (which draws a horizontal line across the page), `\undergrad`, `\grad`, `\sectionheading`, and `\subsectionheading`. These commands all take a varying number of parameters as

arguments. A short sample of the output for my own CV is shown above; for a complete example, readers are advised to read and compile the `example.tex` file included in the package.

Other commands such as `\reference`, `\paper`, and `\presentation` have been defined to help authors avoid the cumbersome process of formatting certain sections repeatedly. For instance, here is the definition of the `\reference` command:

```
\newcommand{\reference}[7]
{\textbf{\#1} \\
 \#2\hfill {Telephone: \#3} \\
 \#4\hfill {Email: \#5} \\
 \#6 \\
 \#7}
```

Here, the parameters are name (#1), title (#2), telephone number (#3), department (#4), email (#5), school (#6), and address line 2 (#7). This is useful since it eliminates the need for authors to repeatedly format the references they provide. Again, see the `example.tex` file for example usage.

In the same vein, this is the definition for the `\paper` command:

```
\newcommand{\paper}[7]
{\#1 (\#2). \#3. \textit{\#4, #5}
 (\#6), \#7.}
```

Here, parameter 1 is for authors, 2 is for year, 3 is for title, 4 is for journal, 5 is for volume, 6 is for number, and 7 is for pages. Students with many publications will especially appreciate this feature since it eliminates the repetitive task of formatting each publication separately.

Finally, new environments `pres` (presentations) and `desc` (descriptions of job positions, etc.) were defined so that authors need not keep formatting sections that require some elaboration. Here is the definition for `desc`:

```
\newenvironment{desc}[3]
{\noindent\textbf{\#1}\hfill \#2 \\
 \textit{\#3}
 \begin{wrapped}}
\end{wrapped}
\medspace
```

We see that the `wrapped` environment is used within the `desc` environment. This illustrates how nested environments can be created.

### 3 Usage

This section shows some examples of the usage of this document class. The goal is to illustrate how several environments and commands can be used in the construction of a resume. For instance, consider:

```
\sectionheading{Teaching}
\subsectionheading{Teaching Assistant}
```

```
\begin{desc}
{Fundamentals of Statistics}
{Spring 2013}
{Professor Jason Roy}
\item{Course for 30 medical residents.}
\item{Graded homework assignments.}
\end{desc}
```

First, we see that the section and subsection are titled “Teaching” and “Teaching Assistant” respectively. Next, we encounter the `desc` environment. The appropriate syntax is `\begin{desc}` with the three arguments `Fundamentals of Statistics`, `Spring 2013`, and `Professor Jason Roy` provided in parentheses immediately after. As noted in the previous section, the body of the `desc` environment contains the nested `wrapped` environment. Here, the items in the itemized list in the above example are spaced more closely than they would have been had they been a part of L<sup>A</sup>T<sub>E</sub>X’s standard `itemize` environment. Further, there is no need to type `\begin{wrapped}` and `\end{wrapped}` since the `desc` environment’s definition already includes them.

Next, we examine the `pres` environment, for presentations. Below is its definition, preceded by the definition for the `presentation` command:

```
\newcommand{\presentation}[4]
{\#1\hfill \#2 \\
 \textit{\#3}\hfill \#4}

\newenvironment{pres}[1]
{\textbf{\#1}% name of the presentation
 \begin{wrapped}}
\end{wrapped}
\medspace
```

For the `presentation` command, the 4 parameters are the section of the conference, date, conference name and location respectively. Much like `desc`, the `wrapped` environment is nested in the `pres` environment. Here’s an example of its usage:

```
\begin{pres}{Dynamic Documents}
\item{\presentation{STAT 701}
 {December 2015}
 {University of Pennsylvania}
 {Philadelphia, PA}}
\end{pres}
```

Again, the user need not specify `\begin{wrapped}` and `\end{wrapped}` when using the `pres` environment since the `wrapped` environment is already included in the definition of `pres`. The reader should also note that the `presentation` command is used within the body of the environment. This is a useful feature since users thereby bypass the need to format each presentation individually and can instead use the command within the body of the environment instead.

The appendix provides definitions for all of the commands and environments in this document class. Again, for an example of a full CV written using this document class, the reader is encouraged to download and run the `example.tex` file available as part the package, e.g., via <http://ctan.org/pkg/gradstudentresume>.

## 4 Discussion

At the risk of preaching to the choir, L<sup>A</sup>T<sub>E</sub>X is not a tool to be used but a product to be developed. Anybody can install a package somebody else has written or use a document class someone else was ingenious enough to come up with. Therein lies the distinction between producers of knowledge and mere consumers. In separating the chaff from the wheat, the inability to *create* knowledge is perhaps the biggest tell-tale. Thus, my biggest hope in writing this paper is to encourage others to contribute to CTAN.

## 5 Appendix

```
\newcommand{\smallspace}{\vspace{1mm}}
\newcommand{\medspace}{\vspace{3mm}}
\newcommand{\largeSpace}{\vspace{5mm}}

\newenvironment{info}[4]
{\begin{center}
  \Large \textsc{#1} \\
  \normalsize \textit{Address}: {#2} \\
  \textit{Telephone}: {#3} \\
  \textit{Email}: {#4}
\end{center}}
\largeSpace
% The parameters are name, address,
% telephone number, and email.

\newenvironment{wrapped}
{\begin{itemize}\setlength{\itemsep}{0pt}}
{\end{itemize}

\newcommand{\sectionrule}{%
  \noindent\hfil\rule{\textwidth}{.6pt}\hfil
  \vspace{2mm}

\newcommand{\undergrad}[5]{%
  \noindent
  \textbf{#1}\hfill {#2} \\
  \textit{#3}\hfill {#4} \\
  {\def\honors{#5}\ifx\honors\empty\else
    Graduation Honors: {#5}\fi}
```

```
\newcommand{\grad}[6]{%
  \noindent
  \textbf{#1}\hfill {#2} \\
  \textit{#3}\hfill {#4} \\
  Thesis: {#5} \\
  Advisor: {#6}}

\newcommand{\sectionheading}[1]{%
  \noindent {\large \textsc{#1}} \\
  \sectionrule}

\newcommand{\subsectionheading}[1]{%
  \noindent {\textsc{#1}}\smallspace}

\newcommand{\reference}[7]{%
  \textbf{#1} \\
  {#2}\hfill {Telephone: {#3}} \\
  {#4}\hfill {Email: {#5}} \\
  {#6} \\
  {#7}.

\newcommand{\paper}[7]{%
  {#1} ({#2}). {#3}.
  \textit{#4, #5} ({#6}), {#7}.}

\newcommand{\myname}{%
  \textbf{Kumar, A.}}
% to be customized for each individual's name

\newcommand{\presentation}[4]{%
  {#1}\hfill{#2} \\
  \textit{#3}\hfill{#4}

\newenvironment{pres}[1]
{\textbf{#1}}% name of the presentation
\begin{wrapped}}
% use the presentation command as the argument
\end{wrapped}
\smallspace

\newenvironment{desc}[3]
{\noindent \textbf{#1}\hfill {#2} \\
  \textit{#3}
\begin{wrapped}}
\end{wrapped}
\medspace

\diamond Anatha Kumar
1711 35th Street NW Apt. 23
Washington DC 20007
anaghakumar2405 (at) gmail dot com
```

---

**Glisternings: Longest string;  
Marching along; A blank argument;  
A centered table of contents**

Peter Wilson

Plain as the glistening planets shine  
When winds have cleaned the skies,  
Her love appeared, appealed for mine,  
And wantoned in her eyes.

*Songs of Travel*, ROBERT LOUIS STEVENSON

The aim of this column is to provide odd hints or small pieces of code that might help in solving a problem or two while hopefully not making things worse through any errors of mine.

The chief defect of Henry King  
Was chewing little bits of string.

*Cautionary Tales*, HILLAIRE BELLOC

## 1 Longest string

Romildo wrote to `comp.text.tex` saying that he tried to implement a macro for determining the longest string in a list but was having problems with the code [18]. Romildo's user view of the macro (`\Widest`) was like this:

```
\newdimen\mydimen
\def\Format#1{\itshape\tiny #1}
\Widest{\mydimen}{\Format}{Good,morning,world}
\the\mydimen
```

There were several responses, including ones from GL [8] and Heiko Oberdiek [16] who got into a bit of a discussion about their suggested solutions, partly because GL preferred the strings to look like multiple arguments (e.g., `{a}{bbb}{cc}`) and Heiko appeared to lean more towards a single argument with the strings being separated by commas (e.g., `(a,bbb,cc)`).

GL suggested (I have used `\Widestg` for GL's macro and `\Widesth` for Heiko's to distinguish between them):

```
\makeatletter
\newskip\result
\def\Widestg#1#2#3\Widestg{%
  #1 = Format
  \setbox\z@ \hbox{#1#2}%
  \ifdim\wd\z@ > \result
    \result \wd\z@
    \edef\longest{#2} % added by PW
    \def\flong{{#1\longest}} % added by PW
  \fi
  \ifx\relax#2\else
    \Widestg#1#3\Widestg
  \fi}
\makeatother
```

Peter Wilson

```
...
\result=0pt
\Widest{\textbf}{one}{two}{three}\relax\Widest
\the\result \\
\longest\ \the\result\% added by PW
\flong\ \the\result% added by PW
```

I added the code for `\longest` which contains the longest string and `\flong` to typeset it using the specified format. This code, applying the macro to the list `{one}{two}{three}`, results in:

```
26.13898pt
three 26.13898pt
three 26.13898pt
```

Heiko came up with a version that uses the `kvsetkeys` package [14] for parsing a comma-separated list where spaces at the beginning and end of an entry are ignored.

```
\usepackage{kvsetkeys}
\newcommand*\Format[1]{\textit{\tiny #1}}
\newlength\WidestResult
\makeatletter
\@ifdefinable{\Widesth}{%
  \def\Widesth#1#2(#3){%
    #1=\z@ % 0 pt
    \comma@parse{#3}{%
      \settowidth\dimen@{#2{\comma@entry}}%
      \ifdim#1<\dimen@
        #1=\dimen@
        \edef\longest{\comma@entry} % PW added
        \def\flong{#2{\longest}} % PW added
      \fi
      \gobble % ignore list entry argument
    }%
  }%
}
\makeatother
...
\Widesth{\WidestResult}{\Format}{Good,morning,
world}
```

Just as with `\Widestg` I added the `\longest` and `\flong` code. Note that the comma-separated list of strings is enclosed in parentheses and not braces. The result from Heiko's example is:

```
21.64417pt
morning 21.64417pt
morning 21.64417pt
```

Applying GL's macro to Romildo's example as:

```
\result=0pt
\Widestg{\tiny\textit}{Good}{morning}
           {world}\relax\Widestg
\longest\ \the\result \\
```

```
\flong\ \the\result
```

results in:

```
morning 21.64417pt
morning 21.64417pt
```

which is the same as that from `\Widesth`.

Although both macros give the same result I prefer Heiko's user interface to GL's, but then you may think it should be the other way round.

Tear along the dotted line.

---

*Instruction, ANONYMOUS*

## 2 Marching along

### 2.1 Oddment

On `ctt` Roger said that he was

*...planning to take a string of the form `mm.nn.pp` where `mm`, `nn`, and `pp` are all integers, and test if `pp` is odd. So I'd like to write a macro that does that and use that as the parameter to `\ifodd`.*

Joseph Wright responded [22] that it sounded as though he wanted something like:

```
\makeatletter
\newcommand*{\MyFunction}[1]{%
  \My@function#1..\@nil\@stop}
\def\My@function#1.#2.#3#4\@stop{%
  \def\My@mm{\#1}%
  \def\My@nn{\#2}%
  \def\My@pp{}%
  \ifx#3\@nil\else
    \My@function@#3#4
  \fi
  % 0 below makes the test work when
  % \My@pp is empty
  \ifodd0\My@pp\relax
    Odd
  \else
    Even
  \fi}
\def\My@function@#1..\@nil{\def\My@pp{\#1}}
\makeatother
...
\MyFunction{11.22.33}
\MyFunction{11.22.44}
\MyFunction{11.22.}
\MyFunction{11}
```

With Joseph's code, running his suggested test examples results in:

Odd	Even	Even	Even
-----	------	------	------

Quite frankly, I do not understand just how his code works. In order to get a better feel for it I decided to write my own macros for dot-separated lists of one, two, and three numbers and then try to

extend them to deal with a list of arbitrary extent. Here are my efforts for the one, two, and three length lists. I included some diagnostic output to help when my code didn't work as I thought that it should.

Firstly, here are the code shorthands that I have used for the diagnostics—the `\cs` macro is defined in the `\tugboat` class, as shown.

```
%\DeclareRobustCommand\cs[1]{%
  \texttt{\char`\#\!1}}
\newcommand*{\sarg}[1]{\texttt{\{#\!1\}}}
\newcommand*{\csparg}[2]{\cs{\!1}\sarg{\!2}}
\newcommand*{\LRA}{%
  \ensuremath{\Longrightarrow}}
```

For a single number the command is:

```
\MyFunctionI{\langle N\rangle}
\newcommand*{\MyFunctionI}[1]{%
  \csparg{\MyFunctionI}{\!1} \LRA
  \ifodd0\#1\relax
    #1 Odd
  \else
    #1 Even
  \fi}
```

Some example results are:

```
\MyFunctionI{11} \Rightarrow 11 Odd
\MyFunctionI{22} \Rightarrow 22 Even
\MyFunctionI{} \Rightarrow Even
```

For a list of two numbers the command is:

```
\MyFunctionII{\langle N.N\rangle}
\makeatletter
\newcommand*{\MyFunctionII}[1]{%
  \csparg{\MyFunctionII}{\!1} \LRA
  \My@FunctionII\#1\@nil
  \ifodd0\My@last\relax
    \My@last\ Odd
  \else
    \My@last\ Even
  \fi}
\def\My@FunctionII#1.#2\@nil{%
  \def\My@last{\#2}}
\makeatother
```

Example results are:

```
\MyFunctionII{11.22} \Rightarrow 22 Even
\MyFunctionII{11.33} \Rightarrow 33 Odd
\MyFunctionII{11.} \Rightarrow Even
```

For a list of three numbers the command is:

```
\MyFunctionIII{\langle N.N.N\rangle}
\makeatletter
\newcommand*{\MyFunctionIII}[1]{%
  \csparg{\MyFunctionIII}{\!1} \LRA
  \My@FunctionIII\#1\@nil
  \ifodd0\My@last\relax
    \My@last\ Odd
  \else
```

```
\My@last\ Even
\fi}
\def\My@FunctionIII#1.#2.#3\@nil{%
\def\My@last{\#3}}
\makeatother
```

Some results are:

```
\MyFunctionIII{11.22.33} ⇒ 33 Odd
\MyFunctionIII{11.22.44} ⇒ 44 Even
\MyFunctionIII{11.33.} ⇒ Even
```

Based on the underlying idea—delimited arguments [1, 6, 9, 20]—of the above macros I then tried to develop one that would take a dot-separated list of any length and return whether the last number was odd or even.

I failed.

Eventually I remembered that the L<sup>A</sup>T<sub>E</sub>X kernel includes an \cfor macro for marching along a comma-separated list of elements and decided to try and create a version that would handle dot-separated lists. It is effectively a copy of the \cfor code replacing every ‘,’ with a ‘.’. I can’t pretend to understand how it works. I have named it \ford as shorthand for ‘\cfordot-separated-list’.

```
\makeatletter
% \cford NAME := LIST \do {BODY}
\long\def\cford#1:#2\do#3{%
\expandafter\def\expandafter\@fortmp
\expandafter{\#2}%
\ifx\@fortmp\@empty \else
\expandafter
\@forloopd#2.\@nil.\@nil\@#1{\#3}
\fi}

\long\def\@forloopd#1.#2.#3\@#4#5{%
\def#4{\#1}\ifx #4\@nnil \else
\#5\def#4{\#2}\ifx #4\@nnil
\else \#5\@forloopd #3\@#4{\#5}\fi\fi}

\long\def\@forloopd#1.#2\@#3#4{%
\def#3{\#1}\ifx #3\@nnil
\expandafter\@forloop \else
\#4\relax
\expandafter\@forloopd\fi#2\@#3{\#4}}
\makeatother
```

I did use this for a macro to handle unlimited length lists of the kind that Roger was interested in. Then there was a further posting from him [17] in response to Joseph (which I have abbreviated):

*Thank you. That works (and was quite educational). However, I failed to completely specify my problem ...*

*Here’s what I have:*

```
{a.b.c, x.y.z} or
{x.y.z} or
```

{, x.y.z}  
where a,b,c, x,y,z are integers.

*What I would like to do is to be able to set a switch in the file that if set then the ... would be included only if z is odd, but if the switch is not set then all ... will be included.*

This requirement seemed to me to be a candidate for a combination of \cfor to handle the comma-separated parts and \cford for the portions that are dot-separated.

Below is what I ended up with to handle an unlimited comma-separated list of unlimited dot-separated lists determining whether the last entry of all is odd or even.

First the \DotFunction for a dot-separated list of numbers. I have added some diagnostic print out just in case together with a means (\ifop) for enabling it. The macro is called like:

```
\DotFunction{N.N.N...N}
```

and sets \gotoddtrue if the last number in the list is odd.

```
\newif\ifgotodd
\newif\ifop
\optrue

\makeatletter
\def\DotFunction#1{%
\ifop \csparg{DotFunction}{#1} \LRA \fi
\def\My@last{\#1} in case arg is empty
\cford\scratch:=\#1\do{%
\edef\My@last{\scratch}%
\ifodd0\My@last\relax
\gotoddtrue
\ifop \My@last\ Odd \fi
\else
\gotoddfalse
\ifop \My@last\ Even \fi
\fi}
\makeatother
```

Some example results are:

```
\DotFunction{} ⇒ 0 Even
\DotFunction{11} ⇒ 11 Odd
\DotFunction{11.22} ⇒ 22 Even
\DotFunction{11.22.33} ⇒ 33 Odd
\DotFunction{11.22.33.44} ⇒ 44 Even
\DotFunction{11.nowt.33.44.55} ⇒ 55 Odd
\DotFunction{11..33.44.55} ⇒ 55 Odd
\newcommand*\{\numM\}{11.22.33.44.55.66.77}
\DotFunction{\numM} ⇒
\DotFunction{11.22.33.44.55.66.77} ⇒ 77 Odd
```

Note that for \DotFunction, only the last element in the list must be an integer (or blank), earlier

elements can be, for example, text. Further, unlike all the previous `\MyFunction...` macros, the argument may be a macro.

Finally, here is the end of the exercise—a generalised solution to Roger’s requests, called as `\HisFunction{<N>...N, ...N, ..., N...N}`

```
\makeatletter
\def\DotCommaFunction#1{%
\csparg{DotCommaFunction}{#1} \LRA
\opfalse% stop \DotFunction printing
\@for\first:=#1\do{%
  \DotFunction{\first}}%
\ifgotodd
\My@last\ Odd
\else
\My@last\ Even
\fi}
\makeatother
```

Some results of using `\DotCommaFunction`:

```
\DotCommaFunction{1.2.3, 4.5.7} ⇒ 7 Odd
\DotCommaFunction{, 4.5.7} ⇒ 7 Odd
\DotCommaFunction{4.5.7} ⇒ 7 Odd
\DotCommaFunction{1, 2.3, , 4.5, 7.8} ⇒ 8
Even
\DotCommaFunction{1,2.3, ,4.5,7.8} ⇒ 8 Even
```

All that remains is for the user to make appropriate changes to the actions of the odd/even result and to eliminate, or change, the diagnostic outputs to suit the application at hand.

## 2.2 Indexing into a list

Alastair asked [2]:

*I’ve got a question about comma separated lists. Is there any way that you can index elements in a list. Lists can be iterated over in the PGF/TikZ package’s `\foreach` loop. How can you access an element whilst not in a loop?*

Often responses to questions on `ctt` provide the bare bones of a solution, leaving the questioner to adapt or extend it to his own situation. There were several responses and the one I found that would best suit me was from Ulrike Fischer [7]. The following is essentially Ulrike’s code, edited to better fit the column:

```
\usepackage{tikz}
\def\values{i5, i4, i3, i2, i1}
\newcounter{loc}
\newcommand{\getitem}[1]{%
\setcounter{loc}{0}
\foreach \x in \values{%
  \stepcounter{loc}%
  \expandafter\xdef\csname
    alsval\the\value{loc}\endcsname{\x}%
\csname alsval#1\endcsname}
```

Ulrike’s `\getitem{<N>}` macro returns the item that is in the  $<N>$ th location in `\values` as `\alsvalN`. With:

```
\getitem{1}, \getitem{4}, \getitem{8}.
```

the result is:

```
i5, i2, .
```

I wondered if there was a solution that did not involve calling the `tikz` package and came up with the following which does not require any packages, being based on the L<sup>A</sup>T<sub>E</sub>X kernel’s `\@for` construct.

```
\let\xpf\expandafter % just to save some space
\makeatletter
\newcount\vindex
\newcommand*\getitem[2]{%
\xpf\xpf\xpf\@getit\xpf{#2}{#1}%
\theans}

\newcommand*\@getit[2]{%
\vindex=0
\def\theans{Index #2 is out of range.}%
\xdef\alist{#1}%
\@for\tmp := #1 \do{%
  \advance\vindex 1
  \ifnum\the\vindex=#2
    \xdef\theans{\tmp}%
  \fi}
\makeatother
```

The macro `\getitem{<N>}{<list>}` returns `\theans` as the value of the  $<N>$ th item in the `<list>` where `<list>` may be either a comma-separated list or a macro defined as one. I have included a check on whether  $<N>$  is valid for the given list (this would be better in the form of an error report in the `log` file external to the document instead of being typeset).

With these inputs

```
\getit{1}{\values},
\getit{4}{\values},
\getit{8}{\values}.
```

```
\getit{1}{i5, i4, i3, i2, i1},
\getit{4}{i5, i4, i3, i2, i1},
\getit{8}{i5, i4, i3, i2, i1}.
```

the results are:

```
i5, i2, Index 8 is out of range.
i5, i2, Index 8 is out of range.
```

The key problem that I had to solve in my method is that the ‘list’ that `\@for` operates on must be an actual sequence of comma-separated items and not a macro defined as such a list. That is why I have separated the code into two macros. The first to grab the list, be it actual or as a macro, and then

to hand that over to `\@getit` as an actual list by utilising a series of `\expandafters` within `\getit`.<sup>1</sup>

The tumult and the shouting dies,  
The captains and the kings depart,  
And we are left with large supplies  
Of cold blancmange and rhubarb tart.

*After the Party, RONALD KNOX*

### 3 A blank argument

The title of a posting by Matthew to `texhax` was *Finding blank argument to a macro*. There is a long history behind this kind of macro, initially posed as a challenge in Michael Downes' *Around the Bend* [5] series in the early 90s, and without looking any further I assumed that the solution would be the `ifmtarg` [3] package which provides a test as to whether a macro argument consists of zero or more blank spaces.

However, I was mistaken, as Matthew's posting continued [11]:

*I am trying to solve a problem in L<sup>A</sup>T<sub>E</sub>X that I thought would be relatively straightforward. I would like to make a macro that will evaluate its argument and tell me whether the result is blank or not ... I managed to come up with a T<sub>E</sub>X macro that handles different types of 'blank' pretty well. It properly recognizes an empty argument, empty braces, spaces, etc. It even works on another macro that evaluates to a blank, so I thought I was home free. However, as soon as I fed it a macro that takes an argument, bad things happen. I've attached a simple document below that shows the problem.*

The 'simple document' contains many lines of code implementing his `\blankArgTest{<arg>}`, together with examples of when it worked and when it didn't give the required result. With the macros:

```
\usepackage{ifthen}
\newcommand{\testA}{%
  \ifnum10=10 \empty\else A\fi}
\newcommand{\testB}{%
  \ifthenelse{10=10}{\empty}{B}}
\newcommand{\testC}[1]{%
  \ifnum#1=10 \empty\else C\fi}
```

`\blankArgTest` worked when `<arg>` was `\testA` but failed for `\testB` and `\testC`.

Michael Barr [4] came up with a remarkably simple solution which I am presenting as:

```
\newcommand{\IfBlank}[1]{%
```

<sup>1</sup> `\expandafter` and when it should be used is to me among the more difficult aspects of T<sub>E</sub>X code. I usually come to a solution by either following what others have done in similar circumstances or by much experimentation — otherwise known as errors and trials.

```
\setbox0\hbox{$\#1$}%
\ifdim\wd0=0pt
  Blank
\else
  Not blank
\fi}
```

The basic idea is to put the argument into an `\hbox` and check if the box's width is zero. This assumes that a 'blank' argument is one that results in no typeset material (or rather, anything typeset ends with zero width). With the following definitions:

```
\newcommand{\blank}{ }
\newcommand{\tout}{\typeout{Typeout}}
```

examples of the `\IfBlank` macro are:

```
\IfBlank{} Blank
\IfBlank{}{} Blank
\IfBlank{Text} Not blank
\IfBlank{\blank} Blank
\IfBlank{\tout} Blank
\IfBlank{{ }} Blank
\testA Blank
\testB Blank
\testC{10} Blank
```

A somewhat different need for an empty/blank argument was expressed by Timothy Murphy who wrote [13]:

*I have a macro `\cmd#1#2`. Both arguments are given in the form `{...}`. I'd like an empty second argument `{}` to be added if none is given, i.e., if the next character after `\cmd{...}` is not `{`.*

*What is the simplest way to do this?*

There were three interesting proposed solutions which I have given below.<sup>2</sup>

Heiko Oberdiek's was the first positive response and was essentially as follows [15]:

```
\newcommand*{\CmdH}[1]{%
  \begingroup
    % remember parameter
    \toks0={#1}%
    % look forward
    \futurelet\NextToken\CmdI}
\newcommand*{\CmdI}{%
  \ifx\NextToken\bgroup
    \edef\next{\endgroup
      \noexpand\CmdImpl{\the\toks0}}%
  \else
    \edef\next{\endgroup
      \noexpand\CmdImpl{\the\toks0}}%
  \fi}
```

<sup>2</sup> I have slightly edited the code, principally by using distinguished macro names instead of the somewhat generic `\cmd`, and using a common set of tests.

```

\newcommand{\CmdImpl}[2]{%
  \textbf{Heiko:} \\
  this is (#1) and here's (#2).} \\
\CmdH{abc}{def} \\
\CmdH{ghi}\relax \\
\CmdH{jkl} %

```

which results in:

**Heiko:** this is (abc) and here's (def).  
**Heiko:** this is (ghi) and here's ().  
**Heiko:** this is (jkl) and here's ().

Dan Luecking [10], noting that there were probably better ways (see Heiko's reply), responded with:

```
\makeatletter
\newcommand*\{\CmdD}[1]{%
  \@ifnextchar\bgroup
    {\Cmd@i{#1}\{\Cmd@i{#1}{}}}
  \newcommand*{\Cmd@i}[2]{%
    \textbf{Dan:}%
    this is (#1) and here's (#2).}
\makeatother
\CmdD{abc}{def} \\
\CmdD{ghi}\relax \\
\CmdD{jkl} %
```

which results in:

**Dan:** this is (abc) and here's (def).  
**Dan:** this is (ghi) and here's ().  
**Dan:** this is (jkl) and here's ().

Dan pointed out that his code does not examine the actual *next* character, but rather the next *nonspace* character. He also commented that Heiko's solution emulates a portion of \ifnextchar without the skipping of spaces.

Joseph Wright [21] proposed a solution based on the `xparse` package [19] developed as part of the L<sup>A</sup>T<sub>E</sub>X3 project. From the user's viewpoint it appears to be the simplest of the three proposed solutions.

```
\usepackage{xparse}
\NewDocumentCommand{\CmdJ}{mG{}}{%
    \textbf{Joseph:}%
    this is (#1) and here's (#2).}%
\CmdJ{abc}{def} \\
\CmdJ{ghi}\relax \\
\CmdJ{jkl} %
```

which results in:

**Joseph:** this is (abc) and here's (def).  
**Joseph:** this is (ghi) and here's ().  
**Joseph:** this is (jkl) and here's ().

The three very different implementations each handled all the test cases correctly.

America is a land whose center is nowhere;  
England one whose center is everywhere.

Pick Up Pieces, JOHN UPDIKE

## 4 A centered table of contents

Bogdan Butnaru<sup>3</sup> uses the `memoir` class and asked me how to have a centered table of contents (ToC). I came up with one solution and passed Bogdan's request on to Lars Madsen, who is now `memoir`'s maintainer, and he came up with a better solution; both of these were based on `memoir`'s tools for manipulating the ToC. I then came up with a more basic solution which is also applicable to the standard `book` and `report` classes.

In these classes a chapter entry is set by the `\l@chapter` macro, a section entry by `\l@section`, and so on. These may be redefined to produce centered entries. These macros have the general calling form of:

`\l@chapter{<number-and-title>} {<page>}`  
 where `<number-and-title>` has the form:  
`{\numberline}{num} title}`  
 where `\numberline` typesets the chapter number.  
 The `\l@...` macros also take into account whether  
 or not the entry should be printed and the surround-  
 ing vertical spacing. The *L<sup>A</sup>T<sub>E</sub>X Companion* [12, §2.3]  
 provides further information about ToCs and related  
 packages.

The following redefinition of \chapter will center the chapter entries, with the chapter number above the title, and a middle-dot between the title and page number.

```

\makeatletter
\renewcommand*{\l@chapter}[2]{%
  \ifnum\c@tocdepth>\m@ne % print chapter entry
    \addpenalty{-\@highpenalty}%
    \vskip 1em plus 0pt
  \begingroup
    \def\numberline##1{##1\\\nobreak}% number
    {\centering\bfseries
      #1~\textperiodcentered~#2\par}%
  \endgroup
}%
\fi}
\makeatother

```

The `\tableofcontents` macro uses `\chapter*` to set the title ragged right. A hack to that can be used to center the title is to make `\raggedright` into `\centering`.

```
\let\saverr\raggedright
```

---

<sup>3</sup> Private email, 2010/07/21

```
\newcommand*{\rrtocenter}{%
  \let\raggedright\centering
}\newcommand*{\restorerr}{%
  \let\raggedright\saverr
}\let\oldtoc\tableofcontents
\renewcommand*{\tableofcontents}{%
  {\rrtocenter\oldtoc}}
```

To typeset the ToC with the heading and chapter entries centered is now as easy as:

`\tableofcontents`

If you wanted the section entries to be centered then `\l@section` can be redefined in a similar, but not identical, manner to `\l@chapter`. However, centered section entries following a centered chapter entry in my view looks rather confusing.

If you want chapter headings to be centered, you can do:

```
{\rrtocenter
  \chapter[...]{...}
}
or
\rrtocenter
  \chapter[...]{...}
\restorerr
```

In each case the effect of `\rrtocenter` is limited to `\chapter`; if it were not then surprises could be in store later on.

## References

- [1] Paul W. Abrahams, Karl Berry, and Kathryn A. Hargreaves. *TEX for the Impatient*. Addison-Wesley, 1990. <http://ctan.org/pkg/impatient>.
- [2] Alastair. Indexing individual elements in a comma separated list. Post to `comp.text.tex` newsgroup, 17 October 2010.
- [3] Donald Arseneau and Peter Wilson. The `ifmtarg` package, 2009. <http://ctan.org/pkg/ifmtarg>.
- [4] Michael Barr. Re: [texhax] Finding blank argument to a macro. Post to `texhax` mailing list, 27 May 2010.
- [5] Michael Downes (ed. Peter Wilson). *Around the Bend*. The Herries Press, July 2008. <http://ctan.org/pkg/around-the-bend>.
- [6] Victor Eijkhout. *TEX by Topic, A TEXnician's Reference*. Addison-Wesley, 1991. ISBN 0-201-56882-9. Available at <http://www.eijkhout.net/tbt/>.
- [7] Ulrike Fischer. Re: Indexing individual elements in a comma separated list. Post to `comp.text.tex` newsgroup, 18 October 2010.
- [8] GL. Re: Finding the widest string. Post to `comp.text.tex` newsgroup, 4 May 2010.
- [9] Donald E. Knuth. *The TeXbook*. Addison-Wesley, 1984. ISBN 0-201-13448-9.
- [10] Dan Luecking. Re: A small LaTeX macro question. Post to `comp.text.tex` newsgroup, 4 June 2010.
- [11] Matthew. [texhax] finding blank argument to a macro. Post to `texhax` mailing list, 25 May 2010.
- [12] Frank Mittelbach and Michel Goossens. *The L<sup>A</sup>T<sub>E</sub>X Companion*. Addison Wesley, second edition, 2004. ISBN 0-201-36299-6.
- [13] Timothy Murphy. A small LaTeX macro question. Post to `comp.text.tex` newsgroup, 4 June 2010.
- [14] Heiko Oberdiek. The `kvsetkeys` package, 2010. <http://ctan.org/pkg/kvsetkeys>.
- [15] Heiko Oberdiek. Re: A small LaTeX macro question. Post to `comp.text.tex` newsgroup, 4 June 2010.
- [16] Heiko Oberdiek. Re: Finding the widest string. Post to `comp.text.tex` newsgroup, 4 May 2010.
- [17] Roger. Re: ifodd question. Post to `comp.text.tex` newsgroup, 18 May 2010.
- [18] Romildo. Finding the widest string. Post to `comp.text.tex` newsgroup, 3 May 2010.
- [19] The LaTeX3 Project. The `xparse` package, 2015. <http://ctan.org/pkg/xparse>.
- [20] Peter Wilson. Glisterings: More on paragraphs regular, L<sup>A</sup>T<sub>E</sub>X's defining triumvirate, *TEX*'s dictator. *TUGboat*, 29(2):324–327, 2008. <http://tug.org/TUGboat/tb29-2/tb92glister.pdf>.
- [21] Joseph Wright. Re: A small LaTeX macro question. Post to `comp.text.tex` newsgroup, 4 June 2010.
- [22] Joseph Wright. Re: ifodd question. Post to `comp.text.tex` newsgroup, 14 May 2010.

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

## Chemistry in L<sup>A</sup>T<sub>E</sub>X 2 <sub>$\varepsilon$</sub> — an overview of existing packages and possibilities

Clemens Niederberger

### Abstract

This article provides an overview of the most useful and popular packages for writing chemistry-related material with L<sup>A</sup>T<sub>E</sub>X 2 <sub>$\varepsilon$</sub> . It is based upon a series of blog posts written by the author on the (German-language) blog <http://texwelt.de/blog>.

### 1 Introduction

It was just about two years ago when I started a series of blog posts on chemistry-related L<sup>A</sup>T<sub>E</sub>X packages. Readers interested in the series and able to understand German will find the whole series on <http://texwelt.de/blog/tag/chemie>. This article more or less is a translation of those blog articles into English and follows the same structure:

- The basics using the `chemmacros` package, see section 2.
- Molecular formulas and reaction equations, see section 3.
- Structural formulas—the `chemfig` package, see section 4.
- Safety Data and GHS—the packages `ghsystem`, `rsphrases`, and `hpstatement`, see section 5.
- Numbering of compounds—the `chemnum` package, see section 6.
- Packages for special applications, see section 7.

I should add the disclaimer that I am the author of the majority of packages discussed. All packages mentioned in this article are on CTAN (<http://ctan.org/pkg/<pkgnname>>), and they are also all included in the two major T<sub>E</sub>X distributions, MiK<sub>T</sub>E<sub>X</sub> and T<sub>E</sub>X Live. Anyone with an up-to-date distribution can find their manuals by typing `texdoc <pkgnname>` on the command line.

### 2 The basics using the `chemmacros` package

The `chemmacros` package (disclaimer: I am the author) offers a variety of macros for different applications in chemistry: writing IUPAC names, supporting formal charges and oxidation numbers, displaying spectroscopy data, and much more. Recently (late August 2015), `chemmacros` was updated to v5.0 which came with huge changes [4]. This article assumes version 5.0 is available.

As of version 5.0, `chemmacros` is constructed in a modular way. Some of the modules are loaded by default, while others have to be loaded by the user. This article loads one additional module:

```
\usechemmodule{redox}
```

### 2.1 IUPAC names

IUPAC (International Union of Pure and Applied Chemistry) names can get rather long and hard to read. Moreover, line breaking often becomes problematic. Treating them as normal text is unlikely to have good results:

```
(4-(4,4'-Bis(dimethylaminophenyl)benz%hydryliden)cyclohexa-2,5-dien-1-yliden)%dimethylammoniumchlorid
```

```
(4-(4,4'-Bis(dimethylaminophenyl)benzhydryliden)cyclohexa-2,5-dien-1-yliden)dimethylammoniumchlorid
```

`chemmacros` offers the macro `\iupac` for easing both input and output. Inside of this command, ‘-’ outputs a dash which also allows for hyphenation in the rest of the word, similar to `\babelhyphen`. ‘|’ inserts a breakpoint along with a tiny amount of space, which can be customized.

```
\iupac{(4-(4,4'-Bis(dimethylamino)%phenyl)benz|hydryliden)cyclo|hexa%-2,5-dien-1-yliden)di|methyl|%ammonium|chlorid}
```

```
(4-(4,4'-Bis(dimethylaminophenyl)benzhydryliden)cyclohexa-2,5-dien-1-yliden)dimethylammoniumchlorid
```

If it were only about the line breaks an easier solution might be to use suitable `babel` shorthands. But `\iupac` does more: inside many macros for common typesetting tasks, IUPAC names are defined:

```
\iupac{\nitrogen-methyl|benz|amide} \\
\iupac{\cip{2S,3S}-Wein\s{"aure}} \\
\iupac{\zusammen-2-Butene}
```

*N*-methylbenzamide  
(*2S,3S*)-Weinsäure  
(*Z*)-2-Butene

### 2.2 Phase descriptors

`chemmacros` tries hard to follow IUPAC’s recommendations whenever possible. IUPAC’s demands concerning phase descriptors are the following:

The [...] symbols are used to represent the states of aggregation of chemical species. The letters are appended to the formula in parentheses and should be printed in Roman (upright) type without a full stop (period). [2, p. 54]

`chemmacros` offers ready-to-use macros for the most common phase descriptors which exactly follow this recommendation:

```
\ch{
  C\solid{} + 2 H2O\liquid{} \\
  -> \\
  CO2\gas{} + 2 H2\gas{} \\
}
C(s) + 2 H2O(l) → CO2(g) + 2 H2(g)
```

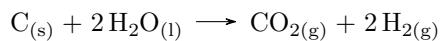
**Table 1:** Basic use cases for mhchem and chemformula.

<b>mhchem</b>		<b>chemformula</b>	
\ce{H2O}	H <sub>2</sub> O	\ch{H2O}	H <sub>2</sub> O
\ce{Sb2O3}	Sb <sub>2</sub> O <sub>3</sub>	\ch{Sb2O3}	Sb <sub>2</sub> O <sub>3</sub>
\ce{H+}	H <sup>+</sup>	\ch{H+}	H <sup>+</sup>
\ce{H2O}	H <sub>2</sub> O	\ch{H2O}	H <sub>2</sub> O
\ce{[AgCl2]-}	[AgCl <sub>2</sub> ] <sup>-</sup>	\ch{[AgCl2]-}	[AgCl <sub>2</sub> ] <sup>-</sup>
\ce{^227_{90}Th+}	<sup>227</sup> <sub>90</sub> Th <sup>+</sup>	\ch{^227_{90}Th+}	<sup>227</sup> <sub>90</sub> Th <sup>+</sup>
\ce{SO4^2-}	SO <sub>4</sub> <sup>2-</sup>	\ch{SO4^2-}	SO <sub>4</sub> <sup>2-</sup>
\ce{KCr(SO4)2 * 12H2O}	KCr(SO <sub>4</sub> ) <sub>2</sub> · 12 H <sub>2</sub> O	\ch{KCr(SO4)2 * 12 H2O}	KCr(SO <sub>4</sub> ) <sub>2</sub> · 12 H <sub>2</sub> O

However, for almost every setting `chemmacros` offers the possibility of customizing the output. The same input with

```
\chemsetup{phases/pos=sub}
```

gives another very common way of denoting phases:



### 2.3 A lot more

Since describing all of chemmacros' features would be rather pointless (they're all described in the manual) and would also exceed the limits of this article, I'll just give one more example before continuing with the next topic.

```

\ch{
  2 "\OX{o1,Na}" + "\OX{r1,Cl}" {}2
  ->
  2 "\OX{o2,Na}" {}+ + 2 "\OX{r2,Cl}" {}-
}
\redox(o1,o2)[->]{\small OX: \ch{- 2 e-}}
\redox(r1,r2)[->][-1]{\small RED: \ch{+ 2 e-}}

```

$\overbrace{2 \text{Na} + \text{Cl}_2 \longrightarrow 2 \text{Na}^+ + 2 \text{Cl}^-}$ 
  
 OX:  $- 2 e^-$ 
  
 RED:  $+ 2 e^-$

### 3 Molecular formulas and reaction equations—the **mhchem** and **chemformula** packages

Two packages need to be mentioned here: **mhchem** by Martin Hensel and **chemformula** by me. **mhchem** has been around for a longer time and thus is better known and probably has a larger user base.

`chemformula` started as part of the `chemmacros` package in January 2012 and was released as an independent package in July 2013. It is still closely connected with `chemmacros` in that `chemmacros` uses it extensively. Thus, people using `chemmacros` should use `chemformula` instead of `mhchem` in order to have consistent input and output.

Both packages are similar in input and output, but have important differences in both aspects.

Table 2: mhchem's and chemformula's arrows

<b>mhchem</b>		<b>chemformula</b>
\ce{->}	→	\ch{->}
\ce{<-}	←	\ch{<-}
\ce{<->}	↔	\ch{<->}
		\ch{<>}
\ce{<=>}	⊒	\ch{<=>}
\ce{<=>>}	⊑⊒	\ch{<=>>}
\ce{<<=>}	⊑⊒	\ch{<<=>}
		\ch{-/>}
		\ch{</->}
\ce{=}	=	\ch{==}
		\ch{<o>}

Martin Hensel, the author of mhchem, once described the differences as follows [3]:

[chemformula's] philosophy is *control to the user*. [...] mhchem's philosophy, on the other hand, is *ease of use*.

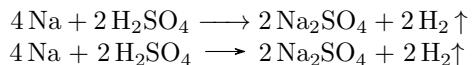
Both packages mainly provide one macro for the typesetting of chemical formulas and reactions:

- mhchem has `\ce{<input>}` and
  - chemformula has `\ch{<input>}`.

Table 1 shows a brief comparison of basic use cases.

The differences are most visible when typesetting reactions:

```
\ce{4 Na + 2 H2S04 -> 2 Na2S04 + 2 H2 ^} \\  
\ch{4 Na + 2 H2S04 -> 2 Na2S04 + 2 H2 ^}
```



You will notice differences in spacing (which is customizable in `chemformula`) and of course the different arrows. Table 2 shows the different kinds of arrows both packages provide.

There is much more to be said about both packages but again I'll leave it with this introduction and refer to the respective manuals for details.

## 4 Structural formulas — the `chemfig` package

Both `mhchem` and `chemformula` lack support for showing the complexities of organic compounds. For typesetting structural (skeletal) formulas, the most common way probably is to use an external program like ChemDraw, export the results as images and include those into the L<sup>A</sup>T<sub>E</sub>X document. However, there are ways to create such formulas and diagrams from within L<sup>A</sup>T<sub>E</sub>X. To my knowledge there are five packages for doing this. People interested in details can have a look at [6]. In this article I'll cover only one of them: `chemfig` by Christian Tellechea. Again this will be a rather brief introduction—the manual has more than 80 pages.

First of all: *chemfig* is a generic package (it's the only one such among the packages described in this article); it can be used in L<sup>A</sup>T<sub>E</sub>X, Con<sup>T</sup>EXt and plain T<sub>E</sub>X.

```
\input chemfig.tex % plain TeX  
\usepackage{chemfig} % LaTeX  
\usemodule[chemfig] % ConTeXt
```

`chemfig` uses PGF (TikZ) for drawing the formulas. The most important command is

\chemfig{<input>}.

The basic rules are simple: letters are atoms and bonds are input as `-`, `=`, etc. Atoms are typeset in math mode (more precisely with `\printatom` which expands to `\ensuremath{\mathrm{\#1}}`) so subscripts can be input as in math:

\chemfig{H-CH\_3}



There are a number of different bond types, shown in table 3. The appearance can be customized with a number of parameters. All bonds have an optional argument taking a comma-separated list of five parameters, in this order:

1. the angle of the bond,
  2. a scale factor,
  3. the “departure” atom number,
  4. the “arrival” atom number, and
  5. TikZ options for customization of the bond.

The angle can be input in three different ways:

- $\langle \text{integer} \rangle$  — denotes an angle in multiples of 45°.
  - $: \langle \text{real} \rangle$  — denotes the angle of the bond counter-clockwise (positive) or clockwise (negative) to the horizontal.
  - $:: \langle \text{real} \rangle$  — denotes the angle of the bond counter-clockwise (positive) or clockwise (negative) relative to the bond before

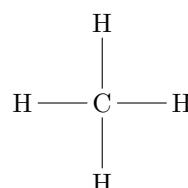
Thus, each of these input lines

**Table 3:** chemfig's bond types.

Input	Output
\chemfig{A-B}	A — B
\chemfig{A=B}	A == B
\chemfig{A^B}	A ≡ B
\chemfig{A>B}	A ▶ B
\chemfig{A<B}	A ◀ B
\chemfig{A>:B}	A    ...  B
\chemfig{A<:B}	A .....   B
\chemfig{A> B}	A ▷ B
\chemfig{A< B}	A ▷◁ B

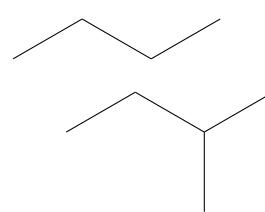
```
\chemfig{H-C(-[2]H)(-[6]H)-H}
\chemfig{H-C(-[:90]H)(-[:-90]H)-H}
\chemfig{H-C(-[:-90]H)(-[:-90]H)-H}
```

gives the same output:



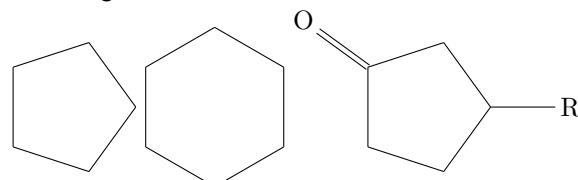
The last example also demonstrates how branching is done: surround the branch with parentheses.

```
\chemfig{:-[::30]-[:-60]-[:60]}\par\bigskip\chemfig{[:-30]-[:-60](-[:-60])-[:60]}
```



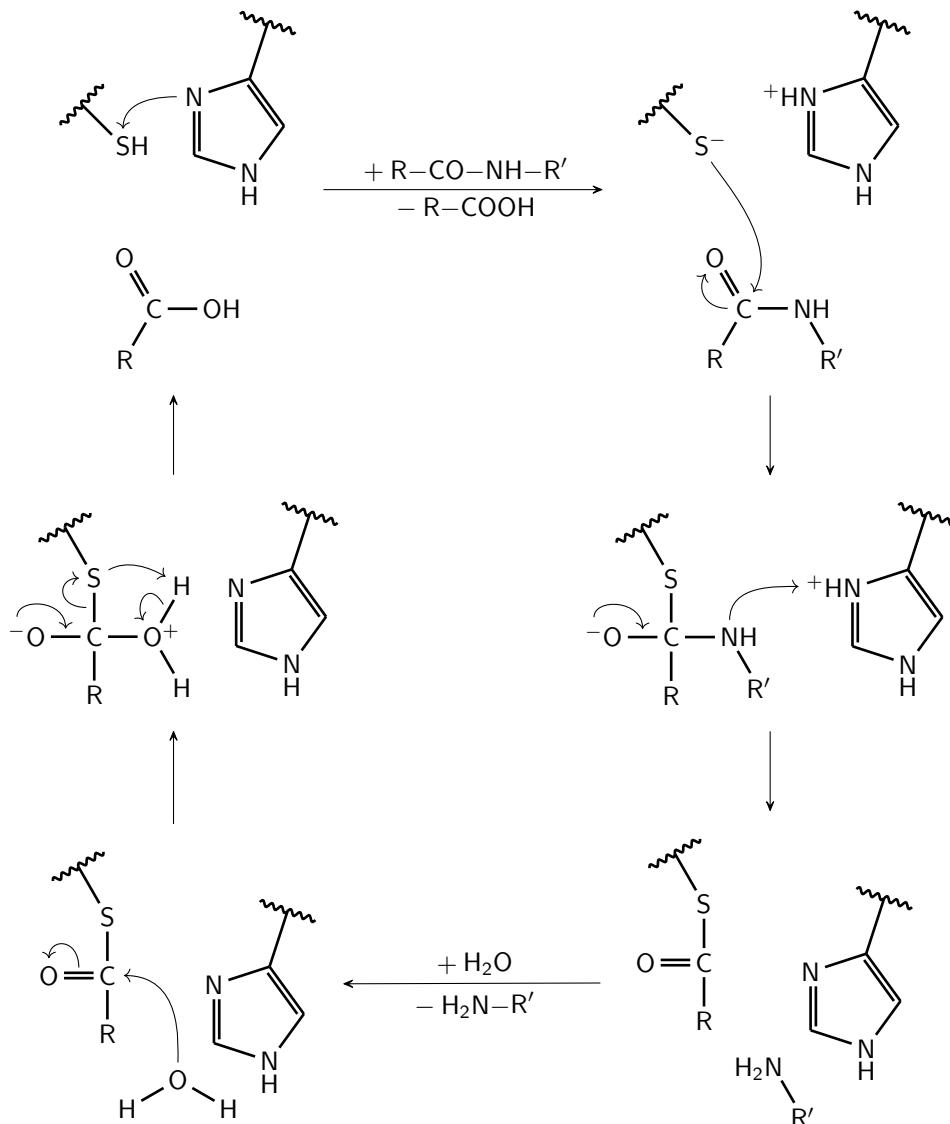
Making rings is also quite easy:

```
\chemfig{*5(-----)}  
\chemfig{*6(-----)}  
\chemfig{*5(--(-R)--(=O)--)}
```



Personally, I find this syntax very intuitive—indeed much more intuitive than some of the other packages mentioned in [6]—and it can be learned very fast.

I hope that the examples so far give a basic idea of chemfig's usage. To conclude this section, figure 1 shows an example of a more complicated scheme created with chemfig, to give you a larger impression of what is doable with the package. The code for the example can be seen in [5].



**Figure 1:** An example of using chemfig to create more complicated diagrams.

## 5 Safety data—the rsphrase, hpstatement, and ghsystem packages

Every student of chemistry knows the safety data phrases coordinated in the *Globally Harmonized System of Classification and Labelling of Chemicals*, in short, GHS. This system defines H (hazard) and P (precaution) statements. Before this became the standard there were R (risk) and S (safety) statements.

The R and S statements are available through the rsphrase package by Martin Hensel:

The statement `\rsnumber{R34}` is '`\rsphrase{R34}`'

The statement R 34 is ‘Causes burns.’

Its usage is described in mhchem’s manual.

The usage of the hpstatement by the same author also is described in mhchem’s manual. The package provides support for the H and P statements of the GHS. It is a bit unfortunate that the package has

`\RequirePackage{babel}`

which forces the user to use the babel package and provide a language option.

The statement `\hpnumber{H200}` is ‘`\hpstatement{H200}`’

The statement H200 is “Unstable explosives.”

There is another package for GHS statements, ghsystem by me, which is a little bit older than hpstatement. It also provides an interface to get the statements per number.

The statement `\ghs[hide]{h}{200}`

```
is ``\għs*{h}{200}’’: \par
\għs{h}{200}
```

The statement H200 is “Unstable explosives.”:  
H200: Unstable explosives.

In addition it provides the possibility of inserting the GHS pictograms.

```
\għspic{skull}
\għspic{flame}
\għspic{aqpol}
```

Those pictograms are available as JPG, EPS, PNG, and PDF. There are plans to eventually provide the pictograms as TikZ pictures.

## 6 Numbering compounds — the **chemnum** package

It is a common task for chemists to refer to compounds with numbers. There are a number of packages for this task but the most flexible one is **chemnum** (again by me).

The basic usage is simple: add `\cmpd{<id>}` where you want the number:

```
This is about \ch{CH3-CH2-OH} (\cmpd{ethanol}).  
\cmpd{ethanol} is used\ldots
```

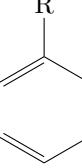
This is about CH<sub>3</sub>—CH<sub>2</sub>—OH (1). 1 is used...

Upon the first usage of ID `<id>` the command will create a new number for the compound. Later uses will then refer to this number. For usage in section titles or captions a version with optional +(`\cmpd+{<id>}`) can be used to refer to the compound without creating the number. There is also a starred version which will create the number but prints nothing. In this respect the syntax is the same as in the older package **chemcompounds** by Stephan Schenck.

**chemnum** also natively supports “sub-labelling”: `\cmpd{<id>}.<sub_id>}`.

Both `<id>` and `<sub_id>` may be comma-separated lists. If `<sub_id>` is a list it *must* be enclosed in braces.

```
\setatomsep{2em}
\chemname
  {\chemfig{*6(==-(R)==)}}
  {\cmpd{benzene.{H,Me,OH,NH2}}}
\begin{tabular}{ll}
& \ch{-R} & Name \\
\midrule
\cmpd[sub-only]{benzene.H} & \ch{-H} & Benzene \\
& \ch{-Me} & Toluene \\
\cmpd[sub-only]{benzene.OH} & \ch{-OH} & Phenol \\
& \ch{-NH2} & Phenylamine (Aniline)
\end{tabular}
```

	−R	Name
<b>a</b>	−H	Benzene
<b>b</b>	−CH <sub>3</sub>	Toluene
<b>c</b>	−OH	Phenol
<b>d</b>	−NH <sub>2</sub>	Phenylamine (Aniline)

This example shows that by default lists of sub-labels are compressed. This can be turned off with an option.

```
\cmpd{benzene.{H,Me,OH,NH2}}
\setchemnum{compress=false}
\cmpd{benzene.{H,Me,OH,NH2}}
```

### 2a–d 2a,b,c,d

When you have a list of labels which contains an `<id>` more than once but with different sublabels, each entry will be printed on its own:

```
\cmpd{a,b.{A},c,b.{B,C},a}
```

### 3, 4a, 5 and 4b,c

However, there is an option to merge those different entries into one:

```
\setchemnum{merge=true}
\cmpd{a,b.{A},c,b.{B,C},a}
```

### 3, 4a–c and 5

What we have not seen yet in the above examples is that lists are sorted as well:

```
\cmpd{a,b,c,b,a,ethanol,benzene} vs. \@\@{a}{b}{c}{b}{a}{ethanol}{benzene}
\cmpd{a}, \cmpd{b}, \cmpd{c}, \cmpd{b}, \cmpd{a}, \cmpd{ethanol} and \cmpd{benzene}
```

**1, 2, 3, 4 and 5** vs. **3, 4, 5, 4, 3, 1 and 2**

## 7 Packages for special applications

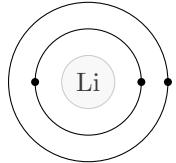
The number of packages available for support of chemistry-related topics is quite large. Both CTAN (<http://ctan.org/topic/chemistry>) and I, on my blog (<http://mychemistry.eu/known-packages/>), maintain a list of packages related to chemistry. The lists differ slightly, mostly because I also include fringe cases, e.g., packages which may belong more to biology than to chemistry. In this section I pick a few of my packages and will present them extremely briefly:

- bohr
- elements
- endiagram
- modiagram
- chemgreek

## 7.1 The bohr package

The package `bohr` was created as an answer to a question on <http://tex.stackexchange.com> [1]. It provides a simple way of drawing atomic orbitals according to the Bohr model:

```
\bohr{3}{Li}
```



## 7.2 The elements package

The `elements` package is rather new (released in June 2015). From its abstract:

This package provides means for retrieving properties of chemical elements like atomic number, element symbol, element name, electron distribution or isotope number. Properties are defined for the elements up to the atomic number 112.

The following example gives a short impression of its capabilities:

```
\elementname{Cu} \\
\elementname{11} \\
\atomicnumber{U} \\
\elconf{Cl} \\
\savemainelementisotope\foo{C}\foo
```

Copper

Sodium

92

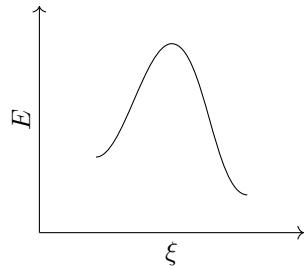
$1s^2 2s^2 2p^6 3s^2 3p^5$

12

## 7.3 The endiagram package

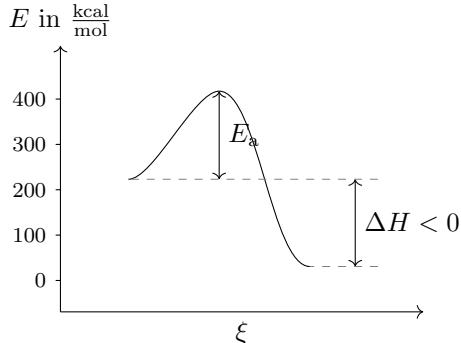
The `endiagram` package lets you create potential energy curve diagrams.

```
\begin{endiagram}
\ENcurve{1,4,0}
\end{endiagram}
```



A more advanced example:

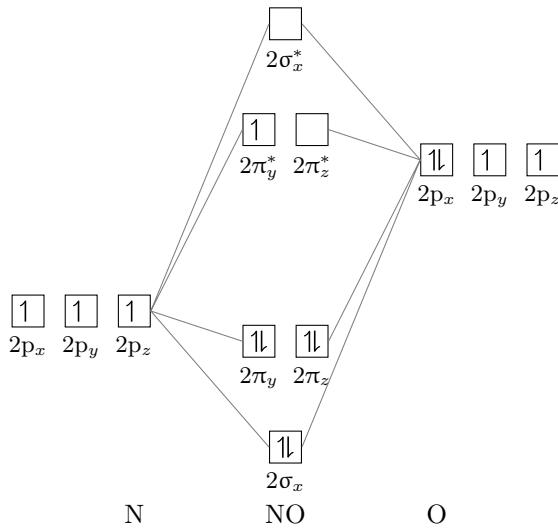
```
% preamble:
% uses 'siunitx' (loaded by 'endiagram')
\DeclareSIUnit{\calory}{cal}
\sisetup{per-mode = fraction}
% document:
\ENsetup{
    y-label = above ,
    energy-step = 100 ,
    energy-unit = \kilo\calory\per\mole ,
    energy-unit-separator = { in } ,
    calculate = false ,
    AddAxisLabel/font = \footnotesize
}
\begin{endiagram}[scale=1.2]
\ENcurve{2.232,4.174,.308}
\AddAxisLabel*[0;1;2;3;4]
>ShowEa[label,connect={draw=none}]
>ShowGain[label]
\end{endiagram}
```



## 7.4 The modiagram package

Like the `bohr` package, the `modiagram` package was created as an answer to a question on <http://tex.stackexchange.com> [7]. It offers an interface for drawing molecular orbital diagrams based on `TikZ`.

```
\begin{M0diagram}[labels,names,style=square]
\atom[N]{left}[
    2p = {0;up,up,up}
]
\atom[0]{right}[
    2p = {2;pair,up,up}
]
\molecule[NO]{
    2pMO = {1.8,.4;pair,pair,pair,up}
}
\end{M0diagram}
```



## 7.5 The chemgreek package

The `chemgreek` package is a support package rather than one to be used by users directly. At the time of writing, the packages `chemmacros`, `chemnum` and `mhchem` make use of its features. The purpose of `chemgreek` is to provide a unified interface for upright Greek letters regardless of which math or font package provides upright letters and regardless of how those are accessed. This is important in chemistry since upright Greek letters are used in a variety of places:

```
\iupac{\alpha-\text{D-glucopyranose}}
```

$\alpha$ -D-glucopyranose

The `chemgreek` package knows about a number of packages providing upright Greek letters. It is able to detect if one of those packages is loaded, and if a unique choice is possible it defines macros for each of the 24 lowercase and uppercase letters. If no unique choice is possible it falls back to a default mapping and users have to make a choice themselves. The `chemmacros` package has a user interface for this:

```
\chemsetup{greek = {mapping}}.
```

Other packages can now use those macros to define macros of their own.

If a document needs a font package with upright Greek letters which `chemgreek` doesn't know about, users have the capability to define a new `(mapping)` themselves, and (for example) activate it with `chemmacros'` interface, and then use `chemmacros` nomenclature commands using the new mapping.

## 8 Summary

This article briefly discussed the usage of the packages `chemmacros`, `mhchem`, `chemformula`, `chemfig`, `rsphrase`, `hpstatement`, `ghsystem`, `chemnum`, `bohr`, `elements`, `endiagram`, `modiagram`, and `chemgreek`. It shows that today there is considerable support for different typesetting tasks in chemistry and gives a short overview of some the existing possibilities.

## References

- [1] Andreas. Draw Bohr atomic model with electron shells in TeX? <http://tex.stackexchange.com/questions/73410/> [accessed 2015-08-29].
- [2] E. Richard Cohan, Tomislav Cvitaš, Jeremy G. Frey, Bertil Holmström, Kozo Kuchitsu, Roberto Marquardt, Ian Mills, Franco Pavese, Martin Quack, Jürgen Stohner, Herbert L. Strauss, Michio Takami, and Anders J. Thor. “Quantities, Symbols and Units in Physical Chemistry”, *IUPAC Green Book*. IUPAC & RSC Publishing, Cambridge, 3rd edition, 2nd printing, 2008.
- [3] Martin Hensel. On mhchem vs. chemformula. [http://tex.stackexchange.com/q/237946#comment565361\\_238214](http://tex.stackexchange.com/q/237946#comment565361_238214) [accessed 2015-08-29].
- [4] Clemens Niederberger. a new `chemmacros`—but how? <http://www.mychemistry.eu/2015/07/a-new-chemmacros-but-how/> [accessed 2015-08-29].
- [5] Clemens Niederberger. A seemingly complex example for the usage of `chemfig`. <https://www.overleaf.com/read/nhbyvqvrqffj> [accessed 2015-08-29].
- [6] rake. Can you make chemical structure diagrams in LATEX? <http://tex.stackexchange.com/questions/52722> [accessed 2015-08-29].
- [7] Richard Terrett. Molecular orbital diagrams in LATEX? <http://tex.stackexchange.com/questions/13863> [accessed 2015-08-29].

◊ Clemens Niederberger  
Am Burgrain 3  
71083 Herrenberg  
Germany  
contact (at) mychemistry dot eu  
<http://www.mychemistry.eu/>

## Automating L<sup>A</sup>T<sub>E</sub>X(3) testing

Joseph Wright and L<sup>A</sup>T<sub>E</sub>X3 team

### 1 Introduction

Testing has always been an important part of the work of the L<sup>A</sup>T<sub>E</sub>X team. Over the last couple of years, ideas first developed in the early 1990s have been used to create a flexible testing tool, l3build (Mittelbach, Robertson, and L<sup>A</sup>T<sub>E</sub>X3 team, 2014; L<sup>A</sup>T<sub>E</sub>X3 team, 2015). In an ideal world, every change would be checked by a full run of the entire test suite. In reality, that's not always the case: mistakes happen. What's therefore needed is to automate testing every time a change is made.

These concerns are not unique to the (L<sup>A</sup>)T<sub>E</sub>X world, and thus it's no surprise that a variety of automated code testing approaches are already available. It can be done using a 'real' local machine, a private virtual machine or, increasingly commonly, a hosted solution. There are many companies now offering automated remote testing on virtual machines, and as this means a minimal amount of setup, it's this approach that the L<sup>A</sup>T<sub>E</sub>X team have been exploring.

Automating testing is part of a wider concept called 'continuous integration', often referred to as CI. As with many ideas in code development, this tends to attract a lot of acronyms and odd tool names: I'll try to keep those to a minimum here.

### 2 Setting up

For open source projects like L<sup>A</sup>T<sub>E</sub>X3, many of the providers of hosted services offer free accounts. L<sup>A</sup>T<sub>E</sub>X has chosen to use Travis-CI (<http://travis-ci.org>), partly as it is well-known, partly as it is easy to set up, and partly as it fits into other parts of our setup (see below). Of course, there are many other worthy choices.

The two key things any automated test system has to know: where to find the code that is changing and how to run the required tests. Travis-CI integrates with GitHub (<http://github.com>), one of the large number of websites offering hosting for version control using Git (Git team, 2015). The team have had the GitHub site for some time (<http://github.com/latex3/latex3>), so integrating with Travis-CI was easy.

Setting up the testing itself means telling Travis-CI about the type of code being tested. This is done using a plain text file, `.travis.yml`, which has to be in the main directory of the code repository. For common programming languages, such as C, Java or Ruby, Travis-CI knows the normal testing setups and likely only needs to be told the language involved.

For T<sub>E</sub>X, that's not the case: we (I) need to give the system more information. Moreover, the virtual machine setup used by Travis-CI doesn't have a T<sub>E</sub>X system installed as standard. So there is a bit of work to do, but luckily in a well-documented and simple format.

Setting up the actual tests means pointing the system at l3build, which can be done in two lines:

`script:`

```
- texlua build.lua check -H
```

This will run in the main directory of the code repository, and will run all of the current tests for L<sup>A</sup>T<sub>E</sub>X3. As the test is, ultimately, just pass/fail, I've told l3build to halt immediately if any test fails (-H).

To get a T<sub>E</sub>X system installed, I need to run a script, which will clearly depend on the nature of the virtual machine. For Travis-CI, that's currently Ubuntu 12.10LTS, so I use a `bash` script. For the test system, I need to make sure that script gets run before our tests; this is what Travis-CI calls the `install` step:

`install:`

```
- source ./support/texlive.sh
```

The script itself needs to run an automated T<sub>E</sub>X (Live) installation. That has two parts: first downloading and installing a minimal system, then adding on extra packages that I need. (We'll see later why a small system is useful.) The script is relatively straightforward:

```
# Obtain TeX Live
wget http://mirror.ctan.org/systems/\texlive/tlnet/install-tl-unx.tar.gz
tar -xzf install-tl-unx.tar.gz
cd install-tl-20*

# Install a minimal system
./install-tl \
--profile=../support/texlive.profile

# Add the TL system to the PATH
PATH=/tmp/texlive/bin/x86_64-linux:$PATH
export PATH

cd ..

# Core requirements for the test system
tlmgr install babel babel-english \
    latex latex-bin latex-fonts \
    latexconfig xetex
tlmgr install --no-depends ptxuptex
```

The run of `install-tl` above uses a so-called profile file to tell the T<sub>E</sub>X Live installer what to do. That's again quite short:

```
# Profile for minimal TeX Live installation
selected_scheme scheme-minimal
TEXDIR /tmp/texlive
TEXMFCONFIG ~/.texlive2015/texmf-config
TEXMFHOME ~/texmf
TEXMFLOCAL /tmp/texlive/texmf-local
TEXMFSYS CONFIG /tmp/texlive/texmf-config
TEXMFSYS VAR /tmp/texlive/texmf-var
TEXMFVAR ~/.texlive2015/texmf-var
option_doc 0
option_src 0
```

Here, we do not install the sources or documentation (clearly not needed for runtime testing) and the installation location is non-standard: I don't have `sudo` on the virtual machine and the profile installation doesn't (yet) support `~` (the home folder) in the installation path.

You might wonder if I could have used `apt-get` to add the Ubuntu managed TeX Live. That runs, and means I wouldn't need a script (`.travis.yml` has an entry type for running `apt-get`). However, it installs TeX Live 2009, which is too old to run `l3build`. (There have been a lot of changes in LuaTeX since then.) For the team tests, we always assume an up-to-date and current TeX Live, so it makes sense to have the same on the Travis-CI setup.

You might also wonder how I worked out exactly what the minimal requirements were for the installation. That was a bit of work, but the idea was simple enough: run the tests on a local virtual machine and add packages one at a time until everything works!

### 3 Refining

Once the above was set up, Travis-CI started running automated tests each time changes were made to the GitHub version of the `LATEX3` code. There were of course a few teething issues: it turned out that `l3build` was returning error level 0 ('success') even when tests failed! That was soon fixed in our code: a first demonstration of the use of automated testing.

With a virtual machine, each time tests are run the machine is 'reset' to a known state. That meant that each code change needed to do a fresh install of TeX Live: one of the reasons for keeping the system small. Ideally, I wanted to avoid the load on CTAN if possible. To do that, we've added *caching* to our `.travis.yml` file:

```
cache:
  directories:
    - /tmp/texlive
    - $HOME/.texlive2015
```

This compresses the directories listed at the end of each test run, then adds them to the 'clean' machine

at the start of the next run.

Caching gives us a way to make sure a TeX system is available, but what about when the cache has to be reset, when new packages are needed or when updates are available? A bit of `bash` scripting sorts all of that. First, the basic installation can be wrapped up in a test looking for a TeX system:

```
# See if there is a cached version of TL
PATH=/tmp/texlive/bin/x86_64-linux:$PATH
export PATH
if ! command -v texlua >/dev/null; then
  # Earlier script code
fi
```

We can then run the update process:

```
# Keep no backups (makes cache smaller)
tlmgr option autobackup 0
# Update the TL install
tlmgr update --self --all \
  --no-auto-install
```

and finally add extra packages

```
tlmgr install \
  adobemapping \
  amsmath \
  ...
```

With this approach, the list of extra packages can keep growing, and any new entries will get added automatically. If the cache has to be cleared, an entirely new TeX Live and all of the required packages will be installed.

With the standard settings, Travis-CI emails the person who made code changes that led to the tests failing. For the `LATEX3` source repository, we have a mailing list for every commit, so it makes sense to send those failure messages to the list too, which is done like this:

```
notifications:
  email:
    recipients:
      - latex3-commits@tug.org
    on_success: change
    on_failure: always
    on_start: never
```

### 4 In use

Setting up all of the above took only a few days, and much of that was working out how best to install a TeX Live setup automatically and then to cache it. The actual `.travis.yml` configuration took only minutes to do.

Running the full test suite for `LATEX3` currently takes around 6 minutes on the virtual machine, depending on whether the TeX Live system needs to be

```

2
3 Build system information
65
66 $ git clone --depth=50 --branch=master https://github.com/l
76
77 This job is running on container-based infrastructure, whic
78 If you require sudo, add 'sudo: required' to your .travis.y
79 See http://docs.travis-ci.com/user/workers/container-based-
80 Setting up build cache
81 $ rvm use default
82 $ ruby --version
83 ruby 1.9.3p551 (2014-11-13 revision 48407) [x86_64-linux]
84 $ rvm --version
85 rvm 1.26.10 (latest-minor) by Wayne E. Seguin <wayneeseguin@r
86 $ bundle --version
87 Bundler version 1.7.6
88 $ gem --version
89 2.4.5
90 $ source ./support/texlive.sh
161 $ texlua build.lua check -H
162 This is TeX, Version 3.14159265 (TeX Live 2015) (preloaded
163 ./l3build.ins (./local/docstrip.tex
164 Utility: 'docstrip' 2.5d <2005/07/29>
165 English documentation <1999/03/31>
166
167 ****
168 * This program converts documented macro-files into fast *
169 * loadable files by stripping off (nearly) all comments! *
170 ****
171
172 ****
173 * No Configuration file found, using default settings. *
174 ****
175
176 )
177
178 Generating file(s) regression-test.tex
179
180 Processing file l3build.dtx (package) -> regression-test.t
181 Lines processed: 1363
182 Comments removed: 986
183 Comments passed: 50
184 Codelines passed: 323
185
186 )
187 No pages of output.
188 Transcript written on l3build.log.
189 This is TeX, Version 3.14159265 (TeX Live 2015) (preloaded
190 ./01-expect.ins (./local/docstrip.tex
191 Utility: 'docstrip' 2.5d <2005/07/29>

```

**Figure 1:** Log output from Travis CI (abridged).

re-installed. That's about the same time as it takes on a MacBook Pro i7 (my own laptop). So checking the code in almost real-time is certainly workable.

Most of the time the tests pass, and the web page shows a simple report to this effect. When a

test fails, as well as the email sent, the web page shows the failure. Notice that in both cases we get the commit reference, which we can use to go straight to the code changes on GitHub. There is also an overview of changes over time, so you can quickly get a feel for what broke and fix the system.

Sometimes of course you need more detail, and for that the terminal log is visible (Figure 1). As you can see, each phase of the process is separated out so you can collapse parts that are not important: for example, if the `install` phase is fine but there is a problem with the tests. The log loads in real time when a test is running, so you can monitor what's happening and if necessary kill a test, for example if something seems to have hung.

That flexibility and speed means it's been possible to add more tests, checking on how the core L<sup>A</sup>T<sub>E</sub>X code interacts with some contributed packages.

## 5 Conclusions

Setting up and running an automated test system using a hosted virtual machine makes running tests on every change easy. It shouldn't be seen as an alternative to running tests *before* changing code, but it is another tool to exploit in keeping ahead of the inevitable bugs.

## References

- Git team. <http://git-scm.com>, 2015.
- L<sup>A</sup>T<sub>E</sub>X3 team. “The l3build package”. Available on CTAN: <http://ctan.org/pkg/l3build>, 2015.
- Mittelbach, Frank, W. Robertson, and L<sup>A</sup>T<sub>E</sub>X3 team. “l3build—A modern Lua test suite for TeX programming”. *TUGboat* 35(3), 287–293, 2014. <http://tug.org/TUGboat/tb35-3/tb111mitt-l3build.pdf>.

- ◊ Joseph Wright  
Morning Star  
2, Dowthorpe End  
Earls Barton  
Northampton NN6 0NH  
United Kingdom  
`joseph dot wright (at)`  
`morningstar2.co.uk`
- ◊ L<sup>A</sup>T<sub>E</sub>X3 team  
[www.latex-project.org](http://www.latex-project.org)

## Two applications of SWIGLIB: GraphicsMagick and Ghostscript

Luigi Scarso

### Abstract

We present two applications of SWIGLIB: a binding to the GraphicsMagick library that under certain conditions can speed up conversion of bitmaps by up to 20% and a binding to the Ghostscript library that simplifies the integration of PostScript programs in ConTeXt with the LuaTeX engine. Examples of TIFF conversion and barcodes in PostScript are shown.

### 1 Introduction

In a previous paper [3], we introduced the SWIGLIB project as a way to add (or extend) functionality in LuaTeX by means of an external binary module. Among the several modules available from the SWIGLIB project site (<https://swiglib.foundry.supelec.fr>), two are directly related to the management of images: `graphicsmagick` for bitmaps, and `ghostscript` for PostScript files. These are the libraries underneath the `gm convert` and `gs` programs, respectively, and in ConTeXt they are used to convert BMP, GIF, TIFF and EPS to PDF.

The conversion is quite simple: the file is saved as PDF, which is subsequently used instead of the original one. In a multi-pass run the conversion happens only the first time, and ConTeXt takes care of keeping the original file and the PDF in sync.

This happens for each file independently, and therefore  $n$  TIFF (for example) files require  $n$  calls to the external program `gm convert`, and we can measure the time of each call as the sum of two times, *setup and close* and *conversion* with mean  $t_s$  and  $t_c$ . Assuming that  $n$  conversions with a module take only one  $t_s$ , the ratio  $s = n(t_s + t_c)/(t_s + nt_c)$  is the “speedup” of the module: for  $n$  great enough such that  $t_s/n$  is negligible with respect to  $t_c$ , the speedup is with good approximation  $1 + t_s/t_c$ . Situations where  $t_s \geq t_c$  means that at least half the time is “wasted” in setup: the program is not very efficient, or more likely it’s not the right fit for the current task. On the other hand,  $t_s/t_c \approx 0$  means that the files take so much time to convert that it’s more robust to use the external program, e.g., to minimize the risk of memory leaks and as protection against crashes.

So, it’s reasonable to expect that  $0.1 \leq t_s/t_c \leq 0.4$ , or  $1.1 \leq s \leq 1.4$ . Let’s emphasize that these figures are valid when each run has a number of conversions  $n$  high enough to make  $t_s/n$  negligible (for example,  $n \geq 100$ ) and each file takes approximately the same time to be converted, conditions

that are fairly likely to be satisfied in servers with automatic workflows: in other cases, any speedup could be irrelevant.

The format used is ConTeXt, which already has a caching system for conversions (more on this at the end of the next section); the measurements were done on a laptop with an Intel Core i7-3610QM CPU @ 2.30GHz quad-core using 8GB memory and a Crucial\_CT512MX1 SSD disk of 512GB.

### 2 The `gm` module

The module for the GraphicsMagick library is probably the most interesting currently available, due to its high number of formats available for conversion, although many of them, for example the PS and EPS formats, require an external program to work and therefore there is no significant gain in speedup. Apart from the PNG and JPEG formats, which are already supported in LuaTeX, the most notable are TIFF, due its use in the printing industry and GIF, which still sees application on web pages. Also of some interest are MIFF and MVG, the bitmap and vector native format of GraphicsMagick, and the set of “portable bitmap” formats such as PNM, PAM and PPM; these can be used to build a portable bitmap image programmatically.

A question arises immediately: why not use the functions from the `epdf` library which is already embedded in LuaTeX? The answer is that a converter returns a complete PDF document as stream (i.e., a sequence of bytes and its length) which the `epdf` library doesn’t know how to manage. Of course it is possible to save the stream into an external file and load it again into memory, and until recently this was the only solution—that is, until the latest release of the `poppler` library, which offers the new `MemStream` function that is tailored exactly for this case, avoiding the expensive task of saving and reloading from a file. A binding to `MemStream` was therefore added to the `epdf` library as `openMemStream`.

Unfortunately, this is only half of the story. While `openMemStream` uses `char *` for the bytes and `long long` for the length of the stream,<sup>1</sup> it is not known in advance how the converter returns the stream. In GraphicsMagick, the conversion in memory is implemented by `MagickWriteImageBlob`:

```
unsigned char *MagickWriteImageBlob (
    MagickWand *wand,
    size_t *length);
```

while on the Lua side the `unsigned char *` (the bytes) is seen as a generic userdata object and not a string, as required by `openMemStream`, and the

<sup>1</sup> Probably `unsigned char*` and `size_t` would be more appropriate.

`length` (the length of the stream) is used as an input parameter, not set as an output parameter (!). It is therefore necessary to have an *adapter*, i.e., a software layer that translates from the converter to `openMemStream`. This could be provided by a third user module or, as in this case, by means of the `helper` module, which can be seen as a kind of “general adapter”—with the limitation that it partially covers only primitive types.

The code, omitting checks for the sake of simplicity, looks like this:

```
local l = -1
local _l = helpers.new_size_t_array(1)
gm.MagickSetImageFormat(wand, "PDF")
local s = gm.MagickWriteImageBlob(wand, _l)
l = helpers.size_t_array_getitem(_l, 0)
helpers.delete_size_t_array(_l)
local _s =
    helpers userdata_to_lightuserdata_uchar_p(s)
local doc, doc_id, doc_uri =
    epdf.openMemStream(_s, l, stream_id)
```

On the Lua side, that final call,

```
epdf.openMemStream(_s, l, stream_id)
```

requires a userdata `s` that is a so-called “light” userdata (i.e., intended to store a C pointer) and must point to a valid memory region of size 1 bytes; the parameter `stream_id` is given by the user to identify the stream and during a given run this identifier must be unique (else the behavior is undefined).

If, in some way, the user converts the stream in a Lua string `s` (taking care of embedded zeroes)<sup>2</sup> then it’s still possible to call

```
openMemStream(s,s:len(),stream_id)
```

which can be eventually wrapped as

```
function openStringStream(s,stream_id)
    return openMemStream(s,s:len(),stream_id)
end
```

If there are no errors, `openMemStream` returns the `doc_id` used to identify the stream at the TeX level; this has the same role as the filename of the PDF figures. Of course, the end user doesn’t need to know these details. Usually, two macros are enough: `\gmloadimage` to load a file, and `\gmloadimage` to return the `doc_id`. In ConTeXt:

```
\gmloadimage{a.tiff}
\externalfigure[\gmloadimage{a.tiff}]
```

If the file contains multiple images:

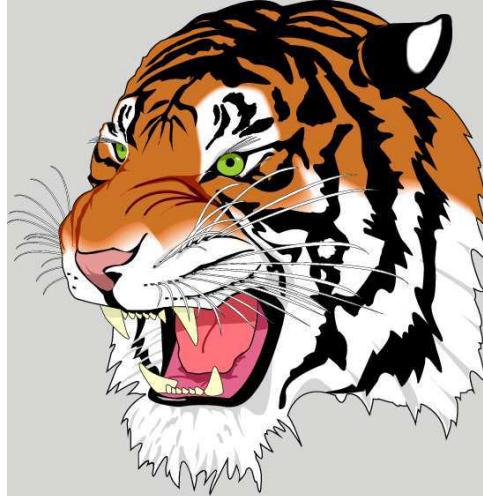
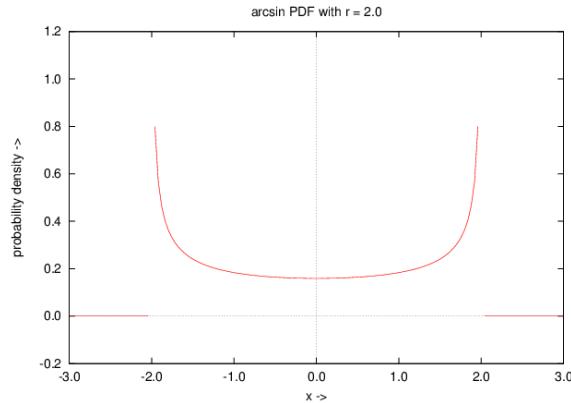
```
\gmloadimage{a.tiff}
\externalfigure[\gmloadimage{a.tiff}] [page=1]
```

<sup>2</sup> A converter that returns a stream as `char *` is wrapped by SWIG using `lua_pushstring`, returning the stream *until the first '\0'*, which is excluded. Since a valid PDF document can contain an arbitrary number of '\0's, this kind of converter must be wrapped by the user in the correct way—for example, using `lua_pushlstring`.

```
\externalfigure[\gmloadimage{a.tiff}] [page=2]
\externalfigure[\gmloadimage{a.tiff}] [page=3]
```

TIFF is not the only possible format. For example, if the library includes the support for calling Gnuplot and Ghostscript as external programs, their formats are valid too:

```
\usemodule[gm]
\starttext \startTEXpage
\gminit{}
%
\gmloadimage{prob-3.gplt}
\externalfigure[\gmloadimage{prob-3.gplt}]
%
\gmloadimage{tiger.eps}
\externalfigure[\gmloadimage{tiger.eps}]
\stopTEXpage \stoptext
```



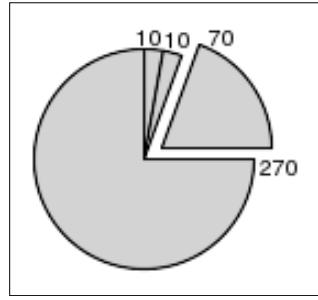
With the MVG native format and a bit of Lua, it is also possible to create a PDF at runtime:

```
\usemodule[gm]
\starttext \startTEXpage
\framed{%
\startluacode
local res
local blob = ""
local gm = moduledata.swiglib.graphicsmagick
gm.init('..')}
```

```

local report_state = gm.report_state
blob =
[=[
push graphic-context
viewbox 0 0 140 130
stroke black
fill lightgray
path 'M 60,70 L 60,20 A 50,50 0 0,1 68.7,20.8 Z'
path 'M 60,70 L 68.7,20.8 A 50,50 0 0,1 77.1,23 Z'
path 'M 68,65 L 85.1,18.0 A 50,50 0 0,1 118,65 Z'
path 'M 60,70 L 110,70 A 50,50 0 1,1 60,20 Z'
stroke none fill black
font-size 10
text 57,19 '10' text 70,20 '10'
text 90,19 '70' text 113,78 '270'
path 'M700.0,600.0 L340.0,600.0 A360.0,360.0 0 0,1
        408.1452123287954,389.2376150414973 z'
pop graphic-context
]=]
local name = 'myblob'
if not(gm.formats['MVG']) then
  report_state("ERROR: MVG FORMAT UNKNOWN")
  return false
end
res,name = gm.blobimage(blob,'MVG',name)
if (res == 0) then
  report_state("ERROR ON BLOB IMAGE")
  return false
end
res = gm.register(name)
if (res == 0) then
  report_state("ERROR ON REGISTERING BLOB IMAGE")
  return false
end
context.externalfigure( {gm.Images[name].doc_id},
                       {width='10cm'} )
\stopluacode\stopTEXpage \stoptext

```



Let's now consider this important point: ConTeXt is a multipass system, storing the results of one pass for the next run in an external file. The same happens for conversion to PDF (i.e., *caching* of the PDF), so that in practice only the first run has the hard task: if a job requires only one run, the cached PDFs are useless and can be deleted saving space, but the time to write them to disk and read them again is lost. Caching is also possible in `gm`, but can be avoided if it is known in advance that the job is one-pass, thus saving both space on disk and the time to write/read. A first measure of the times for a file that loads 100 TIFF of size 500×500 at 300 dpi shows that the standard one-pass conversion takes  $t_i =$

10.94 s, while for `gm` *without caching* of the PDF,  $t_f = 8.52$  s. The gain is therefore  $|t_f - t_i| / t_i \cdot 100 = 22\%$  with speedup  $s = 1.28$ . Things change drastically when we look at a standard multipass run: enabling the caching in `gm` reduces the gain to a value between 6% and 7%.

### 3 The gs module

The module for the Ghostscript library poses a challenge similar to GraphicsMagick: one instance for many conversions. Unfortunately, this library still lacks a clear method to save the PDF in memory and `epdf.openMemStream` is of no help here—each PDF must be saved in an external file and then loaded again. On the other hand, PostScript is not a binary format, and a Lua string is adequate in most cases.

One of the most common uses is the conversion from EPS or PS to PDF:

```

\usemodule[gs]
\starttext \gsinit
%
\gsrunfile{tiger.eps}\gsflush
\externalfigure[tiger.pdf]
%
\gsrunfile{colorcir.ps}\gsflush
\externalfigure[colorcir.pdf]
\stoptext

```

where `\gsflush` closes the output file. There is only one instance and with `\gsrunonce` the instance is also reinitialized after the conversion:

```

\usemodule[gs]
\starttext \gsinit
\gsrunonce[pstopdf,
-dNOPAUSE,
-dBATCH,
-dSAFER,
-sDEVICE=pdfwrite,
-sOutputFile=tiger1.pdf,
-c,.setpdfwrite,
-f,
tiger.eps]
\externalfigure[tiger1.pdf]
\stoptext

```

Converting a buffer is also immediate:

```

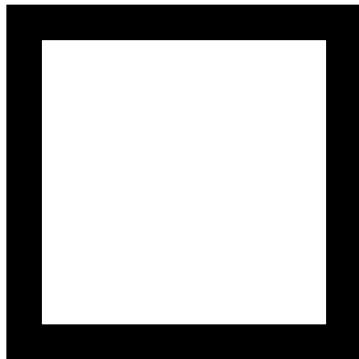
\usemodule[gs]
\starttext \startTEXpage
\gsinit
\startluacode
local psbuf = [=[%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 5 5 105 105
10 setlinewidth
10 10 moveto
0 90 rlineto 90 0 rlineto
0 -90 rlineto closepath
stroke
]==]

```

```

local gs = moduledata.swiglib.ghostscript
gs.Run_buffer(pdbuf, 'mybuf.pdf')
\stopluacode
\gsflush
\externalfigure[mybuf.pdf]
\stopTEXpage \stoptext

```



### 3.1 barcode.ps

A nice application is `barcode.ps`, a widely used PostScript program that supports a huge number of barcodes (see [1]). The program in figures 1 and 2 takes advantage of both the binding and the Lua language:

- it loads and executes `barcode.ps` only once, saving time in a multipass run;  
`(% Load barcode.ps)`
- it saves the barcode in PDF format, storing the filename in a persistent database. This means that only the first run calls the interpreter, while the others load the PDF already produced;  
`(% Make a barcode, save it as pdf and store the name in the table barcode/pdffile);`
- the logic is in Lua—the macro `\gmpsbarcode` calls directly a Lua function and returns the name of the relative PDF.  
`(% bridge TeX--->Lua and %% User macro).`

As mentioned above, caching a PDF for later use is a common practice in ConTeXt but usually the program that produces a barcode is called for each single barcode (i.e.,  $n$  barcodes take  $n(t_s + t_c)$ ), while in this case the program is called only one time ( $n$  barcodes take  $t_s + nt_c$ ). The time of setup  $t_s$  can be important, given that the size of `barcode.ps` file is 723KB, which is loaded every time in the first case. For this reason the distribution at [1] also provides a single file for each barcode. (Ghostscript also currently suffers from suboptimal garbage collection. In case of problems, the collector can be partially disabled with an initial `-dNOGC` option.)

## 4 Conclusions

The module `gm` shows its full potential in a precise context: a single run with many conversions. Typically this is an automatic workflow with minimal typographical requirements and oriented to mass production of documents; for example, a variable-data printing workflow, probably also tuned to reduce the times of reading/writing to file. In this situation the gain could be a time reduction of 20% without increasing the space on disk. On the other hand, for the common single run situation, the gain is negligible and the standard conversion is the better choice.

The module `gs` is interesting not so much for the performance (which in any case is no worse than the standard conversion) but for the tight integration of the TeX engine and the PostScript interpreter. The barcode example fits well in a variable-data printing workflow. It's a pity that Ghostscript cannot save a PDF in memory. If a user has a good knowledge of the PostScript language, the module can also be conveniently used as a replacement for the `gs` program.

Currently the `openMemStream` is available only in the experimental branch of LuaTeX at [4]; it's estimated that around the end of the year, it will move to the trunk branch. Both modules `gm` and `gs` are available at [2].

## References

- [1] Terry Burton. Barcode writer in pure PostScript. <http://bwipp.terryburton.co.uk>. Accessed: 2015-09-30.
- [2] ConTeXt Group. ConTeXt modules. <http://modules.contextgarden.net>. Accessed: 2015-09-30.
- [3] Luigi Scarso. The SWIGLIB project. *TUGboat*, 36(1):41–47, 2015. <http://tug.org/TUGboat/tb36-1/tb112scarso.pdf>.
- [4] LuaTeX Team. Experimental branch. <https://foundry.supelec.fr/scm/viewvc.php/branches/?root=luatex>. Accessed: 2015-09-30.

◇ Luigi Scarso  
 luigi dot scarso (at) gmail dot com  
<http://swiglib.foundry.supelec.fr>

```

\usemodule[gs]
\starttext
\gsinit

%% Load barcode.ps
\startluacode
  moduledata.swiglib.ghostscript.User = moduledata.swiglib.ghostscript.User or {}
  local _t = moduledata.swiglib.ghostscript.User
  _t.make_barcode_global_count = 1
  _t.make_barcode_pdf_prefix = 'gspsrc_1.0'
  _t.make_barcode_hash = {}
  _t.make_barcode_hashname = 'gspsrc_1.0.lua'
  if lfs.isfile(_t.make_barcode_hashname) then
    _t.make_barcode_hash = dofile(_t.make_barcode_hashname)
  end
  return
end
local barcode_ps_file = io.open('barcode.ps','r')
if barcode_ps_file == nil then
  return -1000
end
local barcode_ps = barcode_ps_file:read('*a');
barcode_ps_file:close()

local function mydev(w,h,xoff,yoff,s,name)
  return ''
end
moduledata.swiglib.ghostscript.CalculateBBox = false
moduledata.swiglib.ghostscript.Run_buffer(barcode_ps,'',mydev)
moduledata.swiglib.ghostscript.CalculateBBox = true
\stopluacode

%% Make a barcode, save it as pdf and store the name in the table barcode/pdffile
\startluacode
local function make_barcode(barcode_type,barcode_value,barcode_option,ps_option)
  local frag1, frag2, psload, psload1
  local arg1,arg2,arg3 = barcode_value,barcode_option,barcode_type
  local newline      = '\string\n'
  frag0  = (type(ps_option)=='string' and ps_option) or " 0 1 1 0 0 translate scale rotate 0 0 moveto "
  frag1  = " (%s) "
  frag2  = " (%s) /%s /uk.co.terryburton.bwipp findresource exec "
  psload1 = string.format(table.concat({'gsave ',frag0,frag1,frag2,' grestore '}),arg1,arg2,arg3)
  psload  = table.concat({psload1,' showpage',newline})
  return psload
end
local _t = moduledata.swiglib.ghostscript.User
_t.make_barcode = make_barcode
--[==[ update the db ]==]
luatex.registerstopactions(function()
  local _t = moduledata.swiglib.ghostscript.User
  local f = io.open(_t.make_barcode_hashname,'w')
  f:write("return {\n")
  for k,v in pairs(_t.make_barcode_hash) do
    f:write(string.format("[ '%s' ] = '%s',\n",k,v))
  end
  f:write("}\n")
end)\stopluacode

```

Figure 1: Producing a barcode with barcode.ps in a single instance (first part).

```

%% bridge TeX<-->Lua
\startluacode
moduledata.swiglib.ghostscript.User.gspbarcode = function (btype,bvalue,bopt)
local _t = moduledata.swiglib.ghostscript.User
local make_barcode = _t.make_barcode
local global_count = _t.make_barcode_global_count
local pdf_prefix = _t.make_barcode_pdf_prefix
local hash = _t.make_barcode_hash
local psbuf
local pdffile
local key = table.concat({btype,bvalue,bopt})
pdffile = hash[key]
if (pdffile ~= nil) then return pdffile end
pdffile = table.concat({pdf_prefix,'-',global_count,'.pdf'})
global_count = global_count+1
_t.make_barcode_global_count = global_count
psbuf = make_barcode(btype,bvalue,bopt)
moduledata.swiglib.ghostscript.Run_buffer(psbuf,pdffile)
context.gsfflush()
hash[key] = pdffile
return pdffile
end
\stopluacode

%% User macro
\def\gmpbarcode#1#2#3{\cldcontext{%
  assume no clash of macro name
  context(moduledata.swiglib.ghostscript.User.gspbarcode("#1","#2","#3"))} }

%% Examples
\hbox{\externalfigure[%
  \gmpbarcode{ean13}{2412345678901}{textfont=Courier includetext guardwhitespace}]

\externalfigure[%
  \gmpbarcode{gs1qr}{(01)03453120000011(8200)http://www.example.com}{}]
\blank\hbox{\externalfigure[%
  \gmpbarcode{leitcode}{21348075016401}{includetext}}}

\externalfigure[%
  \gmpbarcode{pdf417}{Strong error correction}{columns=2 eclevel=5}]}
\stoptext

```

**Figure 2:** Producing a barcode with `barcode.ps` in a single instance (second part).



**Figure 3:** The barcodes of figs. 1 and 2 (formatted for *TUGboat*).

## Typesetting the “Begriffsschrift” by Gottlob Frege in plain TeX

Udo Wermuth

### Abstract

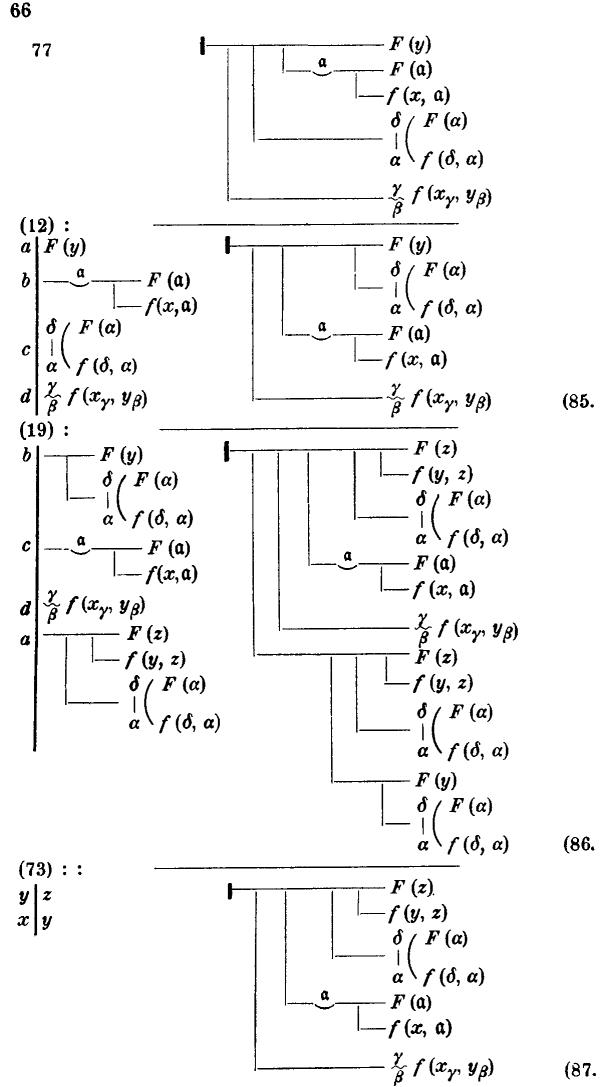
A macro package, `gfnoteation`, is described that can be used to typeset the monograph “Begriffsschrift” published by Gottlob Frege in the year 1879. The package contains two methods to input the unusual notation invented by Frege. The “symbolic representation” allows complete control about the elements and their positions and the “short form” simplifies the complexity to enter the formulas. The package includes macros to build chains of inferences and avoid problems with page breaks. It has been successfully applied to typeset the “Begriffsschrift”.

### 1 Introduction

A well-known strength of TeX is its capability to typeset mathematics. In the long history of printing mathematical formulas, some notations have appeared which are no longer in use. To typeset such notations TeX sometimes does not provide an easy solution. A book with such an outdated notation, which probably only its inventor ever used, is the “Begriffsschrift” [4] published by Gottlob Frege in 1879. Figure 1 shows a page from the book.

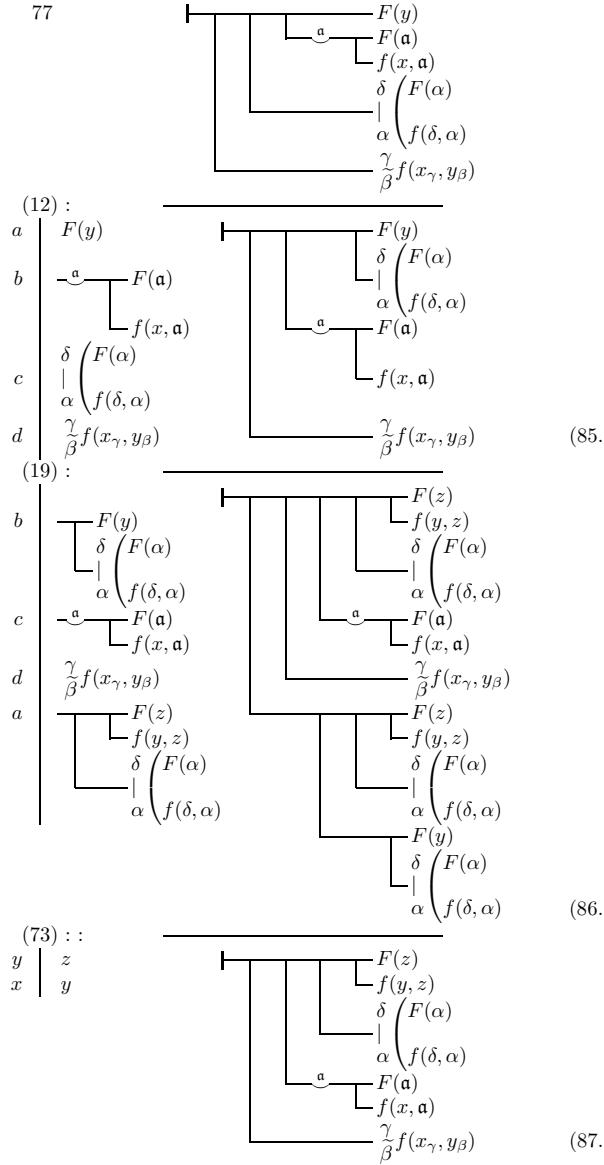
I own a facsimile reprint of the “Begriffsschrift”, and asked myself how it can be typeset with plain TeX. Of course, I realized that this would require much macro programming. My first goal was to produce a layout that comes as close as possible to the one used in the original printing of the “Begriffsschrift”. A second goal was to create a useful set of macros to typeset the whole book without great difficulty and not just a single formula. The output of my macros [25] for Fig. 1 is shown in Fig. 2.

In this article the macros that I developed to typeset the whole book are sketched, several examples of their output are given, and my approach to the problem is discussed. But first, in the next subsection, I briefly introduce the author. Then I discuss the contents of the “Begriffsschrift” and describe the importance of this monograph. The focus of the next subsections is on the notation and the challenge of typesetting it. In sections 2 and 3 I explain the two macro packages (a symbolic representation and a short form) that I wrote to allow a practical handling of Frege’s notation in plain TeX. Finally, in the last section I describe the changes to the format of Frege’s notation that occurs in Frege’s main work [7] of 1893.



**Figure 1:** Page 66 of the Begriffsschrift [4]  
(approx. 62% of original area)

**The author Gottlob Frege.** Friedrich Ludwig Gottlob Frege (1848–1925) was a German mathematician and according to his own words partly a philosopher [3]: “Every good mathematician is at least half a philosopher, and every good philosopher at least half a mathematician.” Several of his articles treat topics in the borderland between mathematics and philosophy. He was interested in an exact and rigorous foundation of mathematics and is one of the founders of the mathematical school called *logicism*, whose ultimate goal is to derive all of mathematics from logic [26]. To reach his goal Frege needed to capture imprecise linguistic phrases by exact and unambiguous statements. And he had to develop an automated system that can transform such statements without using their meaning or ac-



**Figure 2:** Page of Fig. 1 in plain  $\text{\TeX}$   
(approx. 58% of original area)

tual content. This is why he invented a new formal syntax that is introduced in the book “Begriffsschrift”. More about Gottlob Frege and his work can be found, for example, online in [27].

W. Quine wrote in 1955 (see [19, p. 158]) “All of modern logic owes an incalculable debt to Frege. If anyone can be singled out as the founder of mathematical logic, it is by all odds he.” But during his lifetime the work of Frege was not appreciated and mostly ignored. As a consequence of unfavorable reviews of his work, which show that the contemporary reviewers did not understand the important points, Frege’s academic career was severely hampered. He worked as an underpaid Honorary

Professor in Jena until his retirement in 1918. In [18] Bertrand Russell is cited with the words: “In spite of the epoch-making nature of [Frege’s] discoveries, he remained wholly without recognition until I drew attention to him in 1903.” This neglect of his work by other researchers hit Frege hard and filled him with bitterness (see [2]). Maybe even harder was the setback for the scientist through the discovery of a contradiction in his main work. A year before Frege published the second volume of his main work “Grundgesetze der Arithmetik” [7] (Basic Laws of Arithmetic [8]) Bertrand Russell wrote him a letter and pointed out that a contradiction can be constructed from his axioms in the first volume (see Russell’s letter [23] and Frege’s response [6]). Frege wrote an epilogue for the second volume of the “Grundgesetze” [7, pp. 253–265], in which he explained the problem Russell found. He tried—unsuccessfully, as is known today (see [19])—to solve it. Russell used for the contradiction an axiom of Frege about which Frege wrote in the preface of the first volume of the “Grundgesetze” that it might cause controversy [7, p. vii].

It seems that Frege was, or more likely became, a man with a difficult personality. In some of his works he attacked other scientists, and he wrote polemical texts (see [2, pp. 46–47] and [3]).

**The book “Begriffsschrift”.** Frege published his first major work in 1879 under the title “Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens” (Begriffsschrift, a formula language, modeled upon that of arithmetic, for pure thought) [4]. This long title is always shortened to “Begriffsschrift” (Concept Notation; I use the translation of the German terms as they appear in [5]). In this monograph Frege presented his automated system in the framework of a formal syntax; it was the preparation for his subsequent works where he considered the topics “number” and “quantity”. The Begriffsschrift consists of three parts. In the first part the formal system is introduced. The second part shows how to express in this system judgments of pure thought (for example, syllogisms and tautologies like “If  $a$  or  $b$  takes place, then  $b$  or  $a$  takes place.”). In the last part Frege applied the system to the mathematical theory of sequences.

The Begriffsschrift is a short book of less than a hundred pages but great importance is attached to it. The introduction to the translated text in [5, p. 1] contains the words “... it is perhaps the most important single work ever written in logic.” And on page 53 of [2] one can find the statement: “It was also the first example of a formal artificial language

constructed with a precise syntax. From this point of view, the Begriffsschrift was the ancestor of all programming languages in common use today.”

From a typographic point of view the Begriffsschrift is special because of Frege’s notation for his formal syntax. This notation did not become an accepted standard and therefore the book is not easy to read. The opposition to the notation included the waste of space and the vertical writing. Frege answered that mathematical formulas are written in a sequence of lines to obtain the advantage of the two dimensions that paper offers [9, p. 8].

Later in the first volume of the *Grundgesetze* [7] Frege repeats the definitions of his formal system, but with some small changes to the notation. He wrote [8, p. 5]: “My Begriffsschrift (Halle a. S. 1879) no longer corresponds entirely to my present standpoint; it is therefore to be consulted as an elucidation of what is presented here only with caution.” (See [13] for a discussion of the notation and the symbols of [7].)

**Frege’s Notation.** Let’s look at the notation of the Begriffsschrift in more detail as this is the main topic of the article. Greek letters are used for terminal strings:  $A$ ,  $B$ , … (uppercase Alpha, Beta, etc.). Gothic type (i.e., Fraktur) is used for a construction called *concavity*—Frege called them German letters. The following notation is used by Frege:

- *Content stroke* written as  $\text{— } A$ ; it generates an idea of  $A$ ; i.e., it is a statement. The truth of  $A$  has not yet been judged.
- *Judgment*:  $\vdash A$ ; it confirms the truth of  $A$ .
- *Negation*:  $\text{— } A$ ; it states the opposite of  $A$ .
- *Affirmation* written as double negation:  $\text{— } \text{— } A$ .
- *Condition*:  $\vdash \text{— } B$ ; it represents a conclusion.



Frege used a truth table to define the meaning of the condition:  $\vdash \text{— } B$  excludes the case in



which  $A$  is true but  $B$  is false, i.e.,  $A \Rightarrow B$ .

- *Generality*:  $\text{— } \text{— } \Phi(\alpha)$ ; it formulates a statement for all possible values of  $\alpha$ , i.e., it is a “for all” quantification.
- *Identity of content*:  $\vdash (A \equiv B)$ ; it establishes the same content for  $A$  and  $B$ , i.e.,  $A$  and  $B$  are interchangeable.

Frege allowed as variables in the generality not only elements like  $\alpha$  but also functions, i.e.  $\mathfrak{F}$ . A quantification with a function appears, for example, in formula 76 of the Begriffsschrift (see Fig. 5(b)).

As Frege defined only one binary relation, the condition, some well-known operations lack the symmetrical form in Frege’s syntax as they have in mod-

ern notation. For example, Frege analyzed words like “and” and “or” and described them in his notation as  $\vdash \text{— } B$  for “and” and  $\vdash \text{— } \text{— } B$  for “or”.



Besides the notation for formulas he introduced a notation for rules, i.e., *inferences*. At first he used only one inference and stated that later applications of the system shall define more modes of inference (see [4, p. viii]). Frege did this in [7]; his notation is described in more detail in section 4.

An inference creates from two formulas a third:

$$\begin{array}{c} \vdash \text{— } A \\ \vdash \text{— } B \\ \hline \vdash \text{— } A \end{array}$$

It means: If it is true that  $A$  is a conclusion of  $B$  and the truth of  $B$  is known, then  $A$  must be true.

Although there is only one inference three different notations are used in the Begriffsschrift to avoid the repetition of a formula. One of the two input formulas might be referenced by its number instead of being listed again. For example, if  $\vdash \text{— } A$

was established and if this formula is called  $X$  then the above inference is abbreviated:

$$(X) : \begin{array}{c} \vdash \text{— } B \\ \hline \vdash \text{— } A \end{array}$$

On the other hand, if  $\vdash \text{— } B$  is known and called  $XX$  then Frege put two colons after the reference number:

$$(XX) :: \begin{array}{c} \vdash \text{— } A \\ \vdash \text{— } B \\ \hline \vdash \text{— } A \end{array}$$

And in some places more than two formulas are involved in an inference. If the formula  $\vdash \text{— } \Gamma$  is called  $XXX$  then the chain of inferences

$$\begin{array}{c} \vdash \text{— } A \\ \vdash \text{— } \text{— } B \\ \vdash \text{— } \Gamma \\ \hline \vdash \text{— } A \end{array}$$

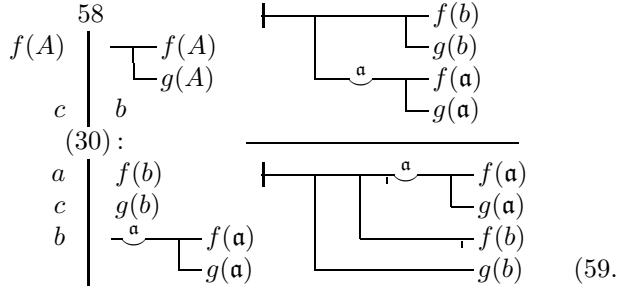
$$(XXX) :: \begin{array}{c} \vdash \text{— } A \\ \vdash \text{— } \text{— } B \\ \vdash \text{— } \Gamma \\ \hline \vdash \text{— } A \end{array}$$

$$(XX) :: \begin{array}{c} \vdash \text{— } A \\ \vdash \text{— } \text{— } B \\ \vdash \text{— } \Gamma \\ \hline \vdash \text{— } A \end{array}$$

can be written:

$$\begin{array}{c} \vdash \text{— } A \\ \vdash \text{— } \text{— } B \\ \vdash \text{— } \Gamma \\ \hline \vdash \text{— } A \end{array}$$

$$(XX, XXX) :: \begin{array}{c} \vdash \text{— } A \\ \vdash \text{— } \text{— } B \\ \vdash \text{— } \Gamma \\ \hline \vdash \text{— } A \end{array}$$

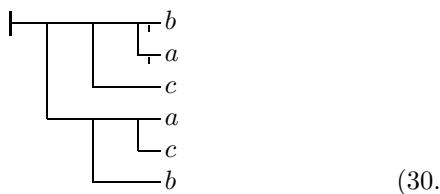


**Figure 3:** The inference for formula 59 of [4]

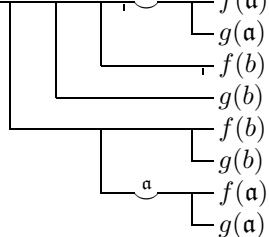
In Fig. 1, the second kind of abbreviation is used in the last inference with formula 73. The first two inferences with formulas 12 and 19 are examples of the first kind of abbreviation.

Another important notation is used for substitutions. In a previously derived formula substitutions with other formulas are possible as long as the substitution is done consistently. Frege used a vertical line to the left of a formula and wrote to the left of that line the statement that is replaced and to the right, the replacement. The formula in which the substitution shall be performed is referenced by its number over the top of the line. Figure 3 shows an example.

The first formula has the number 58 and reads  $\overline{f(c)}$ . The formula is printed above the horizontal line with two changes:  $f(A)$  is replaced by  $\overline{f(A)}$  and  $c$  is replaced by  $b$ . Formula 30 is used in the inference but it is not shown. And in this formula substitutions are performed too: The three substitutions are listed below the number 30. To allow the reader to perform the inference of formula 59 himself, here is formula 30:



And here, the abovementioned substitutions have been applied to it:  $\overline{f(a)}$ . If this



formula is placed at the top of Fig. 3, the inference for formula 59 is clear.

**Typesetting Frege's notation.** Of course, Frege was aware that the typesetting of his notation is very difficult. He argued [10, p. 364] that his two-dimensional form was easier to read and understand than the usual compression of all formulas into a single line. And he was not willing to change his layout to make the work of the typesetters easier. Using one half of a verse by Friedrich Schiller, he wrote (loosely translated): “The ease of typesetting is not, however, the supreme good.”

In the next section I present my symbolic coding to reproduce Frege's notation as closely as possible. Then a recursive notation is introduced that allows writing the input for a formula on a single line. Of course, the output is the two-dimensional structure defined by Frege and the lengths of content strokes are automatically calculated to have all terminal strings aligned.

Other people have worked on the problem to typeset Frege's notation in  $\text{\TeX}$ . J. Parsons developed a package *begriff.sty* for  $\text{\LaTeX}$  [21]. It doesn't fulfill my above mentioned goals; for example, the overall look of the notation of the Begriffsschrift is not reached and the alignment of judgment strokes or terminal strings is not easy, as the lengths of the content strokes are set manually. The look of the output is improved in [20]. The article [17] describes a GUI to support the manual work and outputs formulas in the coding of *begriff.sty*; it was used in a translation project for Frege's Grundgesetze (see [1]). The team created its own package *grundgesetze.sty* [22] to typeset the English translation [8] of Frege's main work [7]. The package is based on *begriff.sty* and inherits some of its weak points. The style of Frege's notation in his main work differs from the style of the Begriffsschrift (see section 4).

## 2 Symbolic representation

It is easy to understand that Frege's notation can be applied in a longer text only if macros are available to support the input. My first decision: Greek and Fraktur letters are entered using control words of length 2; the letter that must be typeset is preceded by either a 'g' for uppercase Greek, or a 'k' for lowercase Greek, or a 'd' for Fraktur. The assignments of characters to Greek letters follows [16, p. 20]. For example, \gA produces an uppercase Greek Alpha ( $A$ ) and \da gives  $a$ . The Fraktur font is taken from  $\mathcal{AM}$ - $\text{\TeX}$ 's *Euler* family of fonts. (Not all combinations of two letters are allowed. For example, the control word \dp is already used by  $\text{\TeX}$ .)

First, I thought to use \halign for the formulas, but this approach has some major drawbacks:

- Page breaks in long inferences are not possible when they are typeset as a single alignment. Frege never broke a formula; page breaks occur inside an inference only after the inference line.
- The templates and the horizontal positions of formulas have to be carefully aligned if each formula is an `\halign` itself.
- Laborious counting of &-signs inside a formula is needed to position the parts correctly.

As the notation requires control words for the line segments anyway, the following simple approach seems to be possible:

1. All symbols (and the empty space) of the notation are defined in a uniform length.
2. Normal text lines are used (which might have to be typeset without interline spacing).

I observed that the symbols consist of three parts. This can be seen most easily in the condition. Therefore I defined a macro `\*` with three parameters. (I saved the plain TeX meaning of `\*` as `\discretionarytimes`.)

For example, I write the triple `\*---` for the content stroke ( $\overline{\phantom{x}}$ ), which consists of three equal parts. `\*-~` is the single negation ( $\overline{\neg}$ ). `\*-:-` gives the then-part ( $\overline{\top}$ ) of a condition, which finds in `\*_~-` its partner ( $\overline{\perp}$ ) in the following line. A `\*_!-` ( $\mid$ ) is used to connect a then- and its if-part over several lines. And `\*.a.` builds the concavity ( $\overline{\text{--}}$ ) with the letter `a`. Although one might think that the colon appears only in the middle of a triple one should consider a triple like `\*: :-` ( $\overline{\top\top}$ ) to get a more compact variant to code two then-parts. (This *compact form* is required in the Grundgesetze [7], which has a two-column page layout. Figure 7 shows the compact form of the formulas of Fig. 1.) In order to connect the symbols in this coding all elements must have the same length. And of course, all horizontal lines must have the same thickness and the same position above the baseline.

First, I identified 19 different parts without the concavity. The flexibility in defining symbols from these parts is more than the Begriffsschrift requires: It uses 8860 triples in the whole text but only 73 different ones; 21 triples occur just once and only 10 more than 100 times.

Here is the list of the seven basic parts:

- is the empty space;
- is a horizontal line for the content stroke and the inference rule;
- $\sim$  is a horizontal line with centered negation;
- $\pm$  is a horizontal line with two centered negation indicators (affirmation);
- : is the then-part of the condition;

- ' is the if-part of the condition;
- ! is a vertical line (part of the so-called *condition stroke*) that connects an if-part with the corresponding then-part over several lines.

Four symbols are used for special situations:

- [ is a vertical line placed left and starts a horizontal line (a judgment with a content stroke);
- | is the vertical line for the substitution;
- = is two horizontal lines used for an inference that involves more than two formulas;
- 3 is two vertical lines used for *definitions* (see Fig. 5).

And finally there are eight symbols to fine-tune the output to match the original in certain situations:

- " a skip, that is, no output at all;
- ( negation, with the indication of negation moved to the left;
- ) negation, indicator moved to the right;
- < negation, indicator placed at the left end;
- > negation, indicator placed at the right end;
- $\sim$  affirmation, with indicators moved to the left;
- / affirmation, indicators moved to the right;
- ] a vertical line placed right, that is, only the judgment stroke is printed.

Second, the definition of the concavity requires typesetting of letters. The width of such a letter is considered to be counted as two elements of a triple. So special symbols for shorter elements are defined, which must be used in pairs. Three additional basic symbols are needed:

- a is the letter for the concavity ('a' can be replaced by other letters);
- . is a short vertical line (i.e., a content stroke);
- , is a short vertical line with a centered negation indicator;
- ; is a short vertical line with double negation.

And two more are needed for fine-tuning:

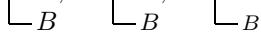
- @ signals (without any output) that a letter for the concavity follows;
- ' represents negation but the indicator is moved to the left;

A triple for the concavity must be started with one of the following symbols: ., ', @. If it is not @, then the symbol after the letter must be one of ., ', .

I call the definition of formulas in this encoding the *symbolic representation* of Frege's notation. Of course, the production of the input needs a lot of keystrokes. But it allows the creation of any formula, even those not obeying the rules, so that I could typeset the errors in the Begriffsschrift. And using an editor with a monospaced font means that the formula can be read in the input file.

```
\nlp1\ci{\hss\thinspace58\hss}\rep3\*__\*_-[-*-:-\*---\*:-:\ce{$f(b)$}
\nlpc1{$f(A)$}\*_|_`*-:\ci{$f(A)$}\rep2\*__\*_-!\*__\*_-`-\ce{$g(b)$}
\nlp1\*_|_`*-:\ci{$g(A)$}\rep2\*__\*_-`-\*.a.\*:-:\ce{$f(\da)$}
\nlpc1{$c$}\*_|_`ci{$b$} \rep3\*__\*__\*_-`-\ce{$g(\da)$}
\bcc1/m30:.rep3\*__\null \rep6\*---\ecc
\nlpc1{$a$}\*_|_`ci{$f(b)$} \rep2\*__\*_-[-*-:\*,a.\*:-:\ce{$f(\da)$}
\nlpc1{$c$}\*_|_`ci{$g(b)$} \rep2\*__\*__\*_-!\*_-!\*__\*_-`-\ce{$g(\da)$}
\nlpc1{$b$}\*_|_`*.a.\*:-:\ci{$f(\da)$} \*__\*_-!\*_-`-\*---(\ce{$f(b)$}
\nlp1\*_|_`*\*_-`-\ci{$g(\da)$} \*__\*_-`-\*---\*---\*---\ce{$g(b)$}
\fono{59}
```

**Figure 4:** Symbolic representation for Fig. 3

The output scales with the size of the font if size-changing macros are set up similarly to p. 414 of [14]. For example, here is a formula in sizes 10pt, 9pt, and 8pt:  (with constant baselineskip here; that can be changed too).

**A detailed example.** As an example, the input of Fig. 3 with formula 59 is given here in detail; see Fig. 4. So it is a real world example, containing more than the code for a single formula. Several macros must be introduced to handle the whole structure of the original text.

First, since spaces after the `\*` macro count, I use a `\null` to ignore spaces and to align the substitution and the formulas for the inference in this example.

The control words `\ci` (Character Inside) and `\ce` (Character at the End) place their arguments in an `\hbox`. In the case of `\ci` the box has the width of the `\*` macro and possibly following spaces are ignored. The macro `\fono` (FOrmula NO.) outputs the number of the formula at the right of the text line with an opening parenthesis and a period. And `\rep` (REPeat) is a macro with two arguments. The second argument must be a `\*` sequence. The output looks as if this sequence was entered as many times as the first argument states. For example, `\rep3\*__` produces exactly the same output as `\*__\*__\*__`.

The three main macros are: (a) `\nlp` (NewLine and Position), (b) `\nlpc` (NLP with Character), and (c) `\bcc #1/#2#3:#4#5\ecc` (Begin/End ConClusion). In the five parameters of (c) the third is the number of the used formula that is placed above the substitution rule. The number of colons, which shows the type of inference, is specified by the fourth parameter. The fifth parameter, which is ended by the `\ecc` marker, draws the line or lines required in the notation of the inference. The second parameter positions the colon(s). In the above-displayed case an ‘m’ (middle) is used. The other possibilities are ‘l’ (left) and ‘r’ (right). (In fact, only the ‘m’ is required by the description given above. But in order to

reproduce the original text at one place an ‘l’ is needed.) The first parameter has the same meaning as the parameter of the `\nlp` macro: it gives the indent from the left as a multiple of the width of the `\*` macro. But the `\nlp` macro does more than just the indentation. It finishes the previous line and uses a strut to specify the height and depth of the new line. This macro is discussed later in more detail. The last macro `\nlpc` is an abbreviation: `\def\nlpc#1#2{\nlp{#1}\llap{#2}}`. It does the work of the `\nlp` macro and places its second argument in the empty space created by the indentation.

My first goal is achieved with this output. It is very close to the text as it is printed in the original. The second goal, to have a set of macros to make the typesetting of the whole book “easy”, is not completely achieved. The amount of typing is huge and some counting for the positioning of symbols is required. So the output is acceptable, but the input has to be improved.

**Parameters.** The macros for the symbolic representation were written to take some parameters for changing the appearance. This is necessary as Frege made some changes in this area for the Grundgesetze [7]. All the internal macro names start with the prefix `\gfb`s and most of them have a German name after this prefix. For example, the dimen register `\gfbssstrichdicke` sets the thickness of the content stroke. And `\gfbssraise` gives the height of the content stroke above the baseline.

Here is a list of a few dimen registers:

- `\gfbssstrichdicke` for horizontal lines; default is 0.5 pt.
- `\gfbssurteildicke` for the judgment bar; default is 1 pt.
- `\gfbssersetzungdicke` for the line that is used in the substitution part; default is 0.8 pt.
- `\gfbsschlussdicke` for the thick inference line; default is 0.8 pt.
- `\gfbssraise` for the height of horizontal lines; default is 0.5 ex.
- `\gfbssneg` for the height of the negation indicator; default is 0.25 ex.

- `\gfbssuht` for the height of the judgment stroke; default is 1.5 ex.
- `\gfbssdyp` for the depth of this stroke; default is 0.5 ex.
- `\gfbsschlussabstand` for the distance between the two lines in an inference; default is 2.5 pt.
- `\gfbssvolleeeinheit` for the width of a single part in the `\*` macro; default is 0.57 em.

These parameters are used to calculate the values of other dimen registers; for example, the register `\gfbselementdimen` contains the width of a complete `\*` macro and is roughly  $3 \gfbssvolleeeinheit$ . (The units overlap a little bit to make sure that no gaps appear between the line parts.) The above dimen parameters are not used directly in the main macros of the code so that values can be adjusted for different output formats. For example, the following dimensions are used in the Begriffsschrift:

- `\gfbshoehe` equals `\gfbssraise`;
- `\gfbsnegdp` equals `\gfbssneg`;
- `\gfbssdicke` equals `\gfbssstrichdicke`;
- `\gfbshht` is the sum of the value `\gfbshoehe` and the value `\gfbssdicke`;
- `\gfbseinheit` is the width of one part of the `\*` macro minus the overlap;
- `\gfbsszweiheit` is  $2 \times \gfbssvolleeeinheit$  minus the overlap.

Besides the listed dimens two flags are defined:

- `\gfbssnegdirekt` controls whether the negation indicator and content stroke touch or leave a small gap; the gap occurs in the Begriffsschrift, but not in the Grundgesetze. The default is `\gfbssnegdirektfalse`, so the gap is present.
- `\gfbssfonooahnepunkt` controls whether the closing period in a formula number is omitted. The default is `\gfbssfonooahnepunktfalse`, i.e., the period is printed.

**Macros.** In this subsection a few aspects of the macro definitions needed for the symbolic representation are discussed. Let us look at the definition of `\nlp`:

```
\def\nlp#1{\hfil\break\gfbssstrut
\hskip#1\gfbselementdimen\relax}
```

The `\hskip` sets the current position to a multiple of the width of the `\*` macro.

The control word `\gfbssstrut` provides a strut (see [14, p. 82]) whose height and depth can be set inside the text. Such a strut is needed to get acceptable page breaks. (See pages 79 and 80 of [5] for bad breaks that can occur.) The macro package provides two commands to change the height

and depth of the strut either by a specified percentage (`\gfbssreduziereabstandum`) or by an explicit value (`\gfbsssetzeabstand`). The control sequence `\gfbssabstandzuruecksetzen` resets the strut to its original height and depth.

The main macro of the symbolic representation is simply a nested `\if` sequence. Well, at its end are 36 `\fis`, so it might not look very simple. The macro `\gfbsteilelement` processes one parameter of the `\*` macro.

Let's look at a few (simplified) examples:

```
\def\gfbsteilelement#1{%
\if...
\else\ifx #1`\% negated content stroke
\hbox to \gfbssvolleeeinheit{%
\gfbssrulefill}%
\hskip-0.5\gfbseinheit
\hskip-0.5\gfbssdicke
\ifgfbssnegdirekt
\vrule width \gfbssdicke
height \gfbshoehe
depth \gfbssnegdp
\else
\vrule width \gfbssdicke
height .8\gfbshoehe
depth \gfbssnegdp
\fi
\hskip-0.5\gfbssdicke
\hskip 0.5\gfbseinheit
\else\ifx...
\else\ifx #1`\% then-connection
\hbox to \gfbssvolleeeinheit{%
\gfbssrulefill}%
\hskip-0.5\gfbseinheit
\hskip-0.5\gfbssdicke
\vrule width \gfbssdicke
height \gfbshoehe
\hskip-0.5\gfbssdicke
\hskip 0.5\gfbseinheit
\else\if...
\fi\fi...\fi\fi...\fi}
```

`\gfbssrulefill` defines a macro for horizontal rules that have a distance of `\gfbshoehe` above the baseline. It acts like `\hrulefill` (see [14, p. 357]).

```
\def\gfbssrulefill{%
\leaders\hrule height \gfbshht
depth -\gfbshoehe
\hfill}
```

The concavity is built with the symbol `\smile`:  $\smile$  (I don't use [11]). It is placed so that the horizontal lines are attached at its ends. At the right and left the  $\smile$  has a little bit more than  $1u$  empty space. The symbol itself has a total width of  $18u$  (see [15, p. 441]). This data is used to calculate the seamless junction with the horizontal lines. For very thick lines the thicker  $\smile$  of *AMS-TEX* is used. The Fraktur letter is placed above this symbol with the macro `\buildrel` (see [14, p. 437]).

```
\ifdim\gfbssdicke<0.59 pt
  \setbox0=% plain \TeX \smile
  \hbox{\$\\mathchar"015E\$%}
\else
  \setbox0=% \AmSTeX
  \hbox{\$\\mathchar"0\hexno\cmmibfam5E\$%}
\fi
\dimen0=\gfbseinheit \advance\dimen0 by-.5\wd0
\dimen2=\wd0 \divide\dimen2 by 18 % approx. 1u
\advance\dimen0 by 1.5\dimen2
\hbox to \gfbzweiheit{%
  \hbox to\dimen0{\gfbrulefill}%
  \kern-1.2\dimen2
  \raise\gfbshoehe\hbox{\lower\ht0\hbox{%
    \$\mathrel{\frac{\#1}{\box0}}\$}}%
  \kern-1.5\dimen2
  \gfbrulefill}%

```

As mentioned earlier, the concavity needs twice the width of a single unit and the two other elements of a triple take only a half width each. So the symbols ., ‘; must appear as a pair.

```
\def\gfbsteilelement#1{%
\if...
\else\ifx #1.% half
  \hbox to 0.5\gfbvolleeinheit{%
    \gfbrulefill}%
  \ifgfbzweitehaelfte
    \gfbzweitehaelftefalse
  \else
    \gfbshoehlungtrue
    \gfbzweitehaelftettrue
  \fi
\else\ifx #1, % half and negated
...
\else\ifx #1;% half, double negation
...
\else
  \ifgfbzweitehaelfte
    \errhelp=\gfbshaelftehilfe
    \errmessage{Unexpected ...}%
  \gfbzweitehaelftefalse
\fi\fi...\fi...\fi}
```

If the flag `\gfbshoehlungtrue` is set then the next symbol is used as the letter for the concavity. The other flag `\gfbzweitehaelfte` signals that a symbol of the set ., ‘; is used instead of the @.

This code snippet shows that error conditions are caught. Using `\newlinechar='^\^J`, the following help message is shown if the pairing didn't work.

```
\newhelp\gfbshaelftehilfe{After a signaled %
“for all” one of (1) \string., (2) \string,, %
(3) \string;^\^J or (4) \string‘ must directly %
follow the variable. These symbols must be %
paired^\^J in any combination around the %
character that is used as the^\^J variable in %
the “for all”.}
```

In total more than 20 error messages with associated help messages are coded in the package.

**Special symbols.** Figure 1 shows that the terminal strings are another idiosyncratic aspect of the work,

$$\Vdash \left( \begin{array}{c} \overbrace{\phantom{\delta}}^{\delta} \overbrace{\phantom{f(\alpha)}}^{\alpha} F(\alpha) \\ \hline f(\delta, \alpha) \\ \hline F(\delta) \end{array} \right) \equiv \frac{\delta}{\alpha} \begin{pmatrix} F(\alpha) \\ f(\delta, \alpha) \end{pmatrix} \quad (69.)$$

(a) Property  $F$  is hereditary in the  $f$ -sequence.

$$\Vdash \left( \begin{array}{c} \overbrace{\phantom{\beta}}^{\beta} \overbrace{\phantom{f(x_\gamma, y_\beta)}}^{\mathfrak{F}(y)} \mathfrak{F}(y) \\ \hline \overbrace{\phantom{\delta}}^{\delta} \overbrace{\phantom{f(x, \alpha)}}^{\mathfrak{F}(\alpha)} \mathfrak{F}(\alpha) \\ \hline \overbrace{\phantom{\alpha}}^{\alpha} f(\delta, \alpha) \end{array} \right) = \frac{\gamma}{\beta} f(x_\gamma, y_\beta) \quad (76.)$$

(b)  $y$  follows  $x$  in the  $f$ -sequence.

$$\Vdash \left( \begin{array}{c} (z \equiv x) \\ \hline \frac{\gamma}{\beta} f(x_\gamma, z_\beta) \end{array} \right) \equiv \frac{\gamma}{\beta} f(x_\gamma, z_\beta) \quad (99.)$$

(c)  $z$  belongs to the  $f$ -sequence beginning with  $x$ .

$$\Vdash \left( \begin{array}{c} \overbrace{\phantom{\epsilon}}^{\epsilon} \overbrace{\phantom{f(\delta, \epsilon)}}^{\alpha} (\alpha \equiv \epsilon) \\ \hline f(\delta, \alpha) \\ \hline f(\delta, \epsilon) \end{array} \right) \equiv \frac{\delta}{\epsilon} f(\delta, \epsilon) \quad (115.)$$

(d) The procedure  $f$  is single-valued.

**Figure 5:** Special symbols of [4] and their meaning

in addition to the Frege notation. These symbols are introduced in the third part of the Begriffsschrift, in which the formalism is applied to the theory of sequences. So these symbols are not part of the formal syntax of the notation, but nevertheless they are needed to typeset the book.

With the use of the abovementioned *definition symbol* (“`\*_3-`” outputs “`\Vdash`”), Frege introduces four special symbols. All of them can be built with available symbols of *Computer Modern*. The symbol for the definition is followed by an equivalence ( $(\mathcal{A}) \equiv \mathcal{B}$ ).  $\mathcal{A}$  is a formula and  $\mathcal{B}$  is defined as the abbreviation for this formula. Just to show that these definitions can be typeset with the macros the definitions are given in Fig. 5. Because of limitations in space the compact form is used although it is not used in the Begriffsschrift.

The right-hand sides of the definitions are coded as “normal” macros. The first one has one parameter, the second and third two and the last none. The right-hand sides in Fig. 5 are named `\1F`, `\2xy`, `\4xz`, and `\5`, resp. These macros are also used in Fig. 6. (I use digits to build control symbols: The `\3` is defined to be an abbreviation for a frequently used subformula, which is written as its replacement text; see line 1 of Fig. 6. But this is not a “normal” macro and to use it “expansion” must be called first; see the exclamation mark in front of every use of `\3` in

```
\def\3#1#2{*.a.{..{{#1(\da)}}}
           .{{f(#2,\da)}}}}
\gfbkskompaktrue % use compact form
\outof p0,0"77"with\thatis
\formula p5|{..{..{..{F(y)}}
           .{!\3Fx}}
           .{\1F}}
           .{\2xy}}
\followswith p0"12"a.p4s7
\substituting p0 a:{F(y)}
b:{!\3Fx}
c:{\1F}
d:{\2xy}
\whichgives
\formula p5|{..{..{..{F(y)}}
           .{\1F}}
           .{!\3Fx}}
           .{\2xy}}
\named "85"
\followswith p0"19"a.p4s7
\substituting p0 \ :{\ }
b:{..{F(y)}}
.{\1F}}
c:{!\3Fx}
d:{\2xy}
a:{..{..{F(z)}}
   .{{f(y,z)}}}
.{\1F}}
\whichgives
\formula p5|{..{..{..{..{F(z)}}
           .{{f(y,z)}}}}
           .{\1F}}
           .{!\3Fx}}
           .{\2xy}}
.{{..{..{F(z)}}
   .{{f(y,z)}}}}
.{\1F}}
.{{..{F(y)}}
   .{\1F}}}
\named "86"
\followswith p0"73"a:p4s7
\substituting p0 y:z
x:y
\whichgives
\formula p5|{..{..{..{F(z)}}
           .{{f(y,z)}}}}
           .{\1F}}
           .{!\3Fx}}
           .{\2xy}}
\named "87"
```

**Figure 6:** The input for Fig. 7 in short form

Fig. 6. The details are discussed in the next section, which introduces the short form.)

### 3 A recursive short form

Of course, the symbolic representation is not easy to code. It would be much better to reduce the complexity of typing the symbolic representation with additional macros. So I developed a short form that

TeX transforms into the symbolic representation to reach the desired quality for the output. Figure 6 shows the input for Fig. 7 that corresponds to Fig. 2 but uses the abovementioned *compact form* for the output (see line 3 in Fig. 6).

The short form has three types of commands:

1. place a content stroke (with or without negation or affirmation indicators, i.e., signs) in front of a formula;
2. place a concavity and optional signs in front of a formula;
3. combine two formulas into a condition construction again with optional signs in front of the two formulas and for the overall construction.

If the terminal strings are called a formula too then every formula in Frege's notation can be build recursively with these commands. The optional judgment stroke is attached after the formula is created.

I decided to use signs in the above-listed cases 2 and 3 and not only in case 1. In this way case 1 is only needed when the content stroke stands alone.

The structures of the commands are as follows:

1. =⟨sign⟩{⟨formula⟩}
2. \*⟨sign⟩⟨character⟩⟨sign⟩{⟨formula⟩}
3. ⟨sign⟩⟨sign⟩{⟨formula⟩}⟨sign⟩{⟨formula⟩}

⟨formula⟩ is a previously generated formula or a terminal string, which is typeset in math mode. ⟨sign⟩ is an optional sign and has one of three values:

- . represents no sign;
- represents the negation;
- + represents the affirmation.

*Note:* The braces around ⟨formula⟩ are not required if it is a terminal string of one token. Double braces are required if the string has more than 5 tokens.

Here are some examples:

1. =.a outputs —a;
2. \*.a.{f(\da)} outputs —<sup>a</sup>—f(a);
3. ..\gA.\gB outputs  $\overline{\overline{A}}_B$ ;
4. ..\gA.{\*.a.{f(\da)}} is  $\overline{\overline{A}}_{\overline{a}—f(a)}$ ;
5. +-`gA-`gB outputs  $\overline{\overline{A}}_{\overline{B}}$ .

The fourth example shows how the formulas are nested: In the formula of the third example the *B* is replaced by the formula of example 2. The construction with = that is shown in the first example is not often used in nested formulas, as the length of the content strokes in a condition is calculated automatically to have a proper alignment.

77	
(12) :	
a	
b	
c	
d	
(19) :	
b	
c	
d	
a	
(73) ::	
y	
x	

**Figure 7:** Typesetting Fig. 1 in compact form; typed into this article (Fig. 6 shows the source)

A single formula is started with the command `\fregel`, which has two parameters. The first parameter codes the decision if the formula is a judgment;

it is a judgment if the first parameter is ‘|’ instead of ‘.’. The second parameter is the formula in the short form.

Again a few examples: `\fregel{|-. \gA+\gB}`. The generated formula is a condition with an overall negation, an unsigned *A*, and an affirmed *B*. This is the output: `\fregel{|-. \gA+\gB}` gives `\fregel{|-. \gA+\gB}` and this code outputs `|-. \gB.\gG`.

Let’s go back to formula 59 of Fig. 3. The short form for this formula is

```
\fregel{|{..{*-a{..{f(\da).{g(\da)}}}}-{f(b)}}.{g(b)}{}}
```

with the expected output `f(a).`

But remember: in the text, Formula 59 does not stand alone. We need the complete chain of inference with a correct placement of several formulas. Such an inference is not coded with the use of `\fregel`, which is only used for formulas in running text. (There are small differences between a formula in running text and in an inference.) In inferences it is replaced by `\formula`, which has an additional parameter to determine the position of the formula. The substitutions are coded as pairs of formulas separated by a colon. Figure 8 shows the complete code for Fig. 3. It uses a control symbol `\0` to make it easier to add the sub-formula with the concavity as explained below.

The parameter type “pn” is used to position the formula, in units of the `\*` macro, as before (see the `\nlp` macro above). The four-parameter macro `\outof p#1,#2#3"with#4\that is` starts a block of formulas. The first parameter is the position. The third parameter is the number of the formula that follows. The second parameter is special: It is used to place the formula number several lines lower. Usually it is 0 to get the same baseline. But in the Begriffsschrift a few cases appear that have no substitutions and the number is placed one line or two lines lower. (Recall that one of my goals was to reproduce the original text as closely as possible.) The last parameter is a sequence of formula pairs separated by colons; this is the list of substitutions. Such a list is also used as the second parameter of the macro `\substituting p#1 #2\whichgives`.

The macro `\followswith p#1" #2" a#3p#4s#5` is used to typeset the lines indicating the type of

```
\def\0#1{*#1a.{..{f(\da)}.{g(\da)}}}
\outof p1,0"58"with
  f(A):{..{f(A)}.{g(A)}}
  c:b
\thatis
\formula p5|{..{..{f(b)}.{g(b)}}}.{!\0.}
\follows p1"30" a.p5s6
\substituting p1 a:{f(b)}
  c:{g(b)}
  b:{!\0.}
\whichgives
\formula p5|{..{..{!\0}-{f(b)}}}.{g(b)}
\named "59"
```

**Figure 8:** Short form notation for Fig. 3

inference. The first parameter is the position of the used formula number that is given as the second parameter. The fourth parameter is the position of the lines for the inference and their length is given by the last parameter. The type of inference is given by the third parameter. The whole macro stands for the following (invalid TeX) statement in the notation of the above-explained \bcc... \ecc macro:  
\bcc{\#1}/m{\#2}:#3\rep{\#4-\#1-1}\*\\_\\_ \rep{\#5}  
\*---\ecc.

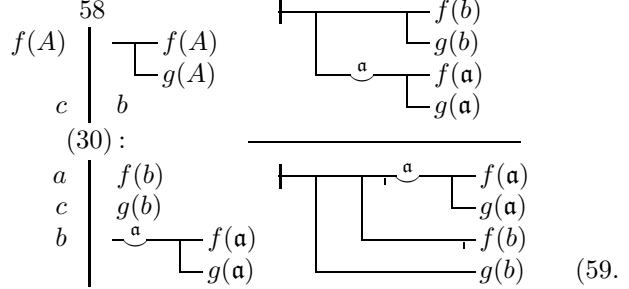
Finally, the macro \named assigns a number to the inferred formula.

Figure 3 of the inference for formula 59 shows that the sub-formula \* .a.{..{f(\da)}.{g(\da)}} has to be entered two times. Besides the three basic commands I defined a fourth one: macro expansion. The token after a ! is expanded as a *formula macro*, which might have up to three parameters. In Fig. 8 the macro \0 is defined as the abbreviation for the above expression. To make it more flexible, a parameter for the first sign was added to the macro.

The output of the code is shown in Fig. 9, which should be identical to Fig. 3.

**Parameters.** The number of lines in a formula is defined by the parameter \gfbsmaxanzahlzeilen. The default is 25 lines, which is sufficient to typeset the Begriffsschrift. For each line two pairs of \toks and \skip registers are created: the first pair codes a single line of the Frege formula, the second a substitution in front of that line. A few flags are available in the macros. The first one is a flag that is not defined as an \if:

- \nosubst must be given before the macro package is loaded. The value ‘t’ indicates that no registers for substitutions are needed, so the number of registers is reduced by 50%. (As a result, several commands are no longer usable, for example, \substituting is “turned off” and the command \outof becomes \use.)

**Figure 9:** Output of the source of Fig. 8

- \gfbslognotation controls if the result of the short form is written to TeX’s log file in an extended symbolic representation; details are in the next subsection. The default is *false*.
- \gfbszeigestats controls the output of some statistics about the maximum number of lines used in the formulas. The output is written to the log file and to the screen. The default is \gfbszeigestatsfalse.

**Macros.** The macros for the short form are much more complicated than those for the symbolic representation. They cannot be explained in greater detail in this article, but a few aspects are described; some of them might be well-known patterns.

The symbolic representation that is produced from the short form uses one \toks register for each line to store the coding. The registers carry the line number of the formula in their names. A static part of the name for the token registers is extended by the line number as a roman numeral (see [14, ex. 7.10]):

```
\csname
  gfbstoks\romannumeral\gfbszeile
\endcsname
```

This code accesses a token register whose name is created with a \csname and \endcsname construction. If the counter \gfbszeile has the value 4 then the above code equals \gfbstoksv. To add the element \gfbselement to the left of a token register the following method is used (see [14, p. 378]):

```
\edef\gfbstoken{\the\gfbselement
  \the\csname gfbstoks\romannumeral\gfbszeile
  \endcsname}
\global\csname gfbstoks\romannumeral\gfbszeile
\endcsname=\expandafter{\gfbstoken}%
```

For each line that is produced from the short form three values are stored:

- the length of the line counted as those parts of the macro \\* that contribute to the output;
- the number of the line to which the current line is connected upwards;
- the maximum line number that is connected directly or indirectly to the current line.

An example best shows how these numbers are used to build a formula. For example,



is to be formed from the two formulas



$\Gamma$ . The first formula occupies lines 1 and 2, the second line 3. Then line 1 gets the three values (2, 0, 2), line 2 (2, 1, 2), and line 3 (1, 0, 3). First the length of lines 1 and 3 are compared and the shorter line and all its “descendants” are filled with the abovementioned technique to add something to the left of a token register. Then the two lines are connected. So to join lines 1 and 3 with a condition the length of line 3 (the first value of the triple) must be increased to match the length of line 1 using one  $\ast\text{---}$ . Next, at the left side of line 1  $\ast\text{--}\text{:}\text{-}$  is added. And all its dependent lines, i.e., line 2, get a  $\ast\text{_\!}_\text{-}$ . Line 3 receives the matching  $\ast\text{_\!}_\text{'}$ . The data for the lines is updated to the values (3, 0, 3), (3, 1, 2), and (3, 1, 3), resp.

I had the (crazy?) idea to store the numbers in a single `\skip` register. The length is the normal part of the skip, the other two parameters are defined as the stretch and the shrink value.

```
\def\ggobble#1#2{\relax}
\def\setskip#1#2#3#4{%
  % #1: line no.; #2: no. of parts;
  % #3: line no. up; #4: max line no. down
  \global\csname
    gfbsskip\romannumeral #1%
  \endcsname=#2pt plus #3pt minus #4pt}
\def\getskip#1.#2 #3 #4.#5 #6 #7.#8{%
  \global\gfbssanzahl=#1% no. of parts
  \global\gfbshaengtan=#4% line no. up
  \global\gfbssgehtbis=#7% max no. down
  \ggobble{}
```

The macro `\setskip` is called with the register number, i.e., the line number, and the three values to be stored. `\getskip` assigns the stored values of the `\skip` register to three named counters when called in this way:

```
\expandafter\getskip\the\csname
  gfbsskip\romannumeral\gfbsszeile
\endcsname
```

The output of `\the` creates characters of the category “other” except for spaces. So a simple “pt” cannot be used in the parameter text (see [14, p. 375]). To delete the last “pt” the macro `\ggobble` is called. It deletes the next two tokens. In order to understand `\getskip`, the output of a well-known quantity, for example, `\bigskipamount`, should be studied.

Outputting the created symbolic representation in the log file of TeX is easy, as only the token registers must be written:

```
\wlog{\the\csname
  gfbstoks\romannumeral\gfbsszeile
\endcsname}
```

For other commands `\string` is used, for example:

```
\wlog{\string\bcc\string{\#1\string}/\m%
  \string{\#2\string}:\#3\string%
  \rep\string{\the\gfbssgrundeinzug\%
  \string}\string\*\_\_\string\rep\%
  \string{\#5\string}\string\*\_\_\string\rep\%
  \string\ecc}
```

Finally, I want to write a few words about the size of the implementation. The macro package for the notation contains more than 1800 lines of code. It is produced from a WEB-like coding system. To make sure that all the macros produce the desired output 330 test cases have been coded.

The macros grew in several steps and not all problems might have the best solution. Nevertheless I hope I have accomplished my goals and the system is quite usable.

#### 4 Extensions for the Grundgesetze

To reproduce the formulas used in Frege’s Grundgesetze [7] some adaptation of the macros is required. Besides the abovementioned compact form, which is used throughout the Grundgesetze, the most obvious changes were the use of lines that have a uniform thickness and an increased set of inferences. Figure 10 shows an example of the style used in the Grundgesetze.

This is not the place to describe the necessary changes in detail, but a few comments are worthwhile. The command `\toggleGGstyle` switches between the style of the Begriffsschrift and the style of the Grundgesetze. It uses the flag `\gfbssuseGGstyle` (with the default setting `\gfbssuseGGstylefalse`) to activate the line thickness of the Grundgesetze.

As explained above, several `\dimen` registers are used to change the output of the symbolic representation. To change these registers into the style of the Grundgesetze a few values are defined. Now the control words start with `\gfgg`:

- `\gfggstrichdicke` represents the uniform line thickness; default is 0.58 pt.
- `\gfggraise` is the height of the content stroke; default is 0.14 ex.
- `\gfggneg` is the height of the negation indicator; default is 0.47 ex.
- `\gfgguht` is the height of the judgment stroke; default is 1.4 ex.
- `\gfggudp` is the depth of the judgment stroke; default is 0.9 ex.

**Compact form.** Frege used a more compact form for the formulas in the *Grundgesetze*. (Figure 7 shows the compact form for the formulas of Fig. 2.) The short form is able to produce this form because it creates a kind of extended symbolic representation. The “extension” is the use of symbols that change their output. The selection process is complex and involves the `\*` macros to the right of the current position. Eleven more symbols are defined and nine of them can change. In the following list two forms are given for the new symbols: one for the normal output and one for the compact form.

- 0 represents negation that is either moved to the left ('`'`) or moved to the right ('`'`):  $0=()$ ;
- 1 represents affirmation:  $1=\wedge/$
- 2 again affirmation:  $2=\wedge+$ ;
- 4 represents optional content stroke:  $4=-$ ;
- 5 represents optional empty space:  $5=_-$ ;
- 6 represents judgment strokes:  $6=[]$ ;
- 7 represents a negated concavity:  $7=',$  ;
- 8 again negation:  $8=<\sim;$
- 9 represents a content stroke that has only one third of its usual length;
- \* is a skip whose length is multiplied by  $5/3$ ;
- \$ is used only with a concavity and represents either a short content stroke or an empty space.

`\gfbskompakt` switches to the compact form. The default for the flag is *false*. The compact form does not always give perfect results, i.e., it doesn't match the original. Therefore some flags are defined that have to be invoked in a problematic formula.

- `\gfbskeinekompattehoehlung` controls the use of the compact form for concavities. Its default value is *false*.
- `\gfbsaussagesichtbar` controls whether a content stroke can disappear completely in front of a statement. Its default value is also *false*.

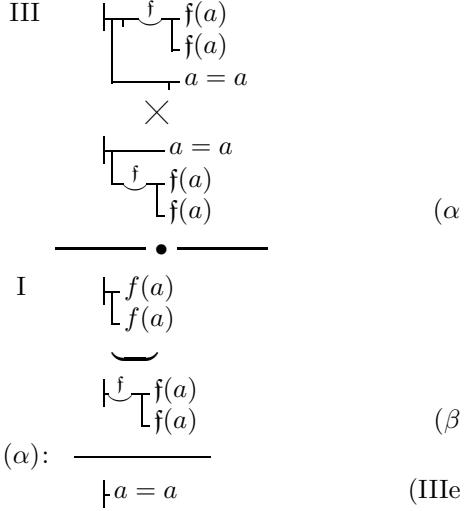
Again examples might be useful to demonstrate the effects of the flags. On the left the non-compact form is given and on the right the same formula with `\gfbskompakttrue` is output:

$$\begin{array}{c} \vdash \overline{\alpha} \Psi(\alpha) \\ \vdash \overline{\epsilon} \Phi(\epsilon) \end{array} \qquad \begin{array}{c} \vdash \overline{\alpha} \Psi(\alpha) . \\ \vdash \overline{\epsilon} \Phi(\epsilon) \end{array}$$

The sign in the first line gives an asymmetrical result on the right. Next, here is the same formula with `\gfbskeinekompattehoehlungtrue` as well as `\gfbskompakttrue`:

$$\begin{array}{c} \vdash \overline{\alpha} \Psi(\alpha) . \\ \vdash \overline{\epsilon} \Phi(\epsilon) \end{array}$$

As a second example, we can look at the output for `\frege.{=.a}` and `\frege.{=.{=.a}}`. It is identical in the compact form:  $\overline{\alpha} a$  and  $\overline{\alpha} a$ . The



**Figure 10:** Inferences from the *Grundgesetze* [7], p. 66

activation of `\gfbsaussagesichtbartrue` shows a difference:  $\overline{\alpha} a$  —  $a$ .

**New inferences.** A flag is defined to control the use of the new inferences: `\gfeggschlussweisetrue` allows the following construction in the formulas. If the first symbol in the `\*` macro is a question mark then the next two symbols are used to define the building blocks of an inference in the style of the *Grundgesetze*. Note that the thick line with the centered circle in Fig. 10 is only a separator for inference chains. It is coded as a normal macro.

The width of each of the two symbols after '?' is 50% larger than the usual width of a part in the macro `\*`. The following symbols are defined:

- ? (must be the first in the triple) signals that the next two symbols build an inference line;
- is a single horizontal stroke;
- . is a single (centered) period;
- = is a double stroke;
- \* is a single stroke in the height of the upper line of the double stroke;
- \_ is an empty space;
- " is a skip;
- > has no output; the next symbol is a *transition-sign* (now the translation of technical terms follows [8]);
- x (must follow a >) represents the *contraposition*;
- u (must follow a >) represents a “quantification”.

Now the eight transition-signs of the *Grundgesetze* can be typeset:

- a) `\*?>x` gives  $\times$  ;
- b) `\*?>u` gives  $\overline{\overline{\alpha}}$  ;
- c) `\*?--\*?--\*?--` gives  $\overline{\overline{\overline{\alpha}}}$  ;

- d)  $\backslash *?==\backslash *?==\backslash *?==$  gives \_\_\_\_\_ ;  
e)  $\backslash *?- \_ \backslash *?- \_ \backslash *?- \_$  gives — — — ;  
f)  $\backslash *?= \_ \backslash *?= \_ \backslash *?= \_$  gives = = = ;  
g)  $\backslash *?==\backslash *?==\backslash *?==\backslash *?==$  gives = = = ;  
h)  $\backslash *?. - \backslash *?. - \backslash *?. -$  gives . . . . .

Although I have not yet finished my copy of [7] I assume that this is the core that is required to typeset the formulas of the Grundgesetze. Of course, this work has special strings too. Its 29 symbols are different from those used in the Begriffsschrift; they all stay on a single line. Some symbols are available in a special font created by J. J. Green (see [12]).

## References



◊ Udo Wermuth  
Babenhäuser Straße 6  
63128 Dietzenbach  
Germany  
u dot wermuth (at) icloud dot com

## GMOA, the ‘General Manipulation Of Arguments’: An extension to the `l3expan` package of the `expl3` bundle and language

Grzegorz Murzynowski

### Abstract

After an introduction on how and why we switched to `expl3`, we present general assumptions and conventions of this language. Then we take a more detailed look at the `l3expan` package and the marvellous improvements of code it facilitates.

Then we describe our generalization and extension to the machinery provided by `l3expan`. We call it “GMOA”, “General Manipulation of Arguments” and argue why it’s quite an earned description.

We point out that GMOA is based on a finite automaton yet parses an arbitrary properly braced ⟨specification⟩ which involves recognizing the Dyck language, and we explain how does this not contradict that fundamental theorem.

### 1 Switching to `expl3` or: “Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb”

#### 1.1 The Proverb

There is a saying about one’s perception of human-made things, especially technology: “Whatever you know before 20, is just a part of Nature. What you learn between 20 and 35, is a ‘novelty’ that you’re curious or even enthusiastic about. Whatever you heard of after 35, you want to neither learn nor even know, as it’s ‘Against Nature’ and an ‘Abomination in the Eyes of the LORD.’”

Well, I understood I’d probably *have* to take `expl3` seriously when Will Robertson rewrote his `fontspec` package in(to) it some six years ago. When I was 36. Yes, exactly as the Proverb says.

Overcoming the Proverb took me four years. A new project had to begin at my work to let me think it’s a good time to learn and deploy “this abominable nonsense”. ;-)

And voilà, since only two months later all the new `TEX` code of mine is shorter by some ten thousand `\expandafter`s, so much more elegant (I hope) and noticeably less buggy.

As probably happens to all neophytes, the “Old Things” (here: `LATEX2ε`) look to me ugly and obscure while code written in `expl3`, no matter how unreadable to any “normal” `TEX` user or even a

---

Editor’s note: Originally published in the Bacho`TEX` 2015 proceedings, pp. 49–56 (GUST). Revised by the author.

`TEX`nician, seems to me to be a visible instance of utter readability and clarity<sup>1</sup>.

The “functions” have their signatures as part of their names; the “variables” (data carriers) also tell their scope and type by themselves; there’s no fear of an open `\if...`, moreover, no fear of unreadably nested conditions and `\fi`’s in wrong places, as compound condition(als) might be written as Boolean expressions and computed in just one expansion of `\romannumeral -`0` (more later on this).

And so many more of such beauties (and oddities) that at least an entire volume the size of this `TUGboat` issue might be written about them.

So—who and why should or shouldn’t be afraid of `expl3`?

#### 1.2 Who’s afraid of Virginia Woolf

As you probably have already noticed, I’m talking about *programming* in `TEX`. Not about directly *using it* for typesetting or writing texts.

The `expl3` language is intended for `TEX` programmers (such as macro package writers), not for the end users.

Therefore, if you’re using `LATEX` for writing/publishing things and when trouble strikes you do not hack some code yourself but look for an existing macro package to do the job, you have very little reason to worry and can safely stop reading here. :)

So far, we’ve discussed the “-love” of the section title. Now let’s get to the “Strange-”.

#### 1.3 General assumptions and conventions

As the reference manual/documentation<sup>2</sup> is quite comprehensive and instructive, let us only summarize what we consider most important for understanding of further material.

First of all, a special `\catcode` regime. The blank characters are assigned catcode 9 “ignored”—which means there is no need for all those to indent and choose line breaks in code yet avoid spurious spaces in text and so on.

In `expl3` you can use blanks as blanks, i.e., to make the code clearer for reading. “The blanks” include the line end (not by re`\catcode`ing it, but in another, beautifully twisted way) so also blank lines can be used just for organizing of code.

It took me some time to get used to this. E.g., under this catcode regime, with this code:

```
\ifnum 1=0
  1a
```

---

<sup>1</sup> cf. O.Wilde, “The Importance of Being Earnest”.

<sup>2</sup> “The `LATEX3` Interfaces”, <http://mirrors.ctan.org/macros/latex/contrib/l3kernel/interface3.pdf>

```
\else
 0z
\fi
```

you get the condition satisfied and the conditional expanded to `a`, the oppose of what what a wannabe-*TEX*nian like myself is used to. (As the first end-of-line doesn't translate to a space (catcode 10) but is just ignored, and the number read at the right side of comparison is 01.

On the other hand, when you use `expl3`, you are not supposed to use *TEX* primitives (with their original names and syntax, at least).<sup>3</sup>

Other important re-catcodings are that `_` and `:` are made letters (cat.11) and used in control sequence names as significant or generic word joiners and separators. Without going into too much detail here, let's give a couple of examples. First:

```
\l__gme_additional_letters_tl
```

means a local `_` internal variable of type `_tl`, ‘token list’, of the “module” (macro package/document class) `gme`.

And second:

```
\prg_replicate:nn
```

means an `expl3` “function” of the module “`prg`”, taking two braced arguments (usually, like a *TEX* macro with two undelimited parameters; this will be discussed later).

The obligatory `:` separates the `expl3` function name from its signature. The signature is a (finite, possibly empty) sequence of the letters:

```
c, D, f, n, N, o, p, T, F, w, x.
```

Each letter specifies one argument, except for `w`, which means “weird syntax” and `D`, which means “DON'T!”, for *TEX* primitives and other commands that should not be used outside the `expl3` kernel.

It should be emphasized that the “functions” of `expl3` are not always macros, either in the sense of taking undelimited arguments or in implementation. One should rather think of them as a conceptual construct, like *LATEX* 2 $\varepsilon$  “commands”.

---

<sup>3</sup> This “hide your *TEX* and forget you ever used it” (cf. Marquis' de Sade's “Speech to the women” at the beginning of “120 Days of Sodom”) is probably the only thing I definitely dislike in `expl3` (as I did in *LATEX* 2 $\varepsilon$ ).

It's partially because of my intellectual inertia, I admit. But also, and more importantly, because of my strong belief that not many people (if anyone) could compete with Prof. Knuth in the *texyve* (‘art/craft’) of computer programming. Which implies that hardly anything written on top of *TEX* could be, I'm not saying: better than it; I'm saying: *comparable* with it. I believe that some of *TEX*'s “beauties-oddities” will strike at some point anyway, and then looking down to some lowest-level things is inevitable.

It's also explicitly stated in the Reference Documentation that the uppercase letters (“argument types”) `N` and `V` require a single token without braces, while all the others allow many tokens, in braces; for those latter, wrapping even a single token in braces is encouraged.

The `p` argument type is solely for macro parameter strings in the sense of *The TEXbook*'s chapter 20 and therefore can neither contain braces nor be wrapped with them. We'll see later how this works with the `I3expan` conventions and (not quite) with our extension to them.

Here is a variant of the previous function:

```
\prg_replicate:Vo
```

This requires the first argument to be an `expl3` variable and renders its value; the second argument is hit with one `\expandafter` and only such preprocessed arguments are given the “original” or “primary” `\prg_replicate:nn` function.

As we are talking “functions” in `expl3`, they “return” a “result”. We use these words in the sense of `expl3` henceforth; bear in mind that, translated to our good old *TEX* jargon, they generally mean ‘what at some point is left in *TEX*'s “mouth” after execution of this stuff’ or, most often, just ‘expand to’.

Which brings us to part two, the pre-expansion of arguments with the `I3expan` package.

## 2 The `I3expan` package: Pre-expansion of macro arguments

Consider the not-so-unusual situation that before applying a macro, the soon-to-be arguments should be preprocessed (“tenderized”<sup>4</sup>). For instance, a control sequence `\arg@i@int`, a `\count` register or `\numexpr`, hit with `\the`; an `{(arg.2)}` subjected to `\edef`; and `{(condition)}{(arg.3T)}{(arg.3F)}` expanded either to `{(arg.3T)}` or `{(arg.3F)}` but not further, depending on `(condition)`.

In *LATEX* 2 $\varepsilon$  we would write something like:

```
\newtoks \l@aux@args@toks
\newtoks \l@auxA@toks
\newtoks \l@auxB@toks
\l@auxA@toks = {{(arg.3T)}}
\l@auxB@toks={{(arg.3F)}}
\edef\aux@macro {%
  \if<test> <condition>
    \the\l@auxA@toks
  \else
    \the\l@auxB@toks
  \fi}
```

---

<sup>4</sup> cf. <http://myfirstdictionary.blogspot.com/2011/03/todays-word-is-shoo.html>

```

}

\l@aux@args@toks
\expandafter {\aux@macro } \% "{\arg.3\@}" 

\edef \aux@macro {\{arg.2\}}
\l@aux@args@toks
\expandafter\expandafter\expandafter
{\expandafter \aux@macro
\the\l@aux@args@toks }
% "{\{arg.2-ed\}}{\{arg.3\@}\}" 

\expandafter \def \expandafter
\aux@macro \expandafter{%
\the \arg@i@int \% it can be a numexpr,
we don't know the num. of tokens
}

\l@aux@args@toks
\expandafter\expandafter\expandafter
{\expandafter \aux@macro
\the\l@aux@args@toks }
% "{\{val.of arg.1\}}{\{arg.2-ed\}}{\{arg.3\@}\}" 

% and, finally,
\expandafter \_\_mod_foo:nnn
\the \l@aux@args@toks

```

As we can see, that's rather hard core. Even with the shorthands of our `gutils` package(s), it wouldn't look much better.

Now, with `I3expan`, we can type

```

\::v \::x \::f \:::
\_\_mod_foo:nnn
\arg_i_int {\{arg.2\}}
{\{test\}:ntF{\{condition\}} {\{arg.3T\}}{\{arg.3F\}}}

```

Four lines instead of ca. 30.

Or, if we expect such pre-expansions more often, we can introduce a *variant* of the `expl3` function `\_\_mod_foo:nnn`:

```

\cs_generate_variant:Nn
\_\_mod_foo:nnn {Vxf}

\_\_mod_foo:Vxf
\arg_i_int {\{arg.2\}}
{\_\_(condition):TF {\{arg.3T\}}{\{arg.3F\}} }

```

By the way, if the nature of pre-processing allows, all those `\:::` macros are expandable. How is that done? Except for `\:::`, all the `\:::`'s are `\long` 3-parameter macros with #1 delimited with `\:::`, #2 undelimited and #3 depending on the nature of pre-processing, usually undelimited. For instance, `\::o` is defined like (`TEX` primitives are given here in their original names, `I3expan` uses them in `expl3` aliases; please remember the blanks are cat.9 “ignored”):

```

\long\def
\::o #1 \::: #2#3

```

```

{ \expandafter \_\_exp_arg_next:nnn
\expandafter {\#3} {\#1} {\#2}
}
\long\def
\::f #1 \::: #2#3
{
\expandafter \_\_exp_arg_next:nnn
\expandafter { \romannumeral -`0 #3 }
{\#1} {\#2}
}
\long\def
\_\_exp_arg_next:nnn #1#2#3
{\#2 \::: { \#3 {\#1} } }

```

Which means that `\::o` applies one `\expandafter` to its #3 and then lets `\_\_exp_arg_next:nnn` do the rest.

Similarly, `\::f` applies `\romannumeral` in such a way that any leading expandable tokens of #3 are expanded until the first unexpandable token is seen. Please note the `-`0` preceding #3. Thanks to it, even if #3 expands to decimal digit(s), `\romannumeral` is satisfied with `-`0` as a complete `<number>` specification and, as this number is negative (the charcode of 0 is 48), expands to `<empty>` (empty sequence of tokens). Thus we get an “AFAP” (‘As Far As Possible’) expansion of #3 in just one `\expandafter`.

(Very few things move me as deeply as this trick, if I may express my personal yet professional feelings here.)

Some of the pre-processors which render the value of a variable also use `\romannumeral-`0`, depending on the variable type. The current implementation of the `_tl` type, for instance, as parameterless macros not, e.g., as `\toks` registers, allows for rendering their value with just an `\expandafter`, or even just the use of the variable name, in many contexts.

What happens next: the `\_\_exp_arg_next:nnn` macro appends the pre-processed `\::\@`'s` #3 to the end of (that `\::\@d`) #2 (the result-so-far). All of that finally looks like:

```

\#1 \% remaining \::\@`'s
\::: {\{#2\}[{\{#3 pre-processed\}}]}
<further input>

```

The mysterious `\:::` delimiter control sequence is `\firstofone` or, in its `expl3` naming, `\use:n`; so that when all the `\::\@`'s` do their job, it strips the outer braces off the final result.

For those pre-processors which pass their result stripped of braces, there's another version of the “pass-the-result” macro:

```

\long\def

```

```
\__exp_arg_next:Nnn #1#2#3
{ #2 \::: { #3 #1 } }
```

As you can clearly see, the difference in behaviour of these macros seems to be reflected in lower-/uppercase opposition of the letter(s) `n` vs. `N`.

However, both the Reference Documentation and further explanations from the L<sup>A</sup>T<sub>E</sub>X3 team discourage such an interpretation, underlining the `expl3` goal to allow (force) the user “not to rely on implementation” and focus on the requirements (in most cases purely conventional): `N` “requiring” a single and un-braced token, most often a control sequence, and `n` “requiring” an arbitrary token(s) in braces, as in the *⟨balanced text⟩* of *The T<sub>E</sub>Xbook*.

However, if we really stick to the conventions and don’t rely on implementation<sup>5</sup> (how is *that* possible for a T<sub>E</sub>X programmer, let alone a T<sub>E</sub>Xnician? :-)), everything seems to work just fine.

Which leads us to some remarks concerning the pre-expansion of arguments and its generalization to GMOA. (With tidying-up the “argument types” with respect to it.)

### 3 GMOA, a ‘General Manipulation Of Arguments’ or: a DFA that seemingly recognizes the Dyck Language

Long story short, on top of the `l3expan` pre-expanders we add arbitrary rearrangements and/or grouping of arguments. In a “one-char” syntactic manner similar to the `tabular` specifications. And expandable (except for `x` and `X`). And with consistent naming conventions, coherent with `l3expan` to some degree.

And, last but not least, performed (at the stage of translation of the specifiers sequence into `\:::`-like macros) by a DFA, a (deterministic) finite automaton. Which includes a Dyck Language recognition. Without falsifying the theorem that there does not exist a DFA to do that.

---

<sup>5</sup> There is a temptation of using some functions as if they were (e)T<sub>E</sub>X primitives, like `\hbox:n` (as currently allowing the `\bgroup... \egroup` syntax) or `\tl_to_str:n` as the supposed `expl3` alias for `\detokenize`. Just DON’T. Quoting Joseph Wright’s email, “...for example, where the team needs to use the primitive behaviour of `\detokenize [...]`, we use the ‘raw’ name `\etex_detokenize:D`”.

I’m not sure what a non-(L<sup>A</sup>T<sub>E</sub>X3 team) person should do in this case, since the Reference Documentation reads: “The `D` specifier means *do not use*. [...] Only the kernel team should use anything with a `D` specifier!” We’re again at “Hide your T<sub>E</sub>X and forget you’ve ever used it”? Over my dead body.

The `gme3u8.sty` package (of mine) provides aliases for the (e)T<sub>E</sub>X primitives my way and anyone is invited to use them (at their own risk, of course). (The list is not complete, I just add what I needed so far.)

**Roadmap for this section.** After an excuse for using non-ASCII chars which need a special font, we give a formal definition of the GMOA specifications. Then we present an overview of the GMOA machinery, its *modus operandi*, and some of its macros alongside with their conceptual structure as a finite automaton.

In the subsequent (subsubsequent? ;-)) (sub-sub)sections we present groups of specifiers and respective automaton states / stages of parsing:

- the `⟨destination⟩` tokens,
- the `⟨prep-or-↓⟩`s,
- the meta-operators,
- the digits / `⟨FSM⟩`s,
- the braces / `⟨BDSM⟩`s

We end this section with examples: one that is almost-comprehensive though not-necessarily-useful and a handful of “real-life” ones. All of them go beyond what’s possible with `l3expan` only.

**A disclaimer about (non-)ASCII chars and the custom font Ubu Stereo.** Before we proceed, a remark. The `expl3` language keeps strictly to ASCII. Any characters outside of ASCII that occur in the next part of this paper, especially those from the “distant far-aways” of the Unicode or even from the Private Use Area (PUA henceforth), are entirely my “fault and guilt”.

I use some letters looking similar to some non-letter ASCII chars with the intention of making the names even more self-explanatory, while simultaneously respecting the `expl3` naming conventions. Other characters are chosen for perfectly rational reasons (e.g., `⌚`, the astrological symbol of the name of my dearest dog, which I use as my “trademark” in the `expl3` module part of names). And, a some chars `FontForged` by myself to depict T<sub>E</sub>X primitives and other often-used things in one character each.

All are put together in one font named Ubu Stereo. The font is based on Ubuntu Mono with some characters copied from other libre fonts, especially DejaVu Sans and FreeSerif. I act with those fonts as I please (PL: “Wedle mojego widzimisię”), hence “Ubu” (cf. A. Jarry, “Ubu le Roi” &c.). “Stereo”, because it’s not all monospaced, as some wide characters are given double width for better distinction, such as `—`, and some combine to double width in pairs in a kind of typographical *rubato*, with one char half-width (declared as another escape char) and the other one-and-a-half-width, to provide one-char control sequences (I’m curious if they constitute reasonable mnemonics to anyone but me):

```
⌚ \expandafter
⌚ \noexpand
```

```
\unexpanded % e-TeX primitive
\csname ... \endcsname % stator(s) of
an electric motor; make the stuff between them
spin.
```

With that description, we switch the mono font to Ubu Stereo from now on and get down to GMOA at last.

### 3.1 Description by examples

To give an idea of GMOA, let us rewrite an example from the previous section.

```
\::: I Vxr :
\__mod_foo:nnn
\l_arg_i_int {\{arg.2\}}
{\<condition>:TF {\{arg.3T\}}{\{arg.3F\}} }
```

where `\:::` is the name of the main GMOA macro (subject to alias on user's request) and the subsequent letters are the specifiers of operations, "the operators" for short. So far, nothing more than in `\I3expan`, as `I` stands for "Identity" and just preserves `\__mod_foo:nnn` until the rest of the arguments are preprocessed, `V` and `x` act as (are translated to) `\:::v` `\:::x` and `r` translates to (an alias of) `\:::f`, i.e., does the `\romannumeral -`0` trick.

But let's have a look at some real-life use (courtesy of the PARCAT project, `parcat.eu`):

```
\::: 1_{4_2_5_6_7_8} 1_3_{4_3_5_6_7_9} :
\DeclareOption
{oneside}%
{twoside}%
\PassOptionsToClass
{report}%
\protected\def
\__ins'_page'oddity'count:%
\c_one \c@page
% 1
% 2, 3
% 4
% 5
% 6
% 7
% 8, 9
```

This code declares a `LATEX 2 $\epsilon$`  document class options `oneside` and `twoside` that differ only in the meaning of `\__ins'_page'oddity'count:` and both pass their names to the basic document class `report`. I.e., `\:::...:` rearranges the code above into:

```
\DeclareOption {oneside}
{\PassOptionsToClass {oneside} {report}%
 \protected\def \__ins'_page'oddity'count:%
 {\c_one }%
}
%
\DeclareOption {twoside}
{\PassOptionsToClass {twoside} {report}%
 \protected\def \__ins'_page'oddity'count:%
 {\c@page }%
```

Some parts of the code have been replicated as many times as needed, their order changed, some groups of arguments were put into the same pairs of braces.

By writing all the *mutatis mutandis* text just once with the "mutandis" not repeated, the code is kept strictly parallel, i.e., change-robust, as any change need be made in only one place.

On the other hand, using this machinery has the obvious disadvantage that you have to learn the mini-language of specifiers. Hopefully, this is an acceptable expense. Another inconvenience is the counting of the arguments, which might be quite tricky, especially if there are mixed sequences of stand-alone operators and `\langle FSM\rangle/\langle BDSM\rangle` parts in a specification (discussed in the examples following the formal definition).

I dare think of `\:::` as superior<sup>6</sup> to both the `\I3expan` low-level `\:::\text{#}` macros and the machinery of `\cs_generate_variant:Nn` in some aspects.

It provides much more general rearrangement and pre-processing options than either of the latter. It has much shorter syntax than the `\:::\text{#}` macros; moreover, it can also be used within the usual catcode regime (provided the very name is aliased properly), as the specifiers are parsed with the `\strcmp` comparisons, which are independent of the catcodes.

Except for braces, which are discussed in the section 3.2.6.

### 3.2 GMOA: Formal language definition

```
<GMOA> ::= \::: <specification> :
<specification> ::= 
  <destination><FSO><optional .>
  <specification>
<destination> ::= <empty> | ξ | ζ | σ
<optional .> ::= <empty> | .
<FSO> "Finite Sequence of Operators" ::=
  <empty> | <SALos><optional ;><FSO>
  | <FSM><optional ;><FSO>
<optional ;> ::= <empty> | ;
<SALos> Stand-Alone's ::= <preps'n'↓s>
  | <SALos><(prep.seq.)><SALos>
  | <SALos>'<prep><SALos>
  | <SALos>*<prep><SALos>
  | <SALos>*<prep><SALos>
  | <SALos><meta-R><SALos>
<preps'n'↓s> ::= <empty>
  | <prep-or-↓><preps'n'↓s>
<prep-or-↓> ::= <prep> | ↓
<prep.seq.> ::= <empty> | <prep><prep.seq.>
```

<sup>6</sup> It's infinitely easier to expand/develop something than to invent it in the first place. `\I3expan` does things I've not even thought of in ten years of my TeXnician's life. Or if I did, it was: "Nah... it's impossible; you just can't hit the 2nd undelimited argument with `\expandafter` since you don't know how many tokens are there in the first one".

$\langle \text{prep} \rangle ::= c | \epsilon | \dot{\epsilon} | \delta | \theta | f | h | H | i | I | k | K | n | N$   
 | o | O | \Theta | p | P | q | Q | r | R | s | S | T | F | v | V | x  
 | X  
 $\langle \text{meta-}R \rangle ::= R \langle \text{arbitrary-meta-}R \rangle$   
 |  $\times \langle \text{number specification} \rangle \langle \text{meta-replicated} \rangle$   
 $\langle \text{arbitrary-meta-}R \rangle ::= a(\text{ny}) \text{ TeX code that}$   
 $R\text{-expands to [consistent part of] a}$   
 $\langle \text{specification} \rangle$   
 $\langle \text{number specification} \rangle ::= \text{the}$   
 $\backslash \text{prg\_replicate:nn}'s \text{ first argument}$   
 $\langle \text{meta-replicated} \rangle ::= \text{the } \backslash \text{prg\_replicate:nn}'s$   
 $\text{second argument}$   
 $\langle \text{FSM} \rangle \text{ "Finite Sequence Manipulation"} ::=$   
 $\langle \text{opt.cardinality} \rangle \langle \text{FSM w.card.par.} \rangle$   
 $\langle \text{opt.cardinality} \rangle ::= \langle \text{empty} \rangle | | \langle \text{digit} \rangle |$   
 $\langle \text{FSM w.card.par.} \rangle \text{ "FSM with cardinality}$   
 $\text{paradigm known"} ::=$   
 $\langle \text{FSM chunk} \rangle \langle \text{optional ,} \rangle$   
 $\langle \text{FSM w.card.par.} \rangle$   
 $\langle \text{FSM chunk} \rangle ::= \langle \text{digits with prep.seqs.} \rangle$   
 |  $\langle \text{BDSM} \rangle$   
 $\langle \text{digits with prep.seqs.} \rangle ::= \langle \text{empty} \rangle$   
 |  $\langle \text{digit} \rangle \langle \text{prep.seq.} \rangle \langle \text{digits with prep.seq.} \rangle$   
 $\langle \text{digit} \rangle ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9$   
 |  $\dot{A} | \dot{B} | \dot{C} | \dot{D} | \dot{E} | \dot{F} | \dot{3} | \dot{2} | \dot{3} | \dot{4} | \dot{5} | \dot{6} | \dot{7} | \dot{8} | \dot{9} | \dot{0} | \dot{B}$   
 |  $\dot{c} | \dot{g} | \dot{e} | \dot{x}$   
 $\langle \text{optional ,} \rangle ::= \langle \text{empty} \rangle | ,$   
 $\langle \text{BDSM} \rangle \text{ "Braced Dyck-language Sequence}$   
 $\text{Manipulation"} ::=$   
 $\{ \langle \text{FSM chunk} \rangle \} \langle \text{prep.seq.} \rangle$

The chars of a specification are hit with `\string`, one by one, so each gets catcode 12 (other) unless it has catcode 5, 9, 10, 14, or 15.

The only cat.1 and cat.2 characters allowed are { and } and they have to be balanced.

Blank chars, either in `expl3` catcode regime (i.e., cat.9 “ignored”) or in the usual (cat.10) are skipped and may be used at will to improve readability.

Let us now explain the semantics, i.e., “see what this mouse trap really does”.<sup>7</sup>

### 3.2.1 Overview of the DFA

`\::` is a macro with one parameter delimited with `:11` (catcode “letter”). Taking an entire `(specification)` at once serves only for checking if there is a predefined macro to handle the given case.

If no such shortcut is predefined, the code

**¶ \\_\\_::<sup>1</sup>\_str<sup>o</sup>s<sup>u</sup>KN:w** ⟨specification⟩ :{}

is executed, where `\K` is just `\csname` and the next control sequence is a GMOA first-stage (`:1`) `\string`-preceded (`str`) macro for the state (`'s`) named `KN`,

<sup>7</sup> cf. William Tenn, "Errand Boy", in "The Seven Sexes", 1968.

'Know Nothing'. Which is, as one can guess, the initial state.

From there on the characters of `\specification` are hit with `\string`, picked one-by-one, tested, and transitions performed accordingly, which TEXnically amounts to putting further and further “telescopic” `\csname`’s, i.e., the sequences of tokens that could and should be transformed into a control sequence at the very moment the matching `\endcsname` is met.

Only, the immediate predecessor of each such `\endcsname` is ... `\expandafter`. And the token next to `\endcsname` is ... another `\csname`:

```
\csname cs-name-1 \expandafter\endcsname%
      \csname cs-name-2 ...
```

I think of this trick as of an (architectural) arch or bridge; and I think of `\csname... \endcsname` as the stator(s) of an electric motor, which make(s) the stuff between them spin. Hence the PUA/Ubu signs based on Japanese quotation marks:

FK cs-name-1 FK cs-name-2 FK

Now, a GMOA

\::: ξΙο σΙιρ ::

is at some point translated into:

¶ ::I\* ¶ ::o\* ¶ ::I ¶ ::i ¶ ::r ¶ :::: ¶

and turns into the `\::[` macros from right to left.

At the moment of submitting this paper for printing, GMOAs DFA consists of 12 states with 64 transitions (and 11  $\langle$ meta- $\rangle$  interruptions).

The transitions are labeled not with particular characters but with equivalence classes of: ⟨prep⟩s, ⟨\*meta-operator⟩s, ⟨destination⟩ tokens, &c.

It's probably not a significant savings of memory or other computational costs, but does result in a great simplifying of the code. It also makes the code more change- and development-robust; e.g., adding a new argument type, which is denoted with a char of equivalence class  $\langle \text{prep} \rangle$ , does not require any changes in the automaton.

### 3.2.2 The destination

Parsing of a  $\langle$ specification $\rangle$  starts from determining of the “destination”, i.e., the way the result of the next  $\langle$ FSoO $\rangle$  is yielded:

`\empty`) If no explicit destination token is given,<sup>8</sup> the usual “just once” is assumed, like `\3expan`'s `\:::@/\__exp_arg_next:nnn`. This is equivalent to the use of `c`.

ς Greek letter small sigma final form, the open variant, for “συναγωγή πολύ” /synagogue poly/.

---

<sup>8</sup> i.e., the first char met is none of  $\varsigma, \sigma, \xi$ .

‘gather (as) many’ (with intended associations with the correlation between social diversity and open-mindedness of people). Therefore let us call this “just once and multi”.

$\sigma$  stands for “συναγωγή μόνο” /synagogue mono/, ‘gather [as] one’, Greek letter small sigma middle form, the closed variant, to be associated with enclosing of all the picked and pre-processed arguments in one common resulting pair of braces. (“Just once and as one”.)

$\xi$  for Greek “Ξανά”, [use] again’: the result is put back as input for further parts of specification, after this ⟨destination⟩ is done (quite like ruminants do). (“We’ll meet again”, as in the final credits of “Dr. Strangelove or . . . ”)

[We skip the description of the “Grand Gathering” post-meta ⟨SAlos⟩  $\Sigma$  and  $\Xi$  as having originated in the formative stage of the GMOA language and most probably doomed for deletion. Basically, they act like  $\varsigma$  and  $\xi$ , only applying to “everything done so far” and have notably less elegant syntax.]

### 3.2.3 The ⟨prep-or-↓⟩’s

If one understands (the beauty of) the l3expan preprocessors, then the ⟨prep⟩’s are the easy part.<sup>9</sup> All those letters either directly correspond to some expl3 “argument type” and the respective  $\backslash\text{\textcolon\textcolon}$  macro (and are internally translated to it), or extrapolate their idea, perhaps even towards a kind of a completeness or full(er) symmetry.

They can be divided into two groups: “argument pre-processors” and “special pickers”, the latter being  $\text{H}, \text{h}, \text{p}, \text{Q}, \text{q}$ . They pick (scan) the arguments delimited in special ways as described below, and don’t do anything else to them. All the others, the “argument pre-processors”, take undelimited arguments (again in the sense of *The TeXbook*, chapter 20) and submit them to various kinds of expansion or “rendering”.

What needs to be emphasized is the expl3 concept of “argument types”, although mostly consistent with my idea of upper- and lowercase operators, uses the opposition of upper- and lowercase with a different purpose. The original intent, expressed both in the Reference Documentation and in the L<sup>A</sup>T<sub>E</sub>X3 team members’ responses to my emails, is that  $\text{N}$  and  $\text{V}$  require a single and unbraced token, usually a control sequence, while all the others allow many tokens, in that case in braces (except the  $\text{p}$  type for obvious reasons).

<sup>9</sup> cf. “Devil’s Advocate”, Al Pacino as Satan talking salary with Tom Cruise.

(And, not to forget that the  $\text{T}$  and  $\text{F}$  argument types refer to the argument originally braced yet returned without braces.)

This is, as far as I understand it, part of the general goal of creating an abstract programming language, not necessarily using TeX in the future.

My ideas are more moderate. I’d like just to provide some easier ways to memorize and shortcut some programmatic constructs in TeX, for a person who is acquainted with TeX and with at least some of its beauties-oddities. Therefore the GMOA mini-language of one-char specifiers consistently (to the maximum extent permitted by expl3) “thinks” of the uppercase specifiers as referring to the resulting arguments unbraced and their lowercase counterparts as referring to the arguments braced, up to introducing “lowercase digits”, as we’ll see later.

So, let us see what the pre-processors do. The ones homonymic with l3expan “argument types” are put in [square brackets].

$[\text{c}], \text{c}, \text{\texttt{C}}$  The uppercase  $\text{\texttt{C}}$  is an alias for  $\text{c}$ , i.e., applying  $\backslash\text{csname}\dots\backslash\text{endcsname}$  before passing the argument on *without* braces. The lowercase  $\text{c}$  does the same, only passing the result on in braces. What is this useful for? First, for the TeX primitives that require a ⟨balanced text⟩. Second, for theoretically possible constructs like

```
cs name 1\expandafter\endcsname\csname cs
      name 2 ...
```

$\text{\texttt{d}}, \text{d}$  (Icelandic, etc., letter eth/Eth) Hit the argument with  $\backslash\text{the}$ . In the current implementation of expl3, it’s almost equivalent to  $\text{V}$  for some expl3 data types, namely:  $\text{_int}$ ,  $\text{_dim}$  and  $\text{_skip}$ .

$\text{\texttt{d}}$  is equivalent to  $\text{V}$ , and  $\text{D} - \text{VI}$ .  $\text{v}$  is equivalent to  $*\text{c}\text{\texttt{d}}$  in stand-alone contexts or  $\text{c}\text{\texttt{d}}$  as ⟨prep.seq.⟩ of an ⟨FSM⟩ or ⟨BDSM⟩.

However, due to the “Don’t rely on implementation” rule one should *always* use the  $\text{v}$  or  $\text{V}$  specifier to render the value of an expl3 data carrier. (Which is a bit more expensive than the  $\backslash\text{the}$ .)

$\text{\texttt{h}}, \text{H}, [\text{p}]$  Pick the #{}-delimited argument and return it without braces ( $\text{H}$ ,  $\text{p}$ ) or wrapped in braces ( $\text{h}$ ).

$\text{\texttt{i}}, \text{I}$  Identity operation, braced or unbraced with respect to the lettercase.

$\text{\texttt{k}}, \text{K}$  \detokenize the argument.

$[\text{n}], [\text{N}]$  No pre-processing. Equivalent to  $\text{i}/\text{I}$  in the current implementation of l3expan (described separately as that implementation should not be relied upon).

$[\text{o}], [\text{O}]$  One level expansion with  $\backslash\text{expandafter}$ . (As currently implemented in l3expan.)

о,ø Cyrillic letter lowercase/uppercase binocular o.  
Two-level expansion with `\expandafter`. Used by the author for elegant (in his opinion) pre-processing of macros that should be expanded to their content and that content hit again, e.g.:

```
\def\number_of_page:{\the\c@page}
```

q,Q Pick the argument delimited with `\q_stop`.

[f],r,R Apply `\romannumeral - \emptyset` to the argument. This causes the leading token(s) of that argument to be fully expanded until an unexpandable token is seen. So, it's called f for the "full" and not called so by myself for the "until first unexpandable". Here comes a point where I prefer to refer to (some) knowledge of TeX, namely, of the `\romannumeral` primitive.

s,S Hit the argument with `\string`. It's worth emphasizing that the uppercase variant returns the result without braces, which for a control sequence means at least two bare tokens (except for `\_u`).

[T],[F] Strictly parallel to the `\Bexpan/expl3` homonyms (functionally equivalent to our I but intended to indicate the conditional branches).

[v],[V] Strictly parallel to the `\Bexpan/expl3` homonyms (rendering the value of a data carrier (an `expl3` variable or constant), given as a control sequence for V or as the tokens of its name for v). Related to `\d` and `\D`; see above.

[x],X Hit the argument with `\edef`. The lowercase variant translates to the `\:::x` macro in `\Bexpan/expl3` which, in its current implementation, returns the result in braces. The uppercase variant returns the result without braces and is not present in `expl3`.

The `\downarrow` operator discards the respective argument. Therefore there is no need to use it in the `(FSM)`s (including `(BDSM)`s), as there you just skip a digit for that purpose.

When used as `SALos`, the `(prep-or-\downarrow)`s refer to and are applied to subsequent arguments from the input.

When following a `(digit)`, the `(prep)`s refer to and are applied to the `(digit)th` argument from the input, counting as explained later.

When following a `BDSM`, the `(prep)`s refer to that `BDSM` as if it was a single argument taken from the input.

Before we deal with the `(FSM)`s and `(BDSM)`s, a word on the meta-operators as they apply to the just-presented `(prep)`s, and other specifiers that constitute categories (GMOA "char-classes") by themselves.

### 3.2.4 The meta-operators

The `\``, `*`, `‡` and `(...)` meta-operators, i.e., the operators modifying actions of `(prep)`s, are intended to allow applying multiple pre-processings to the same argument without calling the `(FSM)` machinery (which is much more expensive, as we'll see). As they are redundant with respect to the general power of GMOA, we leave the details for an interested reader in the documentation.

The `\`` meta-operator (or rather: interruptor) suspends parsing of `(specification)`, hits whatever is next to it with `\romannumeral - \emptyset` and then hopefully resumes the parsing. That allows you to branch the very specification of a given GMOA, not only its arguments. Including nesting of GMOAs. Does it increase its expressive power? Probably not, but it lets you write code in a more meta-way. Also in a way much more obscure, yet shorter.

`\x` is a shorthand for `\` \prg_replicate:nn`; thus it requires two pairs of braces to follow, the first containing a `(number specification)` and the other the things you wish to replicate. This way, instead of

```
\::: ... ↓↓↓↓↓↓ ...:
```

you can type

```
\::: ... ×8↓ ...:
```

As you may have noticed, at this point I do rely on the current implementation of `\prg_replicate:nn`, both on its expandability and on the fact that so far it's a macro with two undelimited arguments (i.e., the presence of braces is not obligatory in fact).

Let's now deal with the `(digit)`s, that is, the general permutations.

### 3.2.5 The general permutations, or the `(FSM)`s without regrouping

The `(digit)`s refer to subsequent arguments from the input. However, the counting starts after the earlier `(FSO)`s are done and the preceding `(SALos)` from the current `(FSO)`.

Which means that we skip the arguments picked and processed within earlier (possibly implicit) `\c` and `\s` destinations and the `\downarrow`'s independent of destination, and include the ones destined `\xi`, and all stand-alone `(prep)`s from the current destiny. Consider

```
\::: \c \ii \xi \o\downarrow. 1343 :  
  \__gme_foo:nnn {\{1st arg.\}}{\{2nd arg.\}}  
  {\\"gme_arg' \B:N \__gme_\d'ed:\}% and then d.1  
  {\{to be \downarrow-discarded\}}  
  {\{d.2-arg (discarded)\}}{\{d.3-arg\}}{\{d.4-arg\}}
```

The text of the 3rd line gets hit by `\expandafter`. The argument from the next line is discarded by

↓ (no matter that it's within a  $\xi$  /*xana/* destination). As the . is seen, the  $\xi$  ⟨FSO⟩ is finished, i.e., the  $\expandafter$  argument is put back for further processing. Which means it becomes the “digit 1” of the last ⟨FSO⟩ (with implicit destination  $\varsigma$  just like the first).

Three subsequent braces are picked according to their specifications and since the digit 2 does not occur in the ⟨FSO⟩, the {⟨d.2-arg...⟩} brace is discarded.

The result looks like:

```
\__gme_foo:nnn {\1st arg.}{\2nd arg.}
{\one-lev.exp.of}\__gme_arg'2B:N \__gme_\`ed:
{d.3-arg}{d.4-arg}{d.3-arg}
```

The processing of a “general permutation” can be described as two stages: (stage one) preparation of the “slab”<sup>10</sup> and then (stage two) picking numbered arguments from it.

Stage one consists of picking of proper number of arguments (only undelimited so far, but one can put some  $\xi$  ⟨FSO⟩ before that contains some “special pickers”, right?), (re)wrapping them in braces and separating with indices which will be the argument delimiters for the “digit”-picking macros.

In this role we have just “bare” digits and Latin capital letters (hex digits so far), which is safe since the (GMOA) arguments’ contents is “invisible” to *TeX*’s macro argument scanner, thanks to the braces.

For instance, a slab (or, closer to *TeX* digestive tract metaphors, a “craw”) for the 4-permutation in the example above looks as follows:

```
(the FSM translated into \:::?-like macros)
\::: {} % container for the result
\:::_:::_FSM"args % start-delimiter of the “slab”
1{\langle d.1-arg\rangle} 2{\langle d.2-arg\rangle} 3{\langle d.3-arg\rangle} 4{\langle d.4-
arg\rangle}
\q_:::_FSM'craw"stop % end-delimiter of the “slab”
or “craw”
{\tail of the (whole) ⟨specification⟩}\::: % 
delimited just like in \3expan
{\langle the result of earlier part of ⟨specification⟩\rangle}%
```

The slab is functionally a one-dimensional array (a vector). The accessor macros read all the stuff until their proper index and the argument after that index and add that argument to the result container, and return all the stuff including the “original” copy of their argument back to the slab.

It looks very expensive; it would probably be more efficient to define index-named macros whose contents would be the ⟨FSM⟩’s arguments. Then

<sup>10</sup> “Let’s go to the lab ‘n’ see what’s on the slab”, “The Rocky Horror Picture Show”.

access to an element would cost only one  $\csname...$   $\endcsname$  plus one one-level expansion of it. (Plus one initial  $\long\edef{\unexpanded{\langle arg.\rangle}}$ )<sup>11</sup> each.)

But when implemented this way, it stays expandable. Why is that so important? I’m not sure. But it’s certainly fun to be able to process quite complex rearrangements with just one  $\romannumeral`0$ .

But, one may ask, how does GMOA know how many arguments should be taken for an ⟨FSM⟩?

As ⟨opt.cardinality⟩ is ⟨empty⟩ here, the largest value of ⟨digit⟩ is assumed. Again, in an expandable way, via an initial assumption of 0 and comparing current ⟨digit⟩ with the largest-so-far, which is passed as an argument to the next step of parsing-expansion.

If the ⟨opt.cardinality⟩ value is given explicitly, the comparisons are performed anyway and an error raised at a digit exceeding the cardinality declared.

Each ⟨digit⟩ of an ⟨FSM⟩ may be followed by ⟨prep.seq.⟩, in which case a respective sequence of operations is applied to the resp. copy of the resp. argument, as stated above.

To allow arguments beyond the 9th, the  $\text{A}\dots\text{F}$  symbols are used with those “ribbon-accents” to distinguish the hexadecimal digits A–F from their Latin-letter counterparts (in this role, Unicode points from the PUA, U+E990–5 as provided in the Ubu Stereo font, and also Elisp functions for inserting them in my GNU Emacs), whereas the latter might become ⟨prep⟩s in the future (the letter D is already in use by *expl3*, although not handled by GMOA). (Yet internally represented as hexadecimal digits A…F.)

The “lowercase” hex digits  $\text{\`a}\dots\text{\`z}$  serve as shorthands for  $\text{1i}\dots\text{f}$ , i.e., while  $\text{1}\dots\text{f}$  render the respective arguments without braces,  $\text{\`a}\dots\text{\`z}$  pass their arguments (re)braced (which is why in the Ubu Stereo font they have those tiny under- and over-braces).

So, it seems we are handling a dynamic-length data structure within purely expandable sub-*TeX*. Are we really? Yes, to some degree. Namely, to the largest number (of arguments) for which “slab”-preparing and “slab”-referring macros are previously defined.

### 3.2.6 The ⟨BDSM⟩’s mystery explained or: how a DFA can recognize a Dyck language ;-)

A ⟨BDSM⟩, “Braced Dyck-language Sequence Manipulation”, is an enrichment of an ⟨FSM⟩ with (balanced) braces (of cat.1 and cat.2).

<sup>11</sup> The need for  $\long$  is clear.  $\edef{\unexpanded$  serves to avoid “the hash clash”.

As stated above, any closing brace can be followed by `\{prep.seq.\}` as if it was a single `\{digit\}`, in which case the sequence of operators is applied to that entire brace as one argument.

The GMOA machinery can parse arbitrary (re-)grouping specifications, limited of course by TeX's capacity and other resources.

The translation is performed by (a part of) a DFA, a deterministic finite automaton. That is, by a construct whose well-known limitation is its inability to recognize a language of properly paired and arbitrarily nested parentheses. Or, as “naturally” comes to a TeXnician's mind, curly braces; a.k.a. the Dyck language.<sup>12</sup>

Nonexistence of such a DFA is proven *ordine geometrico*. A proof formalizes the intuition that a (given) machine with only finite set of states, say  $n$ , cannot properly “count” the braces nested deeper than  $n$  levels.

As stated above, each token of a GMOA `\{specification\}` is hit with `\string` so it's actually the opening brace `\{_{12}` (“other”) taken into further processing from every `\{BDSM\}` chunk.

But—the mystery is unveiled—although the very first and opening brace of a given `\{BDSM\}` is hit with `\string` and thus turned to `\cat{.12}` “other”, and so are all the remaining ones, it's *not* done character by character.

When the automaton meets an opening brace which has already been “petrified” (if we are fans of Platform 9¾) or “denatured” (if we'd like to `\Br{eak}{Ba}{d}`), it uses this trick (`\{`` is `\expandafter`, remember?):

```
\{`\{`ifnum 0=\{`\}`}\fi
```

to put an unbalanced `\{1` back and then use TeX's argument scanner to pick the entire `\{BDSM\}` chunk.

So, it's not the GMOA DFA which “recognizes a von Dyck language” but TeX itself. We have not subverted Computer Science!

But that's just the beginning.

Next, an outermost `\{FSM chunk\}` like this is `\detokenized` and a special delimiter appended to it.

Then, this part of `\{specification\}` is parsed again, char-by-char, with the opening brace starting a new branch of (binary) concatenations, terminated with a (unary) operation of bracing when the closing brace is met, with the leafs being `\{digit\}`s, and rewritten to Reverse Polish Notation (RPN) (with a variant of the Shunting Yard Algorithm, of course).

<sup>12</sup> More precisely, the language recognized apparently “by GMOA's DFA” is a Dyck language's closure with respect to insertions of `/(\{digit\}\{prep.seq.\})*/`'s and concatenations with `/(\{SALos\})*/`'s, where “`/.../`” denotes a regexp.

Once the special BDSM's delimiter is met, the automaton goes to the state “Yield what you've Reverse-Polished” which results in emptying another kind of a “craw” into the main result container.

### 3.3 An almost-comprehensive usage example (yet not necessarily useful)

Assume (redefining core commands just for brevity):

```
\escapechar \c_minus_one
\def\w{wia} \def\v{\u} \def\t{tro} \def\u{\t}
\quark_new:N\q_pia
```

and set

```
\showtokens \{ \_R\_go: % debug
showCommands
\::: % <<<< % My Precioussssssssssss
\x % \ as in “\ava”, “[use] again”; \ as in “hash”,
parameter delimited with #
*I % “Eat the cookie and have the cookie:” take
an argument from input, yield one copy to the
result and put another back at input.
. % the limit of \x's scope.
% a new \destination begins; as the next char
is not a \destination token, an implicit \s,
“συναγωγή πολύ”, ‘gather (as) many’ is assumed.
21 % take two (undel.) arguments from input and
put them in reversed order (outer braces off).
*I
\x % an explicit \destination token terminates the
scope of the previous one.
\o
(ooo)% (parentheses) apply all three o's to the same
argument.
\hi
`oo. % apply o to an arg. and leave for further
 preprocessors, i.e., for the second o; equivalent
 to (oo).
2345671 {28\$]8\$8{\$8\$8\$]1
28\$871 \$86871 79\$79\$79\$1.
1111111. % equivalent to *I*I*I*I*I*I*I*II.
\o 1 4369785 2.
: % end of specifiers
% now the text(s) to be passed through GMOA:
\ta\ metoda\ nazywa\ sie\ {\u-\!1}\{\u-\!2\}
{\^{\u}} % 1
\w\ \v\ areo\ {\te}\{\ra\}
% 2 3 4 5 6
\{`\use_none:nn \_R\_go:\cs_to_str:N\q_pia}
% 7
- ! {\tra} {\ae} {\u}
% 8 9 \ A B C
{\u}
[]\AaBrsu
% 123456789
{\u}
% end of text for \showtokens.
```

And the result is (typed out in the terminal):

```
\:::_ ->
{\showtokens}
>
ta metoda nazywa się
wiatroareoterapia
{{wia}}-{{tro}}-{{areo}}-{{te}}-{{ra}}-{{pia}}
wia-tra-pia
ae-ra-pia
pia! pia! pia!
{[A.Bursa]}

.
l.43 }
% end of text for \showtokens
?
```

And the “`\:::_`-like macros” are (`\escapechar=0`):

```
_:::_prepare'\t<\xi>:w % the letter τ for Greek “το
τέλος”, ‘the End, the Final Destiny’ :3
::h ::↓ ::↓ ::I* _:::_yield:w
_:::_prepare'\t<\zeta>:w :::_prepare'FSM:w "F#2 "I
    "F#1 "I
q_:::_FSM'craw"start 2 % the digit of cardinality
::I* _:::_yield:w
_:::_prepare'\t<\xi>:w ::i ::o ::o* ::o* ::o ::h
    ::i ::i ::o* ::o _:::_yield:w
_:::_prepare'\t<\zeta>:w :::_prepare'FSM:w
"F#2 "I "F#3 "I "F#4 "I "F#5 "I "F#6 "I "F#7
    "I "F#1 "I "Bε "B#2 "B"wrap"i "Bø "B#8
    "B"wrap"i "Bø "B#3 "B"wrap"i "Bø ::i "F#8
    "I "F#4 "i "F#8 "I "Bε "B#5 "B"wrap"i "Bø
    "B#8 "B"wrap"i "Bø "B#6 "B"wrap"i "Bø
    "B#8 "B"wrap"i "Bø "B#7 "B"wrap"i "Bø ::i
    "F#1 "I "F#2 "I "F#8 "I "F#A "I "F#8 "I
    "F#7 "I "F#1 "I "F#B "I "F#8 "I "F#6 "I
    "F#8 "I "F#7 "I "F#1 "I "F#7 "I "F#9 "I
    "F#C "I "F#7 "I "F#9 "I "F#C "I "F#7 "I
    "F#9 "I "F#C "I "F#1 "I
q_:::_FSM'craw"start C % the digit of cardinality
_:::_yield:w
_:::_prepare'\t<\zeta>:w :::_prepare'FSM:w "F#1 "I
    "F#1 "I "F#1 "I "F#1 "I "F#1 "I
    "F#1 "I "F#1 "I
q_:::_FSM'craw"start 1 _:::_yield:w
_:::_prepare'\t<\alpha>:w :::_prepare'FSM:w "F#1 "I
    "F#4 "I "F#3 "I "F#6 "I "F#9 "I "F#7 "I
    "F#8 "I "F#5 "I "F#2 "I
q_:::_FSM'craw"start 9 _:::_yield:w :::
{[]} the main result container
ta metoda nazywa się ... % the input
```

### 3.4 Real-life uses of GMOA

The GMOA mechanism can be used anywhere that `\3expan` can be, and with no need of “generating variants”, yet still with shortcuts for the typical cases,

as the `\specification` is first tested for existence of a predefined macro like `\3expan \exp_args:....`

Thanks to the `p/H` and `h` `\prep`s, we are able to pick any number of unbraced tokens delimited with an opening brace. That allows for, among other things, creating “semi-transparent” conditionals that disappear in one branch or discarding such `h` sequences in the other. Or just pick an entire left side of an assignment with one `h/H` (again, courtesy of the PARCAT project):

```
\::: Hr :
\$_pdef:Npn
\PagesTotal
{ % sth. expandable to sth. hopefully useful
\cs_if_exist:NTF
\c_ins'_pages'total_int % if a count register /
_int variable is defined—
{ \int_use:N \c_ins'_pages'total_int }%
then brrring it on!—
{ \Rstop: \textbf{\{??\}} }% ... otherwise expand to
“We don’t know.”
}
```

The *mutatis mutandis* multiple definitions in a single piece of code were discussed in section 3.1.

Another interesting use has been noticed only while preparing this paper for print:

The well-known trick,

```
\begingroup
\obeylines %
\firstofone{\endgroup %
\def
{\(sth.useful)\}%
}
```

(the line end following the last `}` is back of the usual cat.5, i.e., there’s *no* line end at all ;-) yet the cat.13 “active” line end has been redefined to `\(sth.useful)` could not be easily applied to things other than cat-codes. Such as locally recording a locally changed font size and bringing that local record beyond the scope of the changed font size.

Well, now it can be:

```
\smaller % set the font one-step smaller than current
\font ((gm)resize)
\::: Io :
{ \group_end:
\$_def: \_\$_font'size"smaller: }
\f@size % the inner LATEX2ε macro bearing current
font size is expanded and wrapped in braces
before closing current group, i.e., before the font
size is reverted to its previous value.
```

The code above opens a group, then changes the font size (locally), including redefinition of the LATEX2ε

macro `\f@size` that bears the size value in `pt`; then the GMOA preprocessing expands that (locally) re-defined macro to its contents, i.e., to the literal value of current font size and (re)wraps it in braces; then puts `\group_end:...` before the expanded and braced literal font size value, thus making it robust to the closing of group and making it the body of the macro `\_\$\`font'size"smaller:`.

It's true that that could be written in "pure `\expandafter`", although with different bracing and more `\:::\``'s:

```
\group_begin:
\smaller
\:::N \:::N \:::o \:::
\group_end:
\fedef: \_\$\`font'size"smaller:
{ \f@size }
```

But what if we also need another macro that bears both the value of the smaller `\f@size` and of the normal `\baselineskip`? (So that the `\acro` command changes only the size of letters and not the line spacing.)

```
\group_begin:
\::: \xi{12}o. % render current baseline skip, wrap it
               in brace and return back to the input—
\% ... and revert the order of braces, stripping
off the originally-second; these <digits> constitute
a separate <FSM> with the <destination> implicit
\xi and refer to the rendered and braced value
of \baselineskip and the brace with the inner
GMOA.

:
\the\baselineskip
{
\smaller
\::: \xi{10. I 12\%, 13{45}} : % the last <digit> %
refers to the value of (normal) \baselineskip
rendered by the outer GMOA.

{ \group_end:
\fedef:
\_\$\`font'size"smaller:
\_\$\`font'sizes"smaller:
}
\f@size
}
```

### 3.5 GMOA as a part of the `gme3u8` package and *in statu viae*

We realize that GMOA, along with the whole `gme3u8` macro package, with all its "far" and even PUA Unicode usage that require a tailored font and special input methods (such as Elisp functions for GNU Emacs), are not useful for anyone except the author of this article.

However, we intend to make it usable for the others as soon as we get some signals of interest. Indeed, we'll be grateful for any remarks concerning GMOA, especially suggestions for development, and programming in `expl3` in general.

◊ Grzegorz Murzynowski  
PARCAT.eu  
[g.murzynowski@parcat.eu](mailto:g.murzynowski@parcat.eu)  
[natror@sentr.at](mailto:natror@sentr.at)

### Production notes

Karl Berry

As a reader might imagine, editing this article posed unusual challenges. Grzegorz provided his `Ubu Stereo` font (discussed in the text) to make it possible.

To process the article, we used `XeLaTeX`, using file-name lookups for the fonts: `FreeSerif.otf`, `DejaVuSans.ttf`, `Carlito-Regular.ttf`, `UbuStereo-Regular.ttf`. Avoiding system font lookups allows the article to be processed on different systems without having to change their font configurations, highly desirable for `TUGboat`.

For the actual editing, however, it was necessary to make it work on GNU/Linux, so I edited my configuration file `~/.fonts.conf` to contain the line:

```
<dir>/some/directory</dir>
```

in the `<fontconfig>` block, where `/some/directory` is the directory where I saved `UbuStereo-Regular.ttf`. To my knowledge, all GNU/Linux systems use Fontconfig (`fontconfig.org`) to find application fonts.

To make Fontconfig know about the new directory:

```
fc-cache -fv # | sort >/tmp/fc
```

The commented-out part redirects lots of possibly-but-not-necessarily interesting output from the terminal.

This command shows the names of all the (scalable) monospaced fonts available:

```
fc-list :spacing=mono:scalable=true family |sort
Sadly, Ubu Stereo does not show up here, as it is technically not monospaced (per the article). Eliminating the :spacing=mono selector (i.e., listing all scalable fonts), it does appear.
```

To use the font in a standard terminal:

```
xterm -fa 'Ubu Stereo' -fs 19
```

The font size (`-fs`) is what worked best on my monitor.

Then I ran GNU Emacs (`gnu.org/s/emacs`) within the xterm: `emacs-nw`. Running Emacs directly under X had complications I didn't need to track down. I used the latest Emacs (24.5), compiled from the original source, as Unicode support is one of the most active development areas in Emacs.

I didn't need Grzegorz's Elisp code (referred to in the article), since I could use the existing `unitext`.

A final lament: I find that `xterm`, `emacs`, and other programs just drop characters from UTF-8-encoded source when input and font do not match perfectly. What happened to "be liberal in what you accept"? Beware ...



## The Treasure Chest

The following is a list of selected new packages posted to CTAN (<http://ctan.org>) from March through October 2015, with descriptions based on the announcements and edited for extreme brevity.

Entries are listed alphabetically within CTAN directories. 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. Comments are welcome, as always.

◊ Karl Berry  
tugboat (at) tug dot org

---

### biblio

**bestpapers** in **biblio/bibtex/contrib**  
Maintain an author's list of "best papers".  
**bookdb** in **biblio/bibtex/contrib**  
BIBTeX style for cataloging a home library.

---

### fonts

**academicons** in **fonts**  
Makes available 20 icons from the Academicons font.  
**cjk-gs-integrate** in **fonts/utilities**  
Tools to integrate CJK fonts into Ghostscript.  
**comicneue** in **fonts**  
Comic Neue support.  
**esrelation** in **fonts**  
Symbol set for describing relations per Byron Cook.  
**\*imfellenglish** in **fonts**  
IM Fell English fonts, based on original Fell types.  
**old-arrows** in **fonts**  
CM old-style arrows with smaller arrowheads.  
**sourceserifpro** in **fonts**  
Source Serif Pro support.  
**svrsymbols** in **fonts**  
New font with symbols for use in physics.  
**\*tempora** in **fonts**  
Greek and Cyrillic to accompany Times.  
**typicons** in **fonts**  
Makes available 336 icons from the Typicons font.

---

### graphics

**blochsphere** in **graphics/pgf/contrib**  
Pseudo-3D diagrams of Bloch spheres via PGF/TikZ.  
**karnaughmap** in **graphics/pgf/contrib**  
Karnaugh maps via PGF/TikZ.  
**mcf2graph** in **graphics/mcf2graph**  
Chemical structure diagrams with Metafont/MetaPost.

**roundrect** in **graphics/metapost/contrib/macros**  
Configurable rounded rectangles in MetaPost, with optional text.

**shapes** in **graphics/metapost/contrib/macros**  
Polygons, reentrant stars, and fractions in circles with MetaPost.

**xebaposter** in **graphics/pgf/contrib**  
Scientific posters for **xepersian** using PGF/TikZ.

---

### info

**greekinfo3** in **info/greek**  
"Twenty-five years of Greek TeXing": Overview of systems, packages, and fonts for Greek TeX.

---

### language

**arabi-add** in **language/arabic**  
Support for **hyperref** and **bookmark** with Arabic and Farsi.

---

### macros/generic

**termmenu** in **macros/generic**  
Support for terminal-based menus using **expl3**.

---

### macros/latex/contrib

**alertmessage** in **macros/latex/contrib**  
Display alert messages (informational, warning, etc.).  
**bewerbung** in **macros/latex/contrib**  
Typeset a job application.  
**br-lex** in **macros/latex/contrib**  
Typeset Brazilian legal texts.  
**bxpdfver** in **macros/latex/contrib**  
Specify output PDF version and compression level.  
**cfr-initials** in **macros/latex/contrib**  
Templates for use of **initials** package.  
**\*checklistings** in **macros/latex/contrib**

Pass verbatim contents through a compiler and reincorporate the resulting output.

**cleanthesis** in **macros/latex/contrib**  
Clean, simple thesis style.  
**cntperchap** in **macros/latex/contrib**  
Store counter values per chapter.  
**colorspace** in **macros/latex/contrib**  
PDF color spaces for spot colors and overprinting.

**copyedit** in **macros/latex/contrib**  
Copyediting support for LATEX documents.  
**denisbdoc** in **macros/latex/contrib**  
Denis Bitouzé's documentation support package.

**diadia** in **macros/latex/contrib**  
Diabetes diary.

**dynamicnumber** in **macros/latex/contrib**  
Dynamically typeset numbers and values.  
**easyreview** in **macros/latex/contrib**  
Support reviews and editorial tasks.

**elements** in **macros/latex/contrib**  
Properties of chemical elements (article in this issue).

**elocalloc in macros/latex/contrib**

Local allocation macros (per `etex.sty`) for L<sup>A</sup>T<sub>E</sub>X 2015.

**exercises in macros/latex/contrib**

Typeset exercises and solutions, with automatic addition of points.

**fcavtex in macros/latex/contrib**

Thesis class for FCAV/UNESP, Brazil.

**fitbox in macros/latex/contrib**

Fit graphics on a page.

**fithesis in macros/latex/contrib**

Thesis support for Masaryk University, Czech.

**gradstudentresume in macros/latex/contrib**

Template for graduate student resumes. (See article in this issue.)

**gzt in macros/latex/contrib**

Support for *La Gazette des Mathématiciens*.

**hang in macros/latex/contrib**

Environments for hanging paragraphs and list items.

**lstbayes in macros/latex/contrib**

listings language driver for Bayesian modeling languages: BUGS, JAGS, Stan.

**medstarbeamer in macros/latex/contrib**

Beamer template for MedStar health presentations.

**multiaudience in macros/latex/contrib**

Generate audience-specific versions of one document.

**nevelok in macros/latex/contrib**

Automatic definite articles for Hungarian.

**nmbib in macros/latex/contrib**

Successor to multibibliography for multiple bibliographies with different sort orders.

**numending in macros/latex/contrib**

Morphological end of units for East Slavic languages.

**pdfpagediff in macros/latex/contrib**

Find differences between two PDF documents.

**proofread in macros/latex/contrib**

Commands for inserting annotations.

**recipebook in macros/latex/contrib**

Typeset 5.5" × 8" recipes for browsing or printing.

**reledmac in macros/latex/contrib**

Successor to eledmac/eledpar for critical editions.

**rmathbr in macros/latex/contrib**

Repetition of operator at a line break within an inline equation.

**screenplay-pkg in macros/latex/contrib**

Package version of the screenplay class.

**semproc in macros/latex/contrib**

Seminar proceedings based on KOMA-Script.

**shdoc in macros/latex/contrib**

Float environment to document a terminal session.

**tagpair in macros/latex/contrib**

Word-by-word glosses, translations, and bibliographic attributions.

**uassign in macros/latex/contrib**

Typeset class assignments.

**ucharcat in macros/latex/contrib**

Lua implementation of the (new in 2015) X<sub>E</sub>T<sub>E</sub>X \Ucharcat primitive, for LuaT<sub>E</sub>X.

**whuthesis in macros/latex/contrib**

Thesis template for Wuhan Univ. undergraduates.

**xellipsis in macros/latex/contrib**

Configurable ellipses with predefined formats for common styles.

**xpiano in macros/latex/contrib**

Extension of the piano package.

**macros/latex/contrib/beamer-contrib****fibeamer in m/l/c/beamer-contrib**

Beamer theme for thesis defense presentations at Masaryk University, Czech.

**macros/latex/contrib/biblatex-contrib****archaeologie in m/l/c/biblatex-contrib**

Citation style for the German Archaeology Institute.

**biblatex-opcit-booktitle in m/l/c/biblatex-contrib**

Support for 'op. cit.' in the booktitle of a subentry.

**biblatex-subseries in m/l/c/biblatex-contrib**

Manage subseries with BIBL<sub>E</sub>T<sub>X</sub>.

**macros/luatex****cloze in macros/luatex/latex**

Create cloze reading comprehension texts.

**cstablestack in macros/luatex/generic**

Category code table stacks.

**macros/plain****gfnotation in macros/plain/contrib**

Typeset Frege notation (see article in this issue).

**macros/xetex****bidihl in macros/xetex/latex**

Experimental bidi-aware text highlighting.

**quran in macros/xetex/latex**

Typeset the whole or a part of the Qur'an.

**support****autosp in support**

Preprocessor to generate note-spacing commands for MusiX<sub>T</sub>E<sub>X</sub> scores.

**\*bibtexperllibs in support**

BIBL<sub>E</sub>T<sub>X</sub> (pure) Perl libraries from CPAN.

**comment\_io in support**

Python script to (un)comment lines.

**\*make4ht in support**

Simplified build system (includes library) for tex4ht.

**pdbf-toolkit in support**

Toolkit for creating Janiform data documents.

**pdfbook2 in support**

Create booklets from PDF files, in Python.

**srcredact in support**

Support a master source with output for different audiences. (See article in TUGboat 36:2.)

**tex4ebook in support**

Convert to ebook formats from L<sup>A</sup>T<sub>E</sub>X, using tex4ht.

## An online glossary of typographic terms by Janie Kliever

Boris Veytsman

While movable type appeared only several centuries ago, lettering—whether using chisel, or pen, or brush, or stylus, or ... —is much older. The craft of putting together beautiful letters to express a thought is ancient. Its terminology developed over a long time, and may confuse or intimidate a novice. Take an individual letter, for example. Like a strange animal, a letter can have an *arm*, a *leg*, a *foot*, an *eye*, *joints*, a *chin*, an *ear*, a *tail*, a *shoulder* or even a *crotch*. Like a plant, it can have a *stem*. Like a bus system, it has *terminals*. And like a London bus, it can be *single-story* or *double-story*.

The advent of computers led to a democratization of typography: now anybody can create pages without spending years learning calligraphy or considerable money buying a letterpress. Thus some amount of typographic literacy is needed for many people, if only to be able to talk to a web designer while setting up a personal or corporate site. While there are many books explaining the basics of typography to novices, a free online dictionary is always welcome.

A glossary of typographic terms by Janie Kliever, published online at <https://designschool.canva.com/blog/typography-terms/> by Canva, tries to provide such dictionary. It has 34 short entries, with a minimal amount of text. Instead, the author uses tastefully designed drawings, like the ones on Figures 1–4.

The dictionary is intended for novices. One of the comments on the site says,

Oh my God. I'm obsessed with typography but have no idea there are terms for it. I love this.

Nevertheless, the field of typography may have surprises for everybody, like *gadzook* meaning a part of a ligature which does not belong to constituting letters (Figure 4).

Janie Kliever has created a useful and beautiful site, which deserves to be among typophiles' bookmarks.

◊ Boris Veytsman  
 Systems Biology School and  
 Computational Materials  
 Science Center, MS 6A2  
 George Mason University  
 Fairfax, VA 22030  
 borisv (at) lk dot net  
<http://borisv.lk.net>



Figure 1: Slanted and italic



Figure 2: Swash



Figure 3: X-height



Figure 4: Ligature

## BachoTEX 2015 proceedings

The BachoTEX 2015 proceedings was published by GUST, the Polish language TeX user group ([gust.org.pl](http://gust.org.pl)). The conference web page is [gust.org.pl/bachotex/2015-en](http://gust.org.pl/bachotex/2015-en).

RYSZARD KUBIAK, Z LATEXa na Sphinksa—doświadczenie konwersji [LATEX to Sphinx—a conversion experience]; pp. 5–12

A thousand-page user manual for a large software system was converted from LATEX to Sphinx. The goal was to get the documentation in a nice notation, from which good quality HTML and PDF (via TeX) can be easily generated. It took several months for the conversion to be completed. Why it took so long, what was difficult and what was easy, what lessons it taught—these and other topics are presented in the paper.

JEAN-MICHEL HUFFLEN, From MIBIBTEX 1.3 to 1.4; pp. 13–17

Since last year, we implemented new features from the present version of MIBIBTEX (1.3), such as inexact years for dates, and additional checking for person and journal names. We summarize them briefly and explain that the next step (version 1.4) would cause important change because of dealing with different encodings. We show how this upgrade to a major release has been planned.

JEAN-MICHEL HUFFLEN, How to accompany a popular song? Some bases and tips; pp. 18–22

We show some basic tips about accompanying a popular song. We present basic chords, and simple ways to arrange them. These general tips could be easily adapted to specific instruments such as the piano, the guitar, and double-bass. Examples coming from popular Polish music will be demonstrated and be subjects of exercises. Attending this workshop only requires basic knowledge of music, that is, reading notes on staves.

PAWEŁ ŁUPKOWSKI, Pakietomat—a crowd-sourced LATEX package documentation in Polish; pp. 23–25

This paper presents the project of crowd-sourced LATEX package documentation in Polish. The basic idea is presented, and certain specific solutions are discussed. Also, selected statistics for the project (such as the number of views, users and the most popular packages) are presented.

LUIGI SCARSO, MFLUA 0.5; pp. 26–30

This paper introduces MFLUA 0.5, the first public version of a new implementation of METAFONT which embeds a Lua interpreter to avoid requiring

analysis of the log in order to extrapolate information about the outlines and bitmaps of the glyphs. Some examples of the new possibilities are shown.

VALENTINAS KRIAUCIUKAS, LUKAS RAZINKOVAS and LOLITA ŽAMOITINAITĖ, Parsing LATEX for LATEX; pp. 31–36

When you need a tool to find or change something in LATEX source, you need a LATEX parser. To build such, you must know the syntax of LATEX commands and deal somehow with author's macros. Even with all that knowledge, you need to decide when to stop the improvement loop of TeX engine simulation. In one of our projects, related to computer linguistics, the participants wrote three different LATEX parsers and used a fourth one. We will present our work together with the current statistics. A LATEX parser has several use cases of general interest, like spelling or previewing, which we will briefly consider. The main goal is to discuss components needed for a successful parser project and how a TeX engine could be involved in the parsing (native LATEX parser?).

KRZYSZTOF PSZCZOŁA, O projekcie *progression transformed into art* [On the project ‘progression transformed into art’]; pp. 37–41

The ‘progression transformed into art’ project is about graphics which possess these three properties:

- they visualize certain progressions (until now I have been using arithmetic and geometric progressions but the project is evolving),
- are generated with short and elegant MetaPost code (that's a working assumption, but as happens with assumptions, realizations may differ),
- are visually attractive (that's again an assumption; see above).

I will present the graphics created until now. For some of them, I will also present how the results arrived at their final forms.

The first graphics were created during 2014; I intend to continue in this vein. The project's web page is being developed, currently at <http://pro-trans.tumblr.com>.

Additionally, the thoughts arising while working on the project led me to an idea of teaching basics of programming to elementary and secondary school pupils; co-authors of such a schoolbook are welcome.

PAWEŁ ŁUPKOWSKI, Making your researcher's life easier: How to prepare transparent and dynamic research reports with LATEX; pp. 42–48

In my talk I will present LATEX packages that make preparing a research report (containing widely-used data analyses) easier. We aim at research reports that are transparent and dynamic. By transparent I mean a report whose source code contains

the information about the operations performed on a raw data. Such a solution allows for easy grasping the idea behind the analysis for author and reviewer as well as for collaborators or students. As for the dynamic aspect, the idea is that it should generate the resulting values on a basis of an external raw data file. Thanks to this, each time the raw data change, the output values in the report will also change automatically (thus freeing the author from the worry of updating the values manually). I will present the following packages: `datatool`, `exceltex`, `sweave` and `embedfile`.

GRZEGORZ MURZYNOWSKI, GMOA, the ‘General Manipulation Of Arguments’: an extension to the `l3expan` package of the `expl3` bundle and language; pp. 49–56

[Printed in this issue of *TUGboat*.]

HANS HAGEN, Exporting XML and ePub from ConTeXt; pp. 57–60

[Printed in *TUGboat* 36:1.]

JEAN-MICHEL HUFFLEN, Musical typography’s dimensions; pp. 61–65

In comparison to reading a book—which is a linear process—reading a musical score may induce advanced processes, which influence the interpretation of musical notations. In addition, getting inputs for musical scores seems to be difficult to implement outside interactive systems, that is, WYSIWYG. Some WYSIWYM systems, such as MusiXTEX or LilyPond, have been put into action, but they are not accurate for *any* kind of music. We explore these *dimensions* of reading music and explain why musical typography should implement many facets of musical processes.

DAMIEN THIRIET, Od bépo do nozak: praca nad polską ergonomiczną klawiaturą [From bépo to nozak: working on a Polish ergonomic keyboard]; pp. 66–70

My discovery of the French ergonomic keyboard bépo (<http://www.bepo.fr>) fundamentally changed my approach to working with the computer. I not only was encouraged to invest in better equipment (a matrix keyboard), I also quickly started to work on my own version of the layout for writing in the French and Polish languages, the bépo-pl ([http://bepo.fr/wiki/Utilisateur:Damien\\_thiriet](http://bepo.fr/wiki/Utilisateur:Damien_thiriet)). Recently I’ve realized that it would be better to write with a clean Polish language layout, which I call nozak. In the presentation I will show that layout and discuss the underlying assumptions.

[Received from Tomasz Przechlewski.]

## Die *TeXnische Komödie* 4/2015

Die *TeXnische Komödie* is the journal of DANTE e.V., the German-language TeX user group (<http://www.dante.de>). (Non-technical items are omitted.)

PETER ZIMMERMANN, TeX im Barock. Bericht vom 13. Bayerischen—TeX-Stammtisch am 1. und 2. August 2015 in Eichstätt [TeX in the baroque era. Report from the 13th Bavarian TeX summit on August 1st and 2nd in Eichstätt]; pp. 14–17

The 13<sup>th</sup> Bavarian TeX summit, a joint meeting of Erlangen/Nürnberg and Munich regulars, brought 18 participants to Eichstätt, a town dominated by buildings from the baroque era.

STEPHAN LUKASCZYK, Rückblick: DANTE-Herbsttagung in Graz [DANTE fall meeting in Graz]; pp. 18–23

The DANTE e.V. annual meeting took place in Graz this year. From September 4th to 6th, TeX enthusiasts from Austria and Germany met in the capital of Steiermark to discuss all kinds of TeX-related items.

DOMINIK WAGENFÜHR, Mit LATEX zum E-Book [From LATEX to ebook]; pp. 25–65

Ebook readers and other mobile devices gain more and more popularity. Therefore, LATEX authors also want to publish their works in EPUB and similar formats. This article presents eight workflows to create ebooks, following Christine Römer’s article on this topic in DTK 1/2015.

HERBERT VOSS, Ausgabe einer LATEX-Datei im Format EPUB [Creating an EPUB file from LATEX]; pp. 65–69

`tex4ebook` is a new package (<http://ctan.org/pkg/tex4ebook>) that can create EPUB, a common ebook format. Its capabilities are, however, limited to documents with a not-too-complicated LATEX structure.

ROLF NIEPRASCHK, Ausgabe der Paketliste von TeX Live mit LuaTeX [Creating TeX Live’s package list with LuaTeX]; pp. 70–72

In an earlier article we showed how with the help of a Unix shell script a TeX file can be created that shows all installed TeX Live packages. In this article we show how the same can be achieved with LuaTeX and some Lua code.

[Received from Herbert Voß.]



JimBentonGOcomics.com

© Jim Benton  
by permission of the artist

## TUG Institutional Members

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

American Mathematical Society,  
*Providence, Rhode Island*  
 Aware Software, Inc.,  
*Midland Park, New Jersey*  
 Center for Computing Sciences,  
*Bowie, Maryland*  
 CSTUG, *Praha, Czech Republic*  
 Fermilab, *Batavia, Illinois*  
 Google, *San Francisco, California*  
 IBM Corporation,  
 T J Watson Research Center,  
*Yorktown, New York*

Institute for Defense Analyses,  
 Center for Communications  
 Research, *Princeton, New Jersey*  
 Maluhy & Co., *São Paulo, Brazil*  
 Marquette University,  
*Milwaukee, Wisconsin*  
 Masaryk University,  
 Faculty of Informatics,  
*Brno, Czech Republic*  
 MOSEK ApS,  
*Copenhagen, Denmark*  
 New York University,  
 Academic Computing Facility,  
*New York, New York*  
 River Valley Technologies,  
*Trivandrum, India*  
 River Valley Technologies,  
*London, United Kingdom*  
 RSGP Consulting Pvt. Ltd.,  
*Trivandrum, India*  
 ShareLaTeX, *United Kingdom*  
 Springer-Verlag Heidelberg,  
*Heidelberg, Germany*  
 StackExchange,  
*New York City, New York*  
 Stanford University,  
 Computer Science Department,  
*Stanford, California*  
 Stockholm University,  
 Department of Mathematics,  
*Stockholm, Sweden*  
 TNQ, *Chennai, India*  
 University College, Cork,  
 Computer Centre,  
*Cork, Ireland*  
 Université Laval,  
*Ste-Foy, Québec, Canada*  
 University of Cambridge,  
 Centre for Mathematical Sciences,  
*Cambridge, United Kingdom*  
 University of Ontario,  
 Institute of Technology,  
*Oshawa, Ontario, Canada*  
 University of Oslo,  
 Institute of Informatics,  
*Blindern, Oslo, Norway*  
 VTeX UAB, *Vilnius, Lithuania*

## 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 <http://tug.org/consultants.html>. If you'd like to be listed, please see that web page.

### Aicart Martínez, Mercè

Tarragona 102 4<sup>o</sup> 2<sup>a</sup>  
08015 Barcelona, Spain  
+34 932267827  
Email: [m.aicart \(at\) ono.com](mailto:m.aicart(at)ono.com)  
Web: <http://www.edilatex.com>

We provide, at reasonable low cost, L<sup>A</sup>T<sub>E</sub>X or T<sub>E</sub>X page layout and typesetting services to authors or publishers world-wide. We have been in business since the beginning of 1990. For more information visit our web site.

### Dangerous Curve

PO Box 532281  
Los Angeles, CA 90053  
+1 213-617-8483  
Email: [typesetting \(at\) dangerouscurve.org](mailto:typesetting(at)dangerouscurve.org)  
Web: <http://dangerouscurve.org/tex.html>

We are your macro specialists for T<sub>E</sub>X or L<sup>A</sup>T<sub>E</sub>X fine typography specs beyond those of the average L<sup>A</sup>T<sub>E</sub>X macro package. If you use X<sub>E</sub>T<sub>E</sub>X, we are your microtypography specialists. We take special care to typeset mathematics well.

Not that picky? We also handle most of your typical T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X 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<sub>E</sub>X book.

### Latchman, David

4113 Planz Road Apt. C  
Bakersfield, CA 93309-5935  
+1 518-951-8786  
Email: [david.latchman \(at\) technical-designs.com](mailto:david.latchman(at)technical-designs.com)  
Web: <http://www.technical-designs.com>

L<sup>A</sup>T<sub>E</sub>X consultant specializing in: the typesetting of books, manuscripts, articles, Word document conversions as well as creating the customized packages to meet your needs.

Call or email to discuss your project or visit my website for further details.

### Peter, Steve

+1 732 306-6309  
Email: [speter \(at\) mac.com](mailto:speter(at)mac.com)

Specializing in foreign language, multilingual, linguistic, and technical typesetting using most flavors of T<sub>E</sub>X, 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<sub>E</sub>X-based publishing systems. I have an MA in Linguistics from Harvard University and live in the New York metro area.

### Sievers, Martin

Im Alten Garten 5  
54296 Trier, Germany  
+49 651 4936567-0  
Email: [info \(at\) schoenerpublizieren.com](mailto:info(at)schoenerpublizieren.com)  
Web: <http://www.schoenerpublizieren.com>

As a mathematician with more than ten years of typesetting experience I offer T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X services and consulting for the whole academic sector (individuals, universities, publishers) and everybody looking for a high-quality output of his documents. From setting up entire book projects to last-minute help, from creating individual templates, packages and citation styles (BIBT<sub>E</sub>X, biblatex) to typesetting your math, tables or graphics—just contact me with information on your project.

### Sofka, Michael

8 Providence St.  
Albany, NY 12203  
+1 518 331-3457  
Email: [michael.sofka \(at\) gmail.com](mailto:michael.sofka(at)gmail.com)

Skilled, personalized T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X consulting and programming services.

I offer over 25 years of experience in programming, macro writing, and typesetting books, articles, newsletters, and theses in T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X: Automated document conversion; Programming in Perl, C, C++ and other languages; Writing and customizing macro packages in T<sub>E</sub>X or L<sup>A</sup>T<sub>E</sub>X; Generating custom output in PDF, HTML and XML; Data format conversion; Databases.

If you have a specialized T<sub>E</sub>X or L<sup>A</sup>T<sub>E</sub>X need, or if you are looking for the solution to your typographic problems, contact me. I will be happy to discuss your project.

### Veytsman, Boris

46871 Antioch Pl.  
Sterling, VA 20164  
+1 703 915-2406  
Email: [borisv \(at\) lk.net](mailto:borisv(at)lk.net)  
Web: <http://www.borisv.lk.net>

T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X consulting, training and seminars. Integration with databases, automated document preparation, custom L<sup>A</sup>T<sub>E</sub>X packages, conversions and much more. I have about eighteen years of experience in T<sub>E</sub>X and three decades of experience in teaching & training. I have authored several packages on CTAN, published papers in T<sub>E</sub>X related journals, and conducted several workshops on T<sub>E</sub>X and related subjects.

### Young, Lee A.

127 Kingfisher Lane  
Mills River, NC 28759  
+1 828 435-0525  
Email: [leeayoung \(at\) morrisbb.net](mailto:leeayoung(at)morrisbb.net)  
Web: <http://www.thesiseditor.net>

Copyediting your .tex manuscript for readability and mathematical style by a Harvard Ph.D. Your .tex file won't compile? Send it to me for repair. Experience: edited hundreds of ESL journal articles, economics and physics textbooks, scholarly monographs, L<sup>A</sup>T<sub>E</sub>X manuscripts for the Physical Review; career as professional, published physicist.

## Calendar

### **2015**

- Oct 13 – Dec 20 Ada Lovelace, Celebrating 200 years of a computer visionary, Oxford, UK.  
[blogs.bodleian.ox.ac.uk/  
adalovelace/events/](http://blogs.bodleian.ox.ac.uk/adalovelace/events/)

### **2016**

- Feb 25–27 Typography Day 2016,  
 “Typography and Education”,  
 Srishti School of Art, Design & Technology,  
 Bangalore, India. [www.typoday.in](http://www.typoday.in)
- Mar 11 TUGboat **37:1**, submission deadline.
- Mar 30 – Apr 1 DANTE Frühjahrstagung and  
 54<sup>th</sup> meeting, “TeX and Schools”,  
 Bergische Universität,  
 Wuppertal, Germany.  
[www.dante.de/events.html](http://www.dante.de/events.html)
- Apr 29 – May 3 BachoTeX 2016:  
 24<sup>th</sup> BachoTeX Conference,  
 Bachotek, Poland.  
[www.gust.org.pl/bachotex/2016](http://www.gust.org.pl/bachotex/2016)
- May 12–14 TYPO Berlin 2016, “Beyond  
 Design”, Berlin, Germany.  
[typotalks.com/berlin/](http://typotalks.com/berlin/)
- Jul 4–7 Book history workshop, École de  
 l'institut d'histoire du livre,  
 Lyon, France. [ihl.enssib.fr](http://ihl.enssib.fr)
- Jul 5–9 The 6<sup>th</sup> International Conference on  
 Typography and Visual Communication  
 (ICTVC), “Against lethe ...”,  
 Thessaloniki, Greece. [www.ictvc.org](http://www.ictvc.org)
- Jul 12–16 Digital Humanities 2016, Alliance of  
 Digital Humanities Organizations,  
 “Digital Identities: the Past and the  
 Future”, Kraków, Poland. [dh2016.org](http://dh2016.org)

- Jul 18–22 SHARP 2016, “The Generation and  
 Regeneration of Books”. Society for  
 the History of Authorship, Reading  
 & Publishing, “Languages of  
 the Book”/“Les langues du livre”.  
 Paris, France. [www.sharpparis2016.com](http://www.sharpparis2016.com)

### **TUG 2016**

#### **Toronto, Canada.**

- Jul 23, 24 Optional pre-conference tours.
- Jul 24 Evening reception and registration.
- Jul 25–27 The 37<sup>th</sup> annual meeting of the  
 TeX Users Group.  
 Presentations covering the TeX world.  
[tug.org/tug2016](http://tug.org/tug2016)
- Jul 28 Typographic excursions and banquet.
- Jul 29 Optional post-conference tour [potential].

- Jul 24–28 SIGGRAPH 2015, “Render the Possibilities”,  
 Anaheim, California.  
[s2016.siggraph.org](http://s2016.siggraph.org)
- Aug 1–5 Balisage: The Markup Conference,  
 Washington, DC. [www.balisage.net](http://www.balisage.net)
- Sep 13–16 ACM Symposium on Document  
 Engineering, Vienna, Austria.  
[www.doceng2016.org](http://www.doceng2016.org)
- Sep 14–18 Association Typographique Internationale  
 (ATypI) annual conference, Warsaw,  
 Poland. [www.atypi.org](http://www.atypi.org)
- Sep 16 The Updike Prize for Student Type  
 Design, application deadline, 5:00 p.m.  
 EST. [www.provlib.org/updikeprize](http://www.provlib.org/updikeprize)
- Sep 25 – Oct 1 10<sup>th</sup> International  
 ConTeXt Meeting, “Piece of Cake”,  
 Kalenberg, The Netherlands.  
[meeting.contextgarden.net/2016](http://meeting.contextgarden.net/2016)

*Status as of 31 October 2015*

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](mailto: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](http://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](http://tug.org/calendar.html).

---

**Until the end of 2015, a TUG membership drive continues!**  
**Invite friends, win prizes—info at [tug.org/membership](http://tug.org/membership).**

**Introductory**

- 180 *Barbara Beeton* / Editorial comments
  - typography and *TUGboat* news
- 200 *Taco Hoekwater* / History of cookbooks
  - from ancient Greece to [allrecipes.com](#)
- 188 *Simon Laube* / *TeX* in schools: Just Say Yes!
  - benefits of using *L<sup>A</sup>T<sub>E</sub>X* in schools, for students and teachers
- 214 *Thomas Thurnherr* / Introduction to list structures in *L<sup>A</sup>T<sub>E</sub>X*
  - basic list usage and packages for additional control

**Intermediate**

- 269 *Karl Berry* / The treasure chest
  - new CTAN packages, March–October 2015
- 191 *Charles Bigelow* / About the DK versions of Lucida
  - squarish capital ‘O’ for Don Knuth, in Lucida Grande Mono and Lucida Console
- 208 *Peter Flynn* / Typographers’ Inn
  - usability of digital typography; hierarchy and balance; Emacs on Android
- 217 *Anagha Kumar* / *gradstudentresume*: A document class for graduate student CVs
  - a new class for academic CVs, and practical tips on creating classes
- 210 *L<sup>A</sup>T<sub>E</sub>X Project Team* / *L<sup>A</sup>T<sub>E</sub>X* news, issue 22, January 2015
  - new *L<sup>A</sup>T<sub>E</sub>X2<sub>E</sub>* bug-fix policy, updates to the kernel, hyperlinked documentation
- 212 *L<sup>A</sup>T<sub>E</sub>X Project Team* / *L<sup>A</sup>T<sub>E</sub>X* news, issue 23, October 2015
  - enhanced support for *LuaTeX*, more floats and inserts, updated Unicode data, pre-release releases
- 190 *Linus Romer* / Smoky letters
  - randomized but still elegant capital D for a copperplate ‘Danke’

**Intermediate Plus**

- 268 *Karl Berry* / Production notes
  - editing Unicode text requiring a special font for PUA characters in Emacs
- 227 *Clemens Niederberger* / Chemistry in *L<sup>A</sup>T<sub>E</sub>X2<sub>E</sub>*—an overview of existing packages and possibilities
  - overview of *chemmacros*, *mhchem*, *chemformula*, and more
- 220 *Peter Wilson* / Glisterings: Longest string; Marching along; A blank argument; A centered table of contents
  - even/odd arguments, list indexing, *memoir* toc typesetting, and more
- 234 *Joseph Wright* / Beamer overlays beyond the *\visible*
  - generalized slide overlays for *only*, *alert*, and other operations

**Advanced**

- 257 *Grzegorz Murzynowski* / GMOA, the ‘General Manipulation Of Arguments’:
  - An extension to the *l3expan* package of the *expl3* bundle and language
  - a DFA-based generalization of *expl3*’s *l3expan*
- 237 *Luigi Scarso* / Two applications of SWIGLIB: GraphicsMagick and Ghostscript
  - loading binary modules for performance and generality
- 243 *Udo Wermuth* / Typesetting the “Begriffsschrift” by Gottlob Frege in plain *TeX*
  - handling the unusual mathematical notation invented by Frege

**Contents of other *TeX* journals**

- 272 *EuroBachotEX* 2015; *Die T<sub>E</sub>Xnische Komödie* 4/2015

**Reports and notices**

- 179 *TUG Board* / From the Board of Directors
  - suspension of the TUG President
- 182 *Norbert Preining* / Adrian Frutiger, 1928–2015
- 184 *Joachim Schrod* / Thomas Koch, 1964–2014
- 185 *Stefan Kottwitz* / DANTE e.V. 2015 meeting reports
- 271 *Boris Veytsman* / An online glossary of typographic terms by Janie Kliever
  - review of this pictorial glossary
- 274 *Jim Benton* / A summons
- 274 Institutional members
- 275 *TeX* consulting and production services
- 276 Calendar