TUGboat

Volume 28, Number 2 / 2007

TEX Users Group

TUGboat (ISSN 0896-3207) is published by the TEX Users Group.

Memberships and Subscriptions

2007 dues for individual members are as follows:

- Ordinary members: $85.
- Students/Seniors: $45.

The discounted rate of $45 is also available to citizens of countries with modest economies, as detailed on our web site.

Membership in the TEX Users Group is for the calendar year, and includes all issues of *TUGboat* for the year in which membership begins or is renewed, as well as software distributions and other benefits. Individual membership 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.

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 is $95 per year, including air mail delivery.

Institutional Membership

Institutional membership is a means of showing continuing interest in and support for both TEX and the TEX Users Group, as well as providing a discounted group rate and other benefits. For further information, contact the TUG office or see our web site.

TEX is a trademark of the American Mathematical Society.

Printed in U.S.A.

Addresses

General correspondence, payments, etc.
TEX Users Group
P. O. Box 2311
Portland, OR 97208-2311
U.S.A.

Delivery services, parcels, visitors
TEX Users Group
1466 NW Naito Parkway
Suite 3141
Portland, OR 97209-2820
U.S.A.

Telephone

+1 503 223-9994

Fax

+1 206 203-3960

Electronic Mail (Internet)

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

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

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

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

World Wide Web

`http://www.tug.org/`
`http://www.tug.org/TUGboat/`

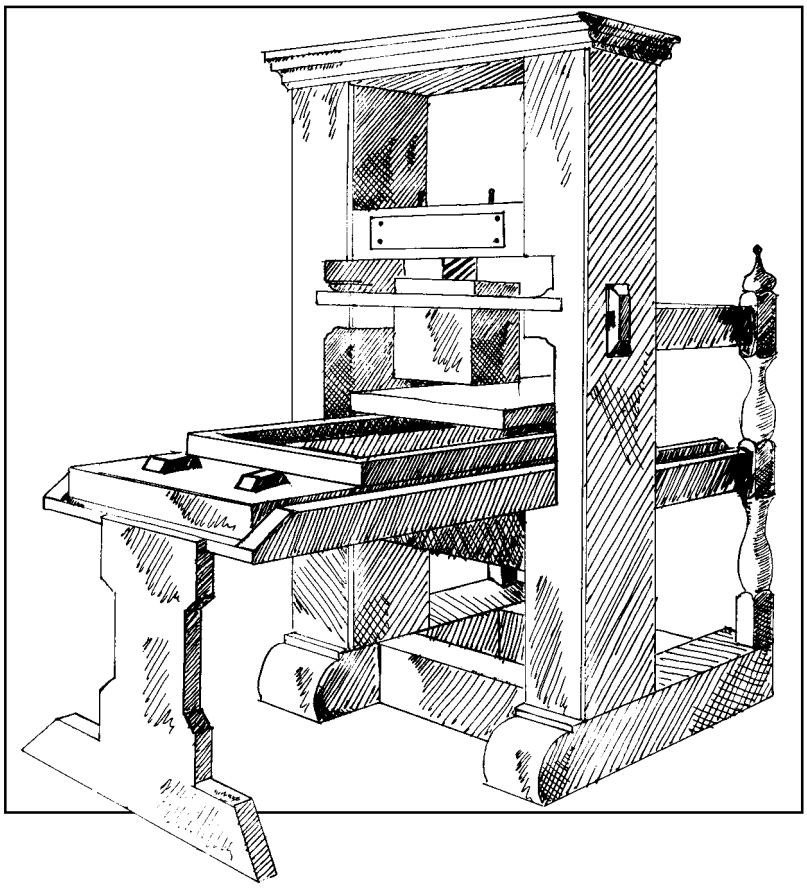
Have a suggestion? Problems not resolved?

The TUG Board wants to hear from you:
Please email `board@tug.org`.

[printing date: July 2007]

TUGBOAT

The Communications of the TEX Users Group



Volume 28, Number 2, 2007

TEX Users Group

TUGboat (ISSN 0896-3207) is published by the TEX Users Group.

Memberships and Subscriptions

2007 dues for individual members are as follows:

- Ordinary members: $85.
- Students/Seniors: $45.

The discounted rate of $45 is also available to citizens of countries with modest economies, as detailed on our web site.

Membership in the TEX Users Group is for the calendar year, and includes all issues of *TUGboat* for the year in which membership begins or is renewed, as well as software distributions and other benefits. Individual membership 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.

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 is $95 per year, including air mail delivery.

Institutional Membership

Institutional membership is a means of showing continuing interest in and support for both TEX and the TEX Users Group, as well as providing a discounted group rate and other benefits. For further information, contact the TUG office or see our web site.

TEX is a trademark of the American Mathematical Society.

Printed in U.S.A.

Addresses

General correspondence, payments, etc.
TEX Users Group
P. O. Box 2311
Portland, OR 97208-2311
U.S.A.

Delivery services, parcels, visitors
TEX Users Group
1466 NW Naito Parkway
Suite 3141
Portland, OR 97209-2820
U.S.A.

Telephone

+1 503 223-9994

Fax

+1 206 203-3960

Electronic Mail (Internet)

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

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

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

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

World Wide Web

`http://www.tug.org/`
`http://www.tug.org/TUGboat/`

Have a suggestion? Problems not resolved?

The TUG Board wants to hear from you:
Please email `board@tug.org`.

[printing date: July 2007]

A designer knows he has achieved perfection not when there is nothing left to add but when there is nothing left to take away.

Antoine de Saint-Exupery
Wind, Sand, and Stars (1939),
translation by Lewis Galantiere

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP
EDITOR BARBARA BEETON

VOLUME 28, NUMBER 2 • 2007
PORTLAND • OREGON • U.S.A.

TUGboat

This regular issue (Vol. 28, No. 2) is the second issue of the 2007 volume year. No. 3 will contain the TUG 2007 (San Diego) proceedings.

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 an issue's 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 still 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

TUGboat will be publishing one issue of conference proceedings in 2007. The deadline to receive the final papers for that issue is August 17, 2007.

Links, locations, and more information about this and all conferences are available at `http://tug.org/meetings.html`.

As always, suggestions and proposals for *TUGboat* articles are gratefully accepted and processed as received. We encourage submitting contributions by electronic mail to `TUGboat@tug.org`.

The *TUGboat* "style files", for use with either `plain` TEX or LATEX, are available from CTAN and the *TUGboat* web site. We also accept submissions using ConTEXt.

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. If you have any reservations about posting online, please notify the editors at the time of submission.

Other TUG Publications

TUG is interested in considering additional manuscripts for publication. These might include manuals, instructional materials, documentation, or works on any other topic that might be useful to the TEX community in general. Provision can be made for including macro packages or software in computer-readable form.

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

TUGboat Advertising

For information about advertising rates and options, including consultant listings, write or call the TUG office, or see our web page:
`http://tug.org/TUGboat/advertising.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 appear in this issue should not be considered complete.

METAFONT is a trademark of Addison-Wesley Inc.
PostScript is a trademark of Adobe Systems, Inc.
TEX and $\mathcal{AMS}$-TEX are trademarks of the American Mathematical Society.

General Delivery

From the President

Karl Berry, president (at) tug dot org

TEX development grants

The most exceptional recent news is about funding for TEX development. We are very excited and grateful to announce that an anonymous foundation has contributed $10,000 to TUG to be used to encourage advances in TEX development.

Furthermore, our generous donor has made a second $10,000 available as matching funds, and so we are seeking other contributors, both institutional and individual, to contribute a matching $10,000 (thus making a total of $30,000 available). Early matching contributions have already reached $5,000 — we thank CAPDM Limited and many other donors for their support. To contribute, please see https://www.tug.org/donate/dev.html.

We expect to use at least some of these funds to augment existing efforts, rather than starting anything major from scratch. TUG's TEX Development Fund committee will have the responsibility for making allocations. As part of the gift, we were asked to develop a basic roadmap for future TEX development to help guide where the contributions are spent. The committee has already solicited input from some of the major extant TEX development projects, and will be soliciting additional input.

At the donor's suggestion, we will also be considering ways to improve the TEX experience specifically for "ordinary" users, such as students. This could include enhancements in front end software, new or improved documentation at that level, and other projects.

We are enthusiastic about this opportunity, and very much hope, as does our donor, that the results will be beneficial for all TEX users.

This is also an appropriate place to mention again our appreciation for the grant from Colorado State University, made to Professor Idris Samawi Hamid, which provided major funding for LuaTEX, as detailed in the last *TUGboat*.

http://mirror.ctan.org

Although CTAN has many mirrors around the world, the main way they have been used is by manually choosing one — so, more often than not, the backbone servers run by DANTE, UK-TUG and TUG end up getting used. With the surge in downloads from the new releases of the biggest distributions (TEX Live 2007, MacTEX 2007, proTEXt, MiKTEX), the backbones have had to limit traffic rates.

The CTAN team and Randy Kobes at the University of Winnipeg have co-operated on developing a convenient method for using mirrors. Now, http://mirror.ctan.org/some/ctan/dir will *automatically* redirect to a (hopefully) near-by mirror. For example, http://mirror.ctan.org/systems/texlive (a /tex-archive is optional).

The choice of mirror is based on the originating IP address, out of the pool of mirrors that are known to be up-to-date. Of course, if there are problems of whatever kind, you can still choose a mirror yourself; the canonical list is at http://www.ctan.org/tex-archive/CTAN.sites.

Incidentally, Randy has created the same service for the GNU mirrors, accessible via http://ftpmirror.gnu.org.

Thanks to Randy and CTAN for making this available. We plan to use the new mirror.ctan.org url in *TUGboat*, where appropriate.

Interview corner

The Interview Corner on the TUG web site (http://tug.org/interviews) continues to grow. Recent interviews come from a wide variety of areas in the TEX world:

- David Carlisle, a member of the LATEX team and participant in math-related W3C activities.
- Jin-Hwan Cho, maintainer of dvipdfmx and a strong supporter of TEX in Korea.
- John Culleton, who runs an indexing and typesetting business using TEX.
- David Fuchs, one of the original members of the Stanford TEX project.
- Jonathan Kew, author of XETEX.
- Haruhiko Okumura, who promotes and supports the use of TEX in Japan.
- Will Robertson, a relatively new contributor in the TEX community.
- Nicola Talbot, a LATEX user, teacher, and package writer.

TUG 2007

The TUG 2007 conference will take place in San Diego, California, from July 17–20, at San Diego State University. For the schedule, registration and accommodation information (inexpensive on-campus housing is available), and more, visit the conference web site at http://tug.org/tug2007. Please join us and an excellent line-up of workshops and speakers. Hope to see you there!

Editorial comments

Barbara Beeton

A pledge of support

The American Mathematical Society has a policy of keeping in print "classic" books formerly published by the Chelsea Press. One such book, *Introduction to Complex Analysis*, by Rolf Nevanlinna and Veikko Paatero, is about to be republished. As both authors are deceased, their heirs have expressed the wish, formally agreed for the period of one year from the publication date, that the royalties be donated "to an organization that advances techniques for disseminating and publishing mathematical and scientific information, for example to the TEX Users Group". We thank the authors' heirs for their support.

Helvetica — 50th anniversary

2007 is the 50th anniversary of the font Helvetica. Developed at the Haas Type Foundry in Münchenstein, Switzerland, by Max Miedinger and Edward Hoffman, the font was originally called "Neue Haas Grotesk", as a revival of older sans serif, or grotesk, typefaces. Looking toward international distribution, it was renamed *Helvetica*, derived from *Helvetia*, the Latin name for Switzerland.

The anniversary is honored by a feature-length documentary film directed by Gary Hustwit. It has been screening at film festivals, museums, art schools and other venues worldwide. I had the pleasure of viewing it at the Rhode Island School of Design in April, where it was followed by a Q & A session with the director. I greatly enjoyed it, and encourage anyone with even a smidgen of interest in type to see it.

The film has its own web page, with a schedule of screenings, clips from the film, and other information, at `http://www.helveticafilm.com`.

Two other sites recognizing this milestone are Slate at `http://www.slate.com/id/2166887/`, and the *Toronto Globe and Mail* at `http://www.theglobeandmail.com/servlet/story/RTGAM.20070418.whelvetica18/BNStory`.

Happy anniversary, Helvetica!

Another font anniversary — Souvenir, 93 years

Souvenir is more often associated with the 1960s and '70s, but it was actually designed in 1914 by Morris Fuller Benton, director of typeface development for the American Type Founders Company. It wasn't particularly successful at its first release, but after a reimplementation as photo lettering in 1967, with complementary italic designs drawn by Ed Benguiat, it became one of the most popular offerings of ITC (the International Typeface Corporation). Overuse made it a pariah for a while during the latter years of the 20th century, but it is being re-examined with a new appreciation for the wide range of possible applications.

Souvenir is highlighted at `http://www.fonts.com/FindFonts/HiddenGems/ITCSouvenir.htm`.

Another honorary doctorate for Don Knuth

On October 29–31, the University of Bordeaux is organizing a colloquium in honor of Donald Knuth, with an honorary doctorate Honoris Causa to be presented on October 30.

Several notable speakers have been invited; the topic of the conference is algorithms. For more information, go to `http://knuth07.labri.fr/`.

How to shrink a box as much as possible

Here's a "pracniqne" from Don, "that may or may not be well known to TEXies:"

> If you want to shrink a box as much as possible, in order to see why TEX refuses to typeset something on a single line (even with `\looseness=-1`), you can use the "spread" feature. For example, try
>
> ```
> \hbox{a little test}
> \hbox spread-1pt{a little test}
> \hbox spread-2pt{a little test}
> \hbox spread-3pt{a little test}
> \hbox spread-4pt{a little test}
> ```
>
> In general you can say `\hbox spread-100pt`, say; you'll get an overfull box, but the amount by which it's overfull tells you how close you came. And by eyeballing the result, you may be able to figure out where it's safe to kern away some space.

How to use a book

It's the middle of the fourteenth century, and you have just obtained an example of the latest technology... a book! How do you use it? Go to `http://www.devilducky.com/media/57946/` and observe a technical support session. It's a lesson you won't soon forget.

Save the signs!

Joe Clark reported on the TYPO-L list the projected demise of some historic signs designed for the Toronto Transit Commission. Some of these signs, now displayed at the St. George station, are the

only remaining examples of a redesign by the Canadian graphic designer Paul Arthur from the 1990s, when the TTC commissioned both new signage and a user test to see what worked best; all tested groups (including average riders, non-English speakers, visually impaired, and low-literacy riders) preferred the new signs. However, the redesign was never implemented, the prototype signs remained in the St. George station, and the signage in the Toronto transit system remains a hodgepodge.

Some of the stations have aged poorly, and repairs are necessary. Current plans include removal (and destruction) of both the Paul Arthur signs and others whose design is unique.

It would be quite feasible, given cooperation from the TTC, to preserve most of the signs as museum pieces, but public support is needed to encourage the Commissioners to prevent the destruction. Historical notes and details of the write-in campaign are posted at `http://joeclark.org/TTC/`.

Practical TEX 2006 recordings

Kaveh Bazargan and his colleagues at River Valley Technologies have created a multimedia presentation of the talks from the Practical TEX conference held at the Busch Campus of Rutgers University in midsummer 2006.

The recording was experimental, and assorted hardware problems prevented completion of all presentations, but most are there with both audio and video.

The recordings are at `http://river-valley.dreamhosters.com/practex2006/`.

Many thanks to Kaveh for envisioning and carrying out this project. He hopes to do the same for this year's TUG meeting.

◇ Barbara Beeton
American Mathematical Society
201 Charles Street
Providence, RI 02904 USA
`tubboat (at) tug dot org`

A wayward wayfarer's way to TEX

Stephen Moye

1 Introduction

I flatter myself that my introduction to TEX was a bit unusual, and that it may be just entertaining enough to share with you. Let me say at the outset that my training is as a musician: specifically as an organist (rather like another very distinguished member of the TEX community...), choir director, and sometime composer — and some would say not nearly sometime enough.

I received a Bachelor of Music Degree in organ from Heidelberg College in Tiffin, Ohio, spent three years studying at the Royal College of Music, London, and am now a whisker away from a Master's degree in ethnomusicology from Brown University in Providence, Rhode Island. Everything was going along smoothly until one day in 1986 when I found myself in the electronics section of a rather fancy department store. Little did I know what an interesting turn life would take....

2 The new love of my life

I rounded a corner and came up against a rather plain display featuring a Mac Plus. Love at first sight. It was so cute! And it in no way corresponded to my preconceptions of what a computer was. It took only a second to get accustomed to the mouse, and in minutes I was drawing (admittedly crude) pictures using MacPaint. MacWrite was a revelation: I could type whatever I liked with no fear of errors as they could be corrected with a simple backspace! Cool! No more whiteout, no more punishing re-typing. Way cool! Moreover, this was my very first experience with a computer.

I am a dyed-in-the-wool Mac user: I'm unhappy if I'm not surrounded by mice, GUI's with dialogue boxes, windows, drop-down menus, and the sleek styling of the computer itself. Microsoft Windows, and the even more arcane, mysterious and mantra-ridden Unix environment are alien to me — or at least they were until a few years ago.

A few months after I saw it in the store, I had my very own Mac Plus. A few months after that,

This article is based on a talk given at PracTEX 2006. The author is an employee of the American Mathematical Society, but the opinions of the author in no way represent the opinions of the AMS.

I had an early copy of Aldus Corporation's PageMaker.[1] A few months later still, I had an opportunity to see the documents I was designing in PageMaker printed on an Apple LaserWriter. I cannot adequately convey to you the wonder of seeing that first output (it would probably embarrass me now) and the sense of empowerment that my humble little Mac Plus gave me. Not too long after that I made the acquaintance of PostScript and the extraordinary *Colophon 3 Alphabet* created by Adobe in which PostScript was made to do some wonderful things. As the original readme file put it, "Our intent in distributing this [material] is to inspire and inform" and that it did. All you needed at that time was a $7,000 Apple LaserWriter to see the results!

In 1988 Quark released XPress[2] in direct competition to PageMaker. I was interested, so I read the reviews and ran across one by Charles Seiter for *MacUser* magazine. At the very end of the review, he mentioned almost in passing with a teasing, throw-away manner that, good as the new page layout programs were, they were nothing compared to the "grand-daddy" [3] of them all, TEX, as exemplified by the program *Textures*.

3 A revelation

Shock! Why hadn't I heard of this TEX, or Textures for that matter? I had to have it! Little, of course, did I know what awaited me.

I discovered that Textures at that time was sold by Addison-Wesley. I ordered it and it arrived with more disks than I could have thought possible, a slim user's manual, and, I kid you not, a copy of *The TEXbook*. I spent an afternoon installing Textures, glanced through the user's guide, and with a trembling hand, double-clicked on the Textures icon and was greeted with... an empty text editing window. What do I do now? I turned to *The TEXbook* for some illumination. Bad move. The introduction was a nightmare of sorts: Lies? Jokes? Warning signs? It wasn't until a few weeks later that I sorted things out and actually typeset the story centering on one Mr. Drofnats.[4]

It wasn't long before I created a number of documents in TEX, PageMaker and XPress in order to compare them. Without exception, I was struck by how much better the type in the TEX sample looked. From that point on I was hooked, and a true believer.

I grew very much to enjoy TEX as embodied in Textures: In my childlike naïveté, I thought all TEX implementations must be like Textures. So, when I found out about OzTEX I got a copy of that. Boy, was that a shock. I can't recall being so befuddled by anything in my life. The whole thing was a nightmare, but fonts were the worst. Fontinst? Afm2tfm? PL files, VF files, FD files? I fled, screaming, back to the loving, comforting embrace of Textures and never looked back. I didn't know it at the time, but Textures had insulated me against the perilous font misadventures which are a distinguishing aspect of TEX.

4 And then I wrote a book

Along the way, I discovered a program called Fontographer, one of the first commercially available font editors. Yet again I was overwhelmed by a feeling of empowerment: I could do things easily and quickly with Fontographer that required tremendous resources and time in traditional type design. I digitized several Goudy typefaces that were then unavailable, and edited others to my liking.

I started to keep a notebook, a kind of *vade mecum*, in which I recorded notes about things I had discovered, ways of working, tips and techniques that I did not want to forget. Eventually I kept this information in a MacWrite document. One day I was assailed by an attack of hubris and wondered if others might be able to use this information... maybe I could write a book....

So I worked on the first few chapters, using plain TEX in Textures, and showed them to Earl Allen, head of technical support at Altsys, the company that at that time developed and marketed Fontographer. He was very enthusiastic and encouraged me to finish it and send it to a publisher. I wrote the rest of the book, and sent a sample chapter to a company that was suggested to me, MIS:Press, then a division of Holt.

A few days later I received a call from the publisher, Paul Farrell. We exchanged pleasantries and established that we liked each other, for the nonce anyway. Then the conversation took a serious turn.

He said, "Well, most people on the west coast use PageMaker and most people on the east coast use XPress, so what do you use?"

"Uh... er...," I began articulately, "well sir, I use... TEX."

[1] Remember that this was in the early, halcyon days of the Mac, when the PageMaker program resided on one *floppy*, and the system software resided on another *floppy*.

[2] Founded in 1981, Quark had in its early days developed word processor software for the Apple II and Apple III computers.

[3] Seiter's word. I tried to find the original article in the MacUser archives but was unsuccessful.

[4] Until I was preparing this presentation, I had no idea that his first name was Revinu Jitis. Live and learn.

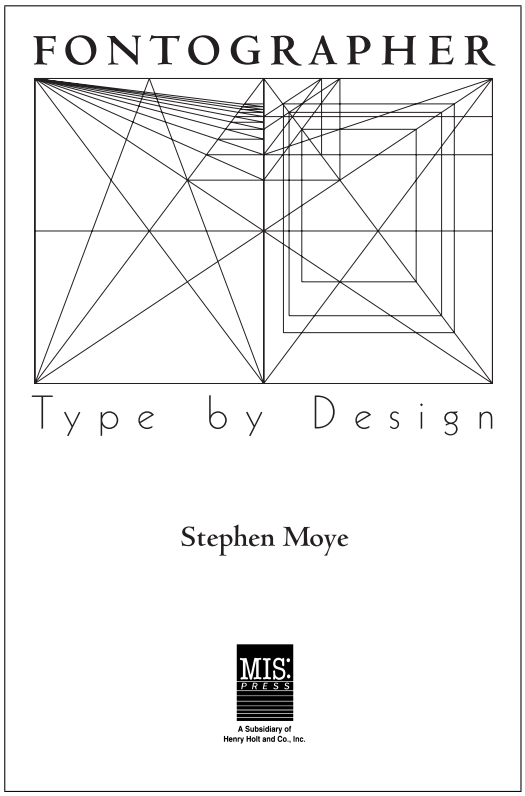


Figure 1: The book

There was a moment of silence so profound on the other end of the line that I thought the connection had been broken. I was just about to say something...

"Oh my God," he said, "bitmaps!"

It was a herculean task to convince him that TEX had evolved far beyond the use of mere bitmap fonts and that, in fact, he'd already seen the results with his own eyes in the sample I sent him. The book, *Fontographer: Type by Design*, finally appeared in 1995. Sadly, two years later, IDG International bought MIS:Press and promptly destroyed all copies of all MIS:Press books that it deemed would not sell in numbers on a par with *The Joy of Cooking* — all this without ever contacting the authors. Maybe they did me a favor of sorts: I have seen copies of the book sell for as much as $300, and an asking price over $1,000. One other note: My book and *The METAFONTbook* share the same Library of Congress call number and sit beside each other on the library shelf.

So, my world seemed happy and stable: I authored a book, typeset concert programs, church bulletins, and all kinds of documents for non-profit arts organizations. TEX was not about to go away, and Textures had become a valued tool that would be around forever. Wouldn't it?

5 And then there was Mac OS X

Apple has an annoying habit of shaking up both itself and its loyal customers. OS X is a good example of this: It is not just OS 9 in a party dress. It is radically different to the bone: So different that OS 9 programs (such as Textures, for example) have to be used in an emulation of OS 9 on OS X. Blue Sky was quick to point out that Textures works just fine under "Classic" (the name for the emulation of OS 9 in OS X) and in fact that was true, to a point. What became progressively more irritating as time went on was having to open up Classic at all. Every other piece of software I used had been rewritten for OS X. Equally irritating were Blue Sky's reassurances given over a period of *years* that an OS X version was in the works, and expected any day now.

And there were problems. It became increasingly difficult to print *reliably* from Textures. In addition, computer typography was moving to Unicode and OpenType; Textures, operating under OS 9, was stuck using standard PostScript fonts with a mere 256 characters. It is a measure of how wonderful Textures was that we stuck with it for as long as we did.

6 "Hey, kid, have I got a job for you..."

The next jog in the road of my journey in TEXland came from a most unexpected source. A choir member, Victoria Ancona, Editor of Book and Journal Production at the AMS (aka *TEX heaven*), at the church for which I was organist and choir director approached me. She knew of my interest in TEX. She said to me, in essence, "Boy, have I got a job for you... Do you want to work at the AMS?"

I was dumbfounded and at a loss for words — a rare occurrence for me. Here I was being offered the possibility of working in the same environment as two stellar figures in the TEX world: Mike Downes and Barbara Beeton. Actually, as I learned later, I would be working *in the very same department*, the Publications Technical Group. I count it a great loss that I never got to know Mike Downes very well: Tragically, he died before I was able to work up the courage to engage him in conversation — my loss. Getting to know Barbara Beeton has been a treat, even though I know I try her patience from time to time. There is so much to learn!

7 As it was in the beginning...

I started to see mentions of Dick Koch's new TEX editor and previewer called TeXShop, and it sounded

interesting. At the time I was still using Textures. My various projects that I had going at the time did not really allow me to change my TEX environment. But it sounded promising and I kept an eye out for future developments.

They were not long in coming. Word was circulating about Gerben Wierda's i-Installer, and how it took care of all the painful details of installing and maintaining TEX. Better and better. I downloaded both TeXShop and i-Installer and put them to work. Lo! and behold, it worked. This was beginning to look a lot like what Textures was to Mac OS 9. My deepest thanks to Dick and Gerben! But the best was yet to come...

7.1 St. Jonathan and the blessèd XETEX

One day I read something about Jonathan Kew's XETEX, and it looked almost too good to be true: Unicode, and the ability to use both AAT and OTF fonts out of the box. Using this extraordinarily powerful tool has been a transforming experience, and has re-energized my interest in TEX. Thanks, Jonathan!

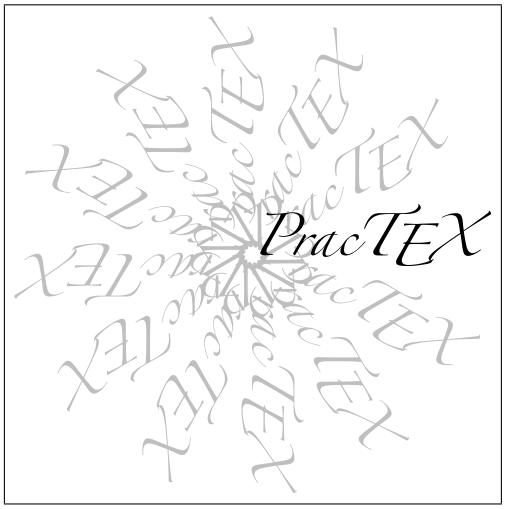


Figure 2: A "fontflake": a typographic frolic done entirely in XETEX, using its native controls.

7.2 St. Will and the miraculous fontspec

And then there came Will Robertson's wonderful fontspec package for XELATEX. Now, at this point I have to make a confession: I am a plain TEX person. From my perspective, LATEX has become the Microsoft Word of the TEX world. In the work that I have done, fonts are a major design issue, and the ability to change the typefaces quickly and efficiently in a given project or series of projects is an important requirement. Yes, I have spent hours with fontinst, afm2tfm, fd files, and all the rest of it, but I do not consider it time well spent. I would also point out that I have a love-hate relationship with Computer Modern: sometimes I hate it, at other times I love to hate it. Will's fontspec removes, for me, one of the major barriers to using LATEX, and for that I'm, well, if not exactly overjoyed, at least pleasurably intrigued. Thanks, Will!

So, at present, I think I can safely say that I am back to where I was when Textures was my TEX tool of choice — indeed, I am much better off, with many more typographic options at my disposal.

8 ... is now ...

Over the years, in addition to the book, I have done a tremendous amount of work for non-profit arts-oriented organizations (programs, pamphlets, order forms, survey forms, and more), and helped a friend to publish genealogical tables nicely formatted — all using plain TEX with Textures and more recently TeXShop. I've also burdened the CTAN archives with a series of type specimens collectively called *typespec*.

Figure 3: A sample of one of the *typespec* specimens created using plain TEX.

Since coming to the AMS, life has been... *interesting,* to say the least. Not only have I had to

develop my Unix skills (as those skills verged on the ethereally exiguous, this was easy), but I've had to come to terms with something called VMS — ugh. At present we do all of our production work on VMS, though this is shortly to change: we hope to move everything to Unix by the end of this year. This will be accompanied by a complete reorganization of the directory structure for processing our materials, and putting the whole thing under Subversion — oh yes, we've had a lot of fun with that — which will serve as both version control and online archive. We'll have considerably more to say about this, I should think, at PracTEX 2007.

We are also moving *The Notices of the American Mathematical Society* away from Quark XPress and to Adobe InDesign. *Notices* is the publication of record for the business of the AMS: membership, meetings, announcements. It was decided in 1995 to alter the nature of *Notices* radically: It would become a glossy, glamorous magazine that would present current mathematical topics in an attractive and colorful way, in addition to performing its original function. The first issue in the new format appeared in January, 1995. In effect, *Notices* was now to become the product of desktop publishing. This was made possible by the fortuitous appearance of two pieces of software: Quark XPress and Mathsetter. Mathsetter, for those who don't know what it is, is a plugin for XPress (version 4.x only) that converts TEX math to XPress type; it uses the Textures TEX engine to do this. Well, Textures is not exactly what you'd call current at the moment, and the same may be said of Mathsetter, only more so. Moving away from XPress and Mathsetter has meant moving to TEX: all of the math-heavy articles in *Notices* are typeset entirely in TEX. This is rather exciting. It is fascinating, and more than a little disturbing, to contemplate how vital a piece of software can become. When we started to look at alternatives to Mathsetter, it was astounding how much material passed through it: blurbs on the back covers of books, catalogue materials, things we post on the web, marketing and promotional materials of myriad kinds — and *Notices* feature articles and other contributions that use a lot of math.

As if all of that were not enough, we have also replaced our aging film imagesetter with a brand new platesetter from basysPrint of Boizenburg, Germany.[5] The interesting thing about this piece of equipment (not counting the fact that it is HUGE: seven feet square by five feet three inches in height, and weighing 2.5 tons!) is that it exposes standard, inexpensive, conventional printing plates using a powerful source of ultraviolet light focused through a solid state chip[6] that contains thousands of tiny little mirrors that focus the light and create an image on the plate. The chip, alone, costs about $9,000 to replace, should that ever be necessary.

[5] About 100 km east of Hamburg.

9 ... And ever shall be?

Given my life thus far with TEX, I trust that you will permit me to venture a few observations. Clearly, we are currently enduring the curse of living in "interesting times". Whereas we used to live in a simple world divided between plain TEX and LATEX, we now face an intimidating array of possibilities: TEX and LATEX; PDFTEX and PDFLATEX; XETEX and XELATEX; ConTEXt; Omega... and on it goes. All this is wonderful for authors, but a veritable nightmare of sorts for publishers who have to streamline their production for maximum effect, minimum waste, and quickest turnaround.

As a side note, what can be said for flavors of TEX goes double for graphics: the dizzying array of graphics packages (in many versions) that turn out a profusion of graphics formats is becoming a real problem to publishers. But that is a subject for another day....

Our production is based on LATEX: for journals we require TEX files, and strongly encourage our authors to use the AMS class files (we typeset virtually all of our journals inhouse). For books we require DVI files, though this has begun to shift significantly: If the author is willing to do *all* of the work to format the book to our standards, we will accept a PDF for the project. But how many times have we heard this at the AMS from an author: "I'm using TeXShop which means I have to give you a PDF: TeXShop can't make a DVI file."[7] Hmmm... Or received PDF files with missing fonts, or PNG or JPEG files accompanying a DVI file?

And whereas at one time it looked as if PDF files might be a miraculous problem solver, now we need to ask which PDF specification: 1.1, 1.2, 1.3, 1.4, 1.5, or 1.6? Or PDF/X-1a or PDF/X-3... and, again, the list goes on. Is the author using the Jaws PDF tools, or Ghostscript's, or Adobe's? And not all PDFs of the same specification are made equal, as we all know.

From time to time we get angry emails from exasperated authors (often in sciences with less overt mathematical tendencies) demanding testily why we

[6] Developed by Texas Instruments; also used in large-screen projection televisions.

[7] Note to Dick: could you please add "and dvi" to the "TeX and Ghostscript" menu item?

are still using TEX, a tired, old-fashioned, and awkward tool at best. Surely there is something else, more modern, more up-to-date that would be more suitable. Why not Microsoft Word? On those occasions, if I knew how to do a hollow laugh, I'd do one.

Well, I think it's a mess, that a catastrophe is looming as increasing costs, diminishing income, and tightening timeframes for publishers conflict with increasing pressure to publish as much as possible in the smallest period of time, using whatever tools happen to be handy at the moment in the academic community, and an increasing interest in bypassing traditional publishing altogether and going right to the Internet. Working in the field of scholarly publishing right now is a bit like watching a train wreck happening in slow motion.

I don't know the solution to these problems. Publishers are in a difficult position because they have workflows that require a limited variety of input for them to produce reliable output in a timely way. It is my obligation as an author to provide what the publisher requires, particularly if I want to be certain that I get what I want. But I'm lucky: I have wonderful tools at my disposal, and I'm lucky enough to know how to use them, usually.

In large part, I think we can make a start by putting well-designed, versatile, and comprehensible tools into the hands of authors, tools such as TeXShop and MacTEX; and to provide documentation that is as good as the software. Not long ago we dealt with an author, a Mac user, who worked on an important book for the last ten years, and used Textures to put it together. At some point he switched to OS X. His problem was that as OS X developed and grew, Textures did not. Finally, the situation became intolerable for him and he asked the AMS for help. We pointed him in the direction of TeXShop and i-Installer and gave him some pointers for getting started. In about two weeks, the work of ten years was easily handed off to TeXShop with no problems. Another transformative experience and a happy author.

In preparing this presentation, I ran across a wonderful interview with Christina Thiele,[8] among a number of other very interesting interviews on the TEX Interviews web page. She makes three points, which I will restate here because they bring what I have been saying nicely into focus.

First, we have to make TEX easy to install and maintain. And for goodness' sake, do let's fix the font mess: Using a variety of fonts should not require an advanced degree in computer science and the patience of Job. You won't get a large number of people to use TEX until it is far more user-friendly, user-comprehensible, and user-supportable. Don't expect anyone necessarily to "RTFM" because, more often than not, they aren't going to, at least not all of it.

Second, I'm sorry to say, we have to abandon the *beautiful documents* argument to try to get people to use TEX — it just doesn't work. Until I had read Christina's interview, I thought I had been an inept salesman of TEX's ability to produce beautiful documents. Fact is, most people just don't care about typographic æsthetics. Users want powerful, effective software that gives the best return on the time and effort they spend to learn and use it: typographic niceties are largely irrelevant.

Finally, we have to do anything and everything within our power to preserve the wonderful, open, empowering, and helpful nature of the TEX community. Everyone and anyone who uses TEX has had occasion to use `comp.text.tex` and to come away from the experience a better TEX user. Help there is free, and often laced with background information that is invaluable to help the learner. The TEX Users Group, I need hardly say, is a magnificent resource that deserves all the support that we can give it.

10 World without end. Amen.

So it has been a long and never uninteresting road from 1986 to the present, with lots of challenges ahead. Yikes! Can it really be twenty-one years? In some ways it seems like yesterday, in others like a lifetime ago: Just look at how the Internet alone has changed our lives and our thinking in a mere fifteen years. To this day, I never cease to marvel at, and be grateful for, the intoxicating sense of empowerment that I have experienced with the tools on my computer. I can only hope that everyone might experience the same joy with the tools that they use.

◇ Stephen Moye
American Mathematical Society
Publications Technical Group
201 Charles Street
Providence, RI 02940
sgm (at) ams dot org

Organist and Choir Director
Bethany Lutheran Church
116 Rolfe Street
Cranston, RI 02910
stephenmoye (at) mac dot com

[8] http://www.tug.org/interviews/interview-files/christina-thiele.html

ConTEXt user meeting 2007: Epen, March 23–25

Mojca Miklavec

The 23rd of March was a drippy Friday afternoon, but the rain did not stop TEXies from using almost all modes of transport to gather in the small village of Epen on the Dutch-Belgian-German border. Most of the 26 attendees from 11 countries arrived by plane, train, bus, or car, but a few preferred to roller blade, to buy a new bike (because the old one broke down on the way to the meeting), or to jump over the fences all the way from Aachen.

The extremely interesting and packed schedule meant evening discussions that often continued into the wee hours of the morning. And even with the delicious food, Taco usually had problems interrupting the lively discussions and wild coding to bring us down to the dining room before the food got cold.

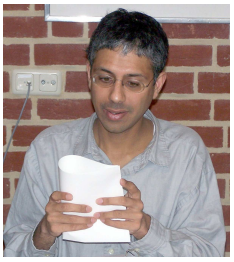
Indeed, we hardly had time to breathe during the meeting. Those who were not in Epen should regret missing out on the fun!

1 Tutorials

Taco Hoekwater
Writing a ConTEXt module

Friday evening started with an excellent tutorial by Taco on how to write a ConTEXt module. He led us from the basics of writing a module to the most important aspects, conventions, tips, and tricks. The tutorial included dozens of pages of documentation to be published on the wiki. Our assignment was to write a new `FIXME` module, which has been on the ConTEXt wishlist for more than a year. Although no module had been written by the end of the meeting (perhaps because there were too many interesting talks), the tutorial and documentation should inspire more authors to provide new high-quality ConTEXt modules.

Two more tutorials followed on Saturday.



Willi Egger
Page layout, arrangements and posters

Willi's tutorial was not just about setting up a page layout in ConTEXt, but also dealt with typographical traditions concerning printing, and related technical aspects like the properties of the paper sheets your book will be printed on.

In the picture you can see Sanjoy Mahajan solving some of the related questions: how many times do I need to fold this sheet to get 16 pages? And what is the grain direction of the paper?

Hans Hagen
XML

In contrast to math where many users can start from their LATEX experience when switching to ConTEXt, XML processing is ConTEXt's specialty, and lacks a beginner's manual. That meant that every single hand was raised when Hans asked about the interest to listen to his XML tutorial, although it was almost bed-time when it started.

Editor's Note: First published in *MAPS* 35, 2007. Reprinted with permission.

Willi in action

2 History of typesetting & typesetting of history

Taco Hoekwater
A short history of ConTEXt

The first talk on Saturday morning was given by Taco Hoekwater, one of the first ConTEXt users outside Pragma ADE. It was enjoyable to listen to the summary of the early revolutionary ideas Hans implemented into his system at a time when some of the attendees in Epen were still school kids who had only dreamt about owning a computer.

Thomas A. Schmitz
Classical Greek with ConTEXt

Coming from a completely different field than most attendees (humanities), Thomas shared his experience of ConTEXt related to typesetting Ancient Greek, struggling with font-system oddities and limitations of pdfTEX. Is LuaTEX going to offer the definitive answer to the problems he had to fight with?

Idris Samawi Hamid
Critical editions

In contrast to the usual development cycles of ConTEXt and its modules, where documentation is lagging way behind the functionality, Idris brought with him a complete specification of a yet-to-be-written module for typesetting critical editions with multiple levels of footnotes, proposing a complete hierarchy of commentaries.

3 Talks from user experience ...

Mari Voipio
ConTEXting in MS Windows — a user's view

Mari presented all the problems a newbie faces when switching from Microsoft products to ConTEXt: no WYSIWYG editor, no drop-down menu with fonts, no copy-paste to include images, no easy way to create tables ... Would a Word2ConTEXt tutorial help?

Sanjoy Mahajan
Typesetting a physics textbook with ConTEXt

Sanjoy described how he uses ConTEXt to typeset his mathematics textbook (*Street-fighting mathematics*), covering project structure, page layout, and figure–text integration (figure placement).

Duncan Hothersall
Using ConTEXt as part of a larger system

Duncan's company uses ConTEXt for typesetting multilingual documents (including Arabic) from XML sources, but he showed a great deal of courage when he dared to make a presentation in PowerPoint during ConTEXt meeting, for which he has been "punished" appropriately during the live demo.

The conference room

4 Useful tools

Patrick Gundlach
ConTEXt integration in the TextMate editor

Patrick's presentation of his TextMate extensions for better ConTEXt support could be called "a story of success" or "a good reason why one should attend TEX meetings". His first announcement about the ConTEXt bundle on the mailing list got no reply. This time, most of the Mac users at the meeting eagerly awaited the upload of the new bundle, in order to try it on their own computers. The bundle supports syntax highlighting and auto-completion of all user-level commands. It also provides an interactive list of arguments that these commands accept, and provides shortcuts for typesetting and previewing.

Patrick Gundlach
Creating a PDF document, the hard way

Probably tired of the ease of typesetting PDF documents with ConTEXt, he has also written support for manual editing of PDF in TextMate, so that cross-reference tables can be calculated automatically. Interested hackers should ask him for details.

5 Discussions

Mojca Miklavec
Documentation

All attendees agreed that our shelves lacked a well-written "The ConTEXtbook" and/or (online) Cookbook, but no one volunteered to write it. It was also agreed that TEXshow is incomplete. Filling the gaps in this area is mostly up to the users.

Some suggested to clean up and improve the wiki pages: to create a site index, to point to the most important recent changes and advances since the last update of ConTEXt manual. More samples should be provided, and submitting test files should be made easier.

Sanjoy Mahajan
Regression testing

The one who makes few mistakes makes little progress. Because ConTEXt development is progressing apace, a repository with test suites has been set up, and Sanjoy has been developing tools to check for broken functionality between different ConTEXt releases. Simplifying submissions of test cases should be one of the first steps towards a better quality control before new "γ releases", as some jokingly called them.

6 A glimpse into the future

Arthur Reutenauer
An introduction to OpenType fonts

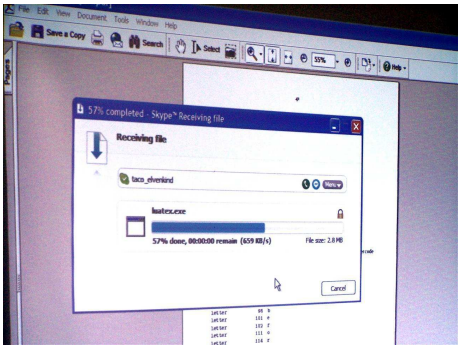
Unfortunately, OpenType fonts are not as open as one would expect them to be, but luckily enough both Hans and Patrick have been open enough to ideas and requirements for proper font support in ConTEXt and on the garden, some of which had secretly been added to `live.contextgarden.net` during the night before presentation, while Patrick himself was sleeping.

Idris Samawi Hamid
Oriental TEX

If typesetting Ancient Greek is problematic — what about Arabic? After (mis)using Aleph for quite some time, Idris claimed that no existing system was able to meet his requirements to typeset old Arabic texts, and applied for a grant at Colorado State University to fund Taco's development of LuaTEX. Without such a grant, we would probably not be seeing the rapid development of LuaTEX, aiming for the first beta release during the TUG conference this year.

We are now eagerly awaiting his next tutorial about writing funding proposals (hopefully applicable to Euro currency as well).

Hans Hagen, Taco Hoekwater
LuaTEX, ConTEXt MkIV, fonts, & typescripts



Last but not least, Hans and Taco filled in the remaining time not already reserved for other presentations. Although I've seen quite a few of their talks already, they always keep us surprising with some really special slides — both in visual appearance and contents. The development has been progressing almost at light-speed — even during the conference.

The picture shows a (non-scheduled) live demonstration of bugfixing in LuaTEX during Hans's tutorial.

The most special thing of the meeting was that most of us have known each other from vivid discussions on the mailing list without ever meeting before. So the introductory discussions that would otherwise start with platitude phrases asking for names, location or profession, were usually replaced by hitting the core of the subject:

"Wait! You are the one who did . . . !"

The meeting was a great success and left a deep impression on everyone. Most attendees left Epen saying "See you next year".

How to fit five nationalities into one car? Well, TEXies have lots of experience with packing boxes. So as long as they go well with each other, they always find a solution.

7 Invitation to Slovenia

The second ConTEXt user meeting, in 2008, will take place in Slovenia, near the Slovenian-Austrian-Italian border, and last probably a day or two longer than this one, in order to leave you more time to discuss the fascinating topics, to exchange ideas, and to breathe fresh air while sightseeing or doing sport. As the date approaches and planning crystallizes, the time and location will be announced on the mailing list and `contextgarden.net`.

Participants in the ConTEXt 2007 user meeting:

Front row (from left to right): Luigi Scarso (it), Mojca Miklavec (si), Zofia Walczak (pl), Taco Hoekwater (nl), Duncan Hothersall (uk), Arthur Reutenauer (fr).

Back row: Bernd Militzer (de), Willi Egger (nl), Thomas Engel (de), David Roderick (uk), Oliver Buerschaper (de), Patrick Gundlach (de), Mari Voipio (fi), Steffen Wolfrum (de), Luuk Beurskens (nl), Alexander S. Berdnikov (ru), Jano Kula (cz), Idris Samawi Hamid (us), Hans Hagen (nl), David Kastrup (de), Michael Guravage (nl), Karel Horák (cz).

Missing from the photo: Sanjoy Mahajan (us), Tobias Burnus (de), Thomas A. Schmitz (de), Jelle Huisman (nl).

◇ Mojca Miklavec
Slovenia

EuroBachoTEX 2007

Michael Guravage

Saturday

Jerzy Ludwichowski, GUST president and conference organizing committee chair, opened the conference by welcoming all the participants, and encouraged everyone to enjoy the proceedings in the spirit of the conference — "Paths to the Future".

Jerzy Ludwichowski

The first speaker was **Jonathan Kew**, who related the history and current status of XeTEX. After its initial appearance in the spring of 2004 on Mac OS X, a version supporting OpenType fonts appeared the following year. XeTEX for Linux was announced at BachoTEX 2006, and was quickly followed in June by a version for Windows. This year marked a milestone for the XeTEX project, in that it became an integral part of TEX Live distribution.

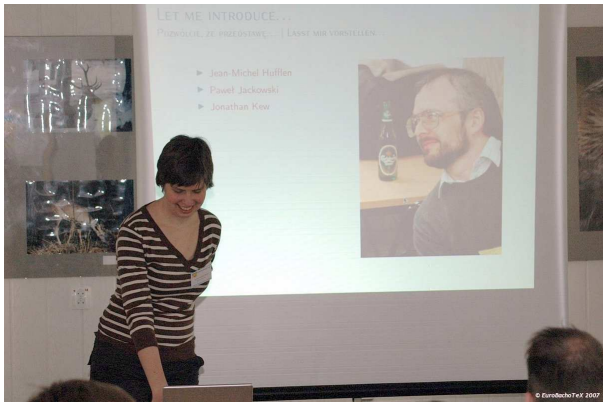
Key features of the current XeTEX implementation include its native Unicode support, improved integration with existing macro packages and its smart inclusion of hyphenation patterns.

Looking ahead, future releases will support host operating system fonts (OpenType, TrueType and PostScript) with no TEX-specific setup. Another new feature is inter-character tokens — inserting arbitrary tokens in-between adjacent characters based on character classes. This allows one to easily mix scripts and fonts, or insert spacing to stretch text. Finally, to better support non-Latin scripts, minority languages, and scripts not yet in Unicode, XeTEX will support SIL's Graphite font system.

Taco Hoekwater began his presentation by announcing that MetaPost version 1.0 is now available. New features include file name templates, new color types, i.e. cmyk, grey-scale and marking-only and an improved manual. To overcome the various size limitations of the current implementation, MetaPost version 1.1 will incorporate dynamic array allocation and provide greater numeric precision. To obviate problems with existing input files, this new version will appear as a separate binary. Finally, in the next year or so Taco anticipates making MetaPost functionality available as a library.

The title of **Hans Hagen's** presentation was "Beware of too much tokenspeak". TEX consumes characters which, in turn, become tokens, and then nodes. Hans gave us a glimpse into how LuaTEX, at the node list level, is simplifying and streamlining previously complex pieces of TEX. So much so that he has been able to retire moderate pieces of existing ConTEXt code. Consistent with the theme of the conference, Hans described how LuaTEX provides a genuine opportunity to embrace the future.

Joanna Ludmiła Ryćko introduced the TEX Clinic. The clinic began last year at BachoTEX, and was open to anyone at the conference seeking relief from a nagging TEX complaint. A number of TEX clinicians were introduced and put at the disposal of the participants.



Joanna Ryćko

Johannes Große presented MathPSfrag — a tool that replaces existing labels in Encapsulated PostScript graphics with LaTEX generated labels. MathPSfrag extends PSfrag, allowing both automatic and fine grained manual control over label content and placement.

Siep Kroonenberg presented her epspdf conversion utility. Written in Ruby and Ruby/Tk, and using Ghostscript and pdftops, epspdf offers both command line and graphical user interfaces for a round-trip conversion between PostScript and PDF.

EuroBachoTEX 2007 participants. *Not standing, from left:* Siep Kronenberg, David Kastrup, Stanisław Walczak, Jonathan Kew, Grażyna Jackowska, Jerzy Ludwichowski, Ewa Koisar, Joanna Ryćko, Marek Ryćko, Jakub Zdroik, Michał Kolany, Hossam A.H. Fahmy, Mojca Miklavec, Frank Küster, Arthur Reutenauer.
Standing, from left: Włodzimierz Martin, Ewelina Łuczak, Harald König, Małgorzata Łuczak, Stanisław Wawrykiewicz, Wojciech Łukaszewicz, Tomasz Łuczak, Jarosław Brykalski, Ross Moore, Taco Hoekwater, John Trapp, Reinhard Kotucha, Hàn Thế Thành Péter Szábo, Sam Guravage, Michael Guravage, Karel Píška, Hans Hagen, Bram Otten, Bogusław Jackowski, Volker RW Schaa, Agata Wylot, Aleksandra Kaptur, Heiko Oberdiek, Leszek Czerwiński, Marcin Woliński, Jan Ryćko, Marta Wolińska, Grzegorz Murzynowski, Adam Kolany, Radosław Tryc, Dorota Kolany, Norbert Preining, Ferenc Wettl, Andrzej Borzyszkowski, Dorota Cendrowska, Zofia Walczak, Ulrik Vieth, Matrin Schröder, Dag Langmyr, Ewa Szelatyńska, Jano Kula, Karel Horák, Petr Sojka, Deimantas Galčius, Klaus Höppner, Natalia Chlebus, Vytas Statulevičius, Atif Gulzar, Alexander Berdnikov.
Photo by Jacek Kmiecik.

Zofia Walczak demonstrated several basic and advanced features of the Portable Graphics Format (PGF) package. Written by Till Tantau at the Institute for Theoretical Computer Science at the University of Lübeck, PGF is partitioned in three layers: system, basic and front-end. Ti*k*Z is a front-end for PGF. It provides access to all the features of PGF, and is intended to be easy to use. If you look closely you will see it has borrowed part of its syntax from both Metafont and PSTricks.

Norbert Preining stood in for **Jim Hefferon** and described a new 'experimental' procedure for uploading software to CTAN. The workflow includes upload, approve and install steps resulting in TDS compliant bundles. A means for updating package meta-data is also present.

Jean-Michel Hufflen introduced us to XSL-FO, comparing and contrasting corresponding LaTeX and XSL-FO structures.

Grzegorz Murzynowski introduced gmdoc, a package for documenting LaTeX style files. It differs from its predecessor by emphasising compact 'minimal' markup.

Grzegorz Murzynowski continued by describing his gmverse and gmcontinuo packages. The former provides right alignment for long and broken lines of verse. The latter allows typesetting paragraphs *in continuo*, marked not with a new line and indent but continuously, marked with only the ¶ sign.

In the last talk for Saturday, **Marek Ryćko** argued for a fine, or finer, grained component architecture for TEX functionality. He hopes that focusing on interfaces to facilitate integration will be the tipping point for TEX development.

The weather was clear and cool throughout the week. So it was under the stars and a waxing moon that, later that evening, we enjoyed the annual bon-

Handmade paper projects

Norbert Preining

fire; replete with food, drinks, songs, and of course fire-breathing pyroTEXnics.

Sunday

This year we found the accommodations not quite ready for guests. For instance, there were no curtains and toilet paper was missing as well. It took us a while to find out that all cloth and paper was being used in the "make yourself some paper" workshop given by **Grażyna Jackowska** that ran in parallel to the talks. As the conferences advanced, the participants had to become more careful where they walked because handmade paper was hanging on trees everywhere.

Andrzej Tomaszewski began the second full day of talks by describing the various conditions and limitations he encountered while producing "The Master of Life Arteries of the Greater Warsaw;" a jubilee book for the Warsaw Municipal Water Authority.

Dorota Cendrowska presented several, oft disregarded, design criteria to consider when typesetting enumerations for inclusion in printed text and multimedia presentations.

Jerzy Ludwichowski described his and **Karl Berry**'s work on consolidating the GUST SOURCE and NONSOURCE font licences into the single GUST Font License (GFL). The result is a license that is legally identical to the LaTeX Project Public License (LPPL), which the FSF and Debian already accept as a legitimate free software license.

Jean-Michel Hufflen described how MlBibTEX strives to be a better BibTEX. Starting in 2000, MlBibTEX originally was written in C, but has been reimplemented recently in Scheme, a Lisp dialect. Jean-Michel anticipates MlBibTEX's first public release in May of this year.

Next, **Jean-Michel Hufflen** showed how lexicographical order relations are language-dependent, and how MlBibTEX addresses this issue in the context of multilingual bibliographies. Bibliography styles can be unsorted or sorted. However, the `bst` language's sort function is suitable for English only. MlBibTEX uses `nbst` and Scheme, which together allows one to sort European languages in correct lexicographic order.

David Kastrup described how to download, install and use the Emacs AUCTEX package. You can retrieve the latest version of AUCTEX from

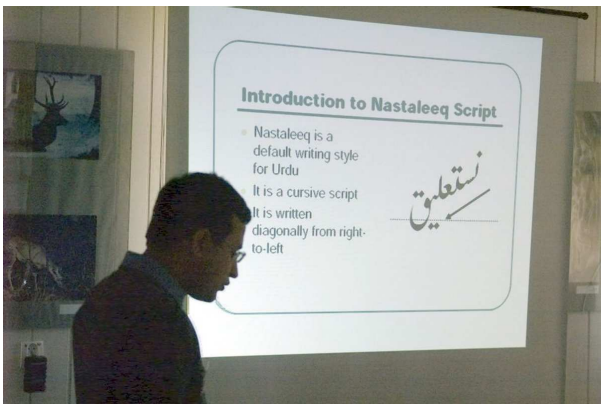
`http://www.gnu.org/software/auctex`.

And for the stout of heart, the source code for a pre-release version of Emacs 22 is available from `http://alpha.gnu.org/gnu/emacs/pretest`.

Péter Szabó demonstrated dvdmenuauthor, a collection of tools, including pdfLaTeX and xpdf, used to create menus for dvdauthor — an excellent low level tool for creating video DVDs on Unix systems.

Norbert Preining described how the Debian "etch" release contains both TEX Live 2005 and teTEX — in parallel — and how both system administrators and regular users can benefit from side-by-side TEX distributions. Norbert concluded with a preview of TEX Live 2007 and further developments regarding TEX on Debian.

The presentation of **Atif Gulzar** and **Shafiq-ur Rahman**, who are from Pakistan, began by explaining how Urdu is used by some sixty million people in twenty countries. Urdu is based on an Arabic script with Nastaleeq as its most prevalent writing style. Nastaleeq is highly contextual — written right-to-left and top-to-bottom. Atif constructed an Omega virtual font containing 827 glyphs, and used Omega external OTPs in a two-pass solution



Atif Gulzar

to achieve the appropriate ligature placement and kerning. From the more than twenty thousand valid ligatures in Urdu, Atif was able to correctly render and place a subset of approximately seven thousand ligatures.

Hossam A.H. Fahmy presented his joint paper with **Amir M.S. Hamdy** about their aim to create a font suitable for typesetting the Qur'an. Using examples from existing fonts, he explained many of the problems that one encounters when attempting to digitize a calligraphic script like Arabic. The second part of the talk focused on a detail of that: how to simulate a real-world pen nib in Metafont.

All the news about pdfTEX version 1.40 was brought to us by **Martin Schröder**. Most prominent among the new features are the ability to create compressed object streams, support for JBIG2–encoded images, and the addition of a colorstack à la dvips. The colorstack feature is already in use in the new releases of the hyperref package, and solves the LATEX problem of the text color disappearing at a page break.

Karel Horák walked us through the history of the háček — or caron, if you prefer — in Czech typesetting. He showed us not only an objective historical progression of the symbol shape, but also many forms that occur in actual fonts. Some few he considered good, some more not so good, almost all are apparently simply hideous and out of touch with Czech tradition. The likely cause is that the big font foundries never considered asking a Czech typographer for an opinion.

Hàn Thế Thành also talked mostly about accents, but in this case about the ones used in Vietnamese. The writing system is based on the Latin alphabet, but it has a great many accented characters to denote sounds that are not differentiated in the roman alphabet. His VnTEX package is a complete solution for typesetting Vietnamese, including support for large number of fonts, some of which he created himself.

Hàn Thế Thành

The day ended with two presentations by **Tomasz Łuczak**. The first talk was about the LyX document processor (`www.lyx.org`), the second about how to convert wiki markup into TEX source. Unfortunately, both talks were given in Polish, and even with the simultaneous English translations provided by kind audience members they were hard to follow.

Monday

There were no lectures scheduled for Monday. Instead, we took an excursion to Toruń where we visited the District Public Library–Copernican Library and toured the town. After which we drove on to Chełmno where we enjoyed a scrumptious dinner and music before returning home.

Toruń, situated astride the Vistula (Wisla) river, has been an important regional and trading center since medieval time. A member of the Hanseatic League, Toruń boasted a fleet of one hundred and fifty ships, whose trade allowed Toruń's prosperity to rival that of Brugge, Copenhagen and London. UNESCO designated the Gothic buildings of Toruń's Old Town a World Heritage Site in 1997.

At the Copernican Library we were treated to a sample of the treasures of their collection, including a first edition of Copernicus's "Revolutionibus Orbium Coelestium", or "The Revolution of the Heavenly Orbs", which appeared in print in 1543. Lastly, we were shown a recent reproduction of Gutenberg's Bible. The exemplar is one of 180 copies, matching Gutenberg's original number. Each exemplar was made using the same materials and techniques as the originals, including individual letter variations (font

Gutenberg Bible exemplar at the Copernican Library

Our excellent entertainers

expansion) that Gutenberg used to achieve aesthetic interline spacing.

Our tour of Toruń's Old Town began at the historic Town Hall under a statue of Nicholas Copernicus with the inscription, "He moved the earth, and made the sun stand still". We visited several churches and historical landmarks before ending where we started.

We were running late, so it was late in the afternoon when we arrived in Chełmno, a town located on seven hills, and one of Europe's best examples of defensive architecture. Chełmno's several churches date from the thirteenth and fourteenth centuries. On the fourteenth of February each year, the inhabitants ostentatiously celebrate Saint Valentine's Day since the local parish church has kept the saint's reliquary for many centuries.

After a short stroll through the town, we retired to a local restaurant where we enjoyed a delicious buffet dinner. Entertainment was provided by a group of musicians including Bogusław Jackowski's daughter.

Tuesday

In the first presentation Tuesday, **Hàn Thế Thành** presented a summary of font-related topics in pdfTEX. Some, like font expansion and margin kerning, are already documented in the pdfTEX manual. The rest are scattered across `README` and example files, e-mails and mailing lists. For the first time, all these topics were brought together in one place. Topics include adjusting letter and interword spacing, adding additional kerning before or after certain characters from a font, Unicode support for browser cut, paste and search actions and subfonts — a mechanism for supporting CJK languages.

Hans Hagen began his presentation by describing the issues driving the development of ConTEXt's font system, namely switching between different font styles and sizes, and proper font handling in math mode. To make font switching easier, ConTEXt can assemble a collection of different fonts into a single structure called a typescript. For example, a typescript might use palatino-regular as the default serif font, palatino-sans as the sans font, courier as the monospace font and euler as the math font. Instantiating this typescript would make these fonts available when using the commands `\rm`, `\ss`, `\tt`, and `$...$` respectively.

Hans Hagen

Hans concluded by describing how the trend toward OpenType fonts, consistent user interfaces and DTP-like functionality will continue to inform where and how ConTEXt controls fonts — and vice versa.

Taco Hoekwater explained how LuaTEX, with its native support for OpenType fonts, will obviate the need for static font metric files. Currently LuaTEX implements a few dozen callbacks at strategic points in TEX. When populated, callbacks will override

TEX's default behaviour with custom code. Taco demonstrated how, when using OpenType fonts, LuaTEX callbacks invoke code that extract the font metric information directly from the OpenType font itself.

Grzegorz Murzynowski identified two differing opinions concerning the TEX & Co. logos. The first group contends that the font is part of a logo, and therefore the combination is inviolate. The second group contends that a logo should be typeset in the same font as its context. For the latter group Grzegorz suggests several slight modifications to the LATEX logo to make it fit better with various fonts.

Sam Guravage, the youngest speaker ever to address a BachoTEX conference, explained how he uses TEX for all his school assignments. Sam enumerated what he found easy in TEX e.g. sectioning and lists, and what he found difficult e.g. figures and error messages. Sam's conclusion was that TEX makes his work look better, and looking better meant higher grades.

Sam Guravage

David Kastrup began a series of talks by introducing `qstest` — a LATEX macro package for writing regression tests. The idea is that a user can include a number of tests in his `.dtx` files and use pattern and keyword lists to specify which tests should be run, either when his package is loaded or while running a separate test file through LATEX. The `qstest` package, together with the `dtx` documentation format and `docstrip`, allows one to integrate unit testing and documentation in a single `.dtx` file.

David Kastrup continued with a discussion of the `makematch` LATEX macro package. Factored out of the `gstest` package, `makematch` matches patterns with wildcards against a list of targets.

David Kastrup

David Kastrup concluded his series of talks by explaining how the `bigfoot` macro package, originally written as a footnote apparatus for text-critical editions, can benefit the ordinary LATEX user. For example, default footnote behavior bypasses TEX's global pagebreak optimization whenever a footnote does not completely fit on one page. In contrast, footnote breaks in `bigfoot` are reconsidered for each possible breakpoint of the main text. This means TEX will find the optimum combination of breaks in main and footnote texts.

Robustness, optimization, color continuity and paragraph footnotes are just a few reasons why LATEX users might consider using `bigfoot` to replace TEX's native footnote apparatus.

Klaus Höppner walked us through the process of creating PostScript Type 1 fonts from MetaPost sources using MetaType1. Created by Bogusław Jackowski, Janusz Nowacki and Piotr Strzelczyk, MetaType1 is a collection of tools including MetaPost, `t1utils` and AWK; together they are used to generate PostScript Type 1 AFM, TFM and PFB files. Though documentation was scarce, MetaType1 proved to be the correct tool for the job.

Petr Sojka and **Michal Růžička** explained how they generated PDF, HTML and XHTML+MathML output from a single LATEX source file. While many single-source publishing approaches begin with XML, the amount of mathematics involved made TEX the only viable input format. By enforcing a strict separation of form and content, and modifying the TEX4ht sources, the authors were able to realize individual workflows for each output format.

Péter Szabó reflected on his experience compiling various conference proceedings — including those of last year's EuroTEX conference. Péter described

how the judicious use of procedures and tools can clarify and simplify the work of authors, editors and printers. Revision control software, mailing lists, shell scripts, utilities, instant messaging and of course TEX, can be combined to realize effective publication workflows.

David Kastrup described DocScape Publisher, an XML-oriented database publishing system from QuinScape GmbH. At its core, DocScape uses LATEX, pdfTEX, and David Carlisle's xmltex. Current applications include financial reports, a variety of product catalogs, and online excerpts.

Karel Píška described procedures and programs he has developed for comparing and viewing font elements. His workbench can be downloaded from

`http://www-ep.fzu.cz/ piska/tfcpr.html`.

From this set, Karel demonstrated several tools:

cprpk, cprpkt1, cprpkt1c, cprticpk, cprpkpk: tools for comparing two bitmapped representations of a glyph pair at two different resolutions.

prfkrn, prfkrna, cpkrn, cpkrna: tools for comparing kerning pairs in two (or three) relative TEX fonts, or in two releases of one font.

prfof, cprof: tools for comparing and proofing outline fonts.

Karel Píška

In his second presentation, **Karel Píška** applied his tools to analyze and verify the Latin Modern fonts. His results included examples of individual letter defects and inconsistencies. Interestingly, he found an inordinately large number of kerning pairs; the majority of which he thinks are not relevant to any language. Through his exacting work, Karel is improving the quality of the fonts we use every day.

Janusz M. Nowacki unveiled his complete set of Latin glyphs for the Cyklop font. Designed and cast in lead in Warsaw in the 1920s by J. Idźkowski, Cyklop is a very heavy sans-serif two-element font, Originally produced only in the oblique form, in sizes from 8 to 48 pt, Cyklop is used for newspaper titles, posters, forms, labels and invitations. In addition to the new Latin glyphs, Janusz has added a complete new upright variant.

To round out the day, an informal reception was held in the lecture hall, where participants could enjoy a glass of wine, pleasant conversation, and an exhibition of black and white prints taken by Janusz.

Wednesday

Paweł Jackowski presented this year's crop of TEX beauties and oddities, sixteen in total. You have to see these pearls to believe them. The entire collection can be found at:

`http://www.gust.org.pl/pearls`.

Paweł Jackowski

Ross Moore spoke about his experience typesetting articles for *The Journal of The Australian Mathematical Society*. Leveraging the interactive capabilities of PDF, AMS journal articles, available free online, now incorporate useful meta data that readers would otherwise have to research themselves.

To enlighten our path to the future, **Arthur Reutenauer** recounted TEX's recent history. Subtitled "Pax TEXnica — The program on which the sun never sets", Arthur described how, from TEX78 to Aleph, XETEX and LuaTEX, the various TEX engine extensions and macro packages have gradually enabled us to typeset every language and script of the world — well, almost.

Ulrik Vieth presented an overview of the TEX historic archive, an archive of historic TEX distributions and packages hosted on the TUG FTP server (`http://ftp.tug.org/historic/`). TEX's history spans thirty years now, and while its early history is well documented, the history of various macro packages, fonts, and systems like Metafont and MetaPost must often be pieced together from anecdotal evidence.

After thirty years, the history of TEX remains an interesting topic of research. The archive contains a wealth of information, but gaps still exist. Contributions are welcome, especially those about (pdf)TEX and Latin Modern fonts.

Bogusław Jackowski, Jerzy Ludwichowski and **Janusz M. Nowacki** described the current status of the two large font projects being developed by the TEX user groups: Latin Modern and TEX Gyre.

The Latin Modern fonts project was begun in 2002. Based on Computer Modern, the Latin Modern family currently consists of seventy-two text and twenty math fonts, available in both OpenType and PostScript Type 1 formats.

The Gyre font project that was begun in 2006 aims to supplement the thirty-three URW++ fonts distributed with Ghostscript to cover all Latin languages, similar to the LM fonts. Hinting is improved and files in OpenType format are provided. Extensions to the math capabilities are planned for the near future.

Here are the TEX Gyre fonts which have already been given new names:

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

The Latin Modern and Gyre project pages are on the `http://www.gust.org.pl` website, in the folders `/projects/e-foundry/latin-modern` and `/projects/e-foundry/tex-gyre`.

Recalling Niklaus Wirth's statement that "algorithms plus data structures equal programs", **Marek Ryćko** demonstrated how to realise Lisp-like structures and methods in TEX. Marek argued that a clean and consistent approach to handling lists of elements will make programming TEX simpler, and TEX programs, i.e. macros, more reliable.

Jerzy Ludwichowski concluded the conference proceedings by thanking the organizers, authors and participants. And as a particular encouragement, the GUST board awarded Sam the award for the best conference presentation. The award was impressed on one of the handmade paper sheets.

◇ Michael Guravage
(with Hans Hagen & Taco Hoekwater)

New TEX activities in Korea

Kihwang Lee, Korean TEX Society

The Korean TEX Society (KTS) was founded in January 2007 to promote TEX-related academic activities in Korea, focusing on research and development on the Korean TEX environment, Korean typography, and user support. The establishment of KTS marks the creation of an official and stable organizational foundation which can take over and expand the efforts of the Korean TEX Users Group (KTUG).

KTUG was formed and maintained by a few dedicated TEX users including Kangsoo Kim (director), Jin-Hwan Cho, Koaunghi Un, and Dohyun Kim since 2002. KTUG has played a vital role in spreading TEX in Korea providing extensive user support. KTUG put its effort into implementing and improving the Korean language support in TEX and Korean typography specialized for Hangul, the unique writing system of the Korean language. This effort has been realized as Hangul-ucs, a LATEX package which can typeset unlimited Korean texts encoded in Unicode including archaic Korean texts. Despite these fruitful achievements, KTUG had inherent limits as an unofficial and casual users group for effectively continuing the development activities. This, eventually, gave the motivation for establishing KTS. KTUG will keep its existence in the form of an unofficial and informal user community.

As a big initial move, KTS published the first issue of *The Asian Journal of TEX* (*AJT*) in April. It is the first TEX-related journal in Asia. The first issue contains seven inspiring articles written in Korean. Jin-Hwan Cho, the author of DVIPDFMx and DVIasm, is taking the role of the editor of *AJT*. The editorial board has Hong Feng (chairman of Chinese TEX Users Group), Kangsoo Kim (director of KTUG), Werner Lemberg (author of CJK package), Haruhiko Okumura (maintainer of the biggest TEX Q&A forum in Japan), C.V. Radhakrishnan (founder of Indian TEX Users Group), and Hàn Thế Thành (author of pdfTEX) as its members. The second issue of *AJT* featuring articles and notes written in English is expected to appear in October.

The society is also preparing to host the Asian TEX Conference under the general theme of "TEX in the Age of Digital Humanities" in January, 2008. This event is sponsored by the Kongju National University. KTS is inviting the members of the editorial board of *AJT* as plenary speakers of the conference. Details of the conference will be posted soon.

For more information on KTS and *AJT*, please visit the society web site at `http://kts.ktug.kr` and the journal web site at `http://ajt.ktug.kr`.

Typography

Typographers' Inn

Peter Flynn

1 Web *vs* Paper

Three years into a long-term project to move an entire organisation's documentation into a consistent format, to move their huge web site into a content management system, and to provide PDF print-on-demand, a row has broken out between the web designers and the people who until now have managed the print versions of the documents. Thirty years of uncontrolled free-for-all has left them with documents in all kinds of formats, both in terms of the physical file type and the design layout.

At the core of the dispute is the question, should the print (PDF) version of a document look the same as the web (HTML) version? The web designers, who have produced a nice-looking site, unsurprisingly say yes, print it from the browser with a print CSS stylesheet, and maintain the layout and the look-and-feel they have given the site. The publications people say no, there are things a print document needs that a web document does not, and vice versa, like referenceable page numbers, a specific typeface, and layout spacing designed for a particular paper size.

Their current house print style includes a number of features difficult to achieve consistently in a browser, such as drop caps, 50% indentation, hanging punctuation, and context-sensitive running headers and footers — the kind of stuff routinely familiar to LATEX users — but the interesting parts of the debate have centered around the minutiæ. The publications staff, having for years been used to working to tolerances of less than 1pt, are aghast at the rough-and-ready look of browser-printed documents; the web developers, conscious of the need to satisfy customers with hugely disparate technologies, place a high value on the self-adjusting nature of browser formatting.

What has been refreshing is to see the debate spread outside the web developers and the publications office. You see comments in various online design and typographic forums from time to time to the effect that 'no-one is bothered about it these days', often used as a justification for sloppy design or sloppy typesetting. But people *do* take an interest in the details of typography when they have something to make comparisons with.

It is often said that the objective of typographic design is to be invisible; that is, you should arrange things so that the author's message is conveyed as effectively as possible, without the reader necessarily being aware that any design has actually gone on. This is the way most people read. Only people like us actually spend time checking out the typefaces and the design. In the Real World Outside™, layout only gets noticed when it gets in the way, and typefaces get noticed hardly at all.

An average wordprocessor user probably knows that there are several odd-looking letterforms in her font menu, and may well have used some of them for occasional variety in ephemera like birthday invitations and personal correspondence, but letters and reports get typed in Times New Roman because it is 'what everyone else uses'. In the LaTeX classes I teach, I show some enlarged samples of types while explaining the difference between the web and a piece of paper, and almost everyone is surprised at how different they are, and they are shocked that there are — what is it? — 30,000 typefaces in existence.

With what we have to choose from, wouldn't it be nice if documents formatted for paper *did* look different from those printed from the browser display?

2 Oddities of punctuation

TeX users will be aware of the vast range of signs and symbols available (see Scott Pakin's *Comprehensive LaTeX Symbol List* on CTAN), especially in math mode. A user on `comp.text.tex` asked about several of the rarely-used punctuation marks like the asterism (⁂), the irony mark (؟), the doubt mark, and the certainty mark. I had vaguely heard of the first two, so I did a little digging.

The asterism actually exists as a Unicode character, and although it is not implemented in the UTF-8 packages, it is easily constructed in LaTeX:

```
\newcommand{\asterism}{\smash{%
  \raisebox{-.5ex}{%
  \setlength{\tabcolsep}{-.5pt}%
  \begin{tabular}{@{}cc@{}}%
    \multicolumn2c*\\[-2ex]*&*%
    \end{tabular}}}}
```

The value of `\tabcolsep` needs testing for your surrounding typeface and size, and re-expressing in relative units.

The irony mark is even easier with the graphicx package: `\reflectbox?`, but the other two seem to be harder to track down. Wikipedia and Stumbleupon mention them, but without examples. Does anyone know where to find them?

3 Helvetica

Films about typefaces are rare to the point of nonexistence, so the appearance of *Helvetica*, a documentary to mark the typeface's 50th anniversary this year, was a red-letter day in the calendar.

Although it is possibly one of the most heavily-used typefaces in existence (along with Times), it is a tribute to its designers that it has remained so popular and effective for so long. It was one of my first sheets of Letraset, and a recent paean of praise I read linked from Slashdot (and which I failed to bookmark and now cannot find!) went on at length about how suitable it had been found for every possible application, from corporate web pages to labelling the city corporation's waste facilities. Personally I prefer Univers, but there is no question about Helvetica's popularity.

H E L *v e t* ***I C A***
V E T *i c a* ***H E L***
I C A *h e l* ***V E T***

The web site at `http://www.helveticafilm.com/` has details and links, including the dates of screenings worldwide (many already sold out). I write this before it reaches Ireland, and by unfortunate mischance it screens in San Diego a week before the TUG 2007 conference there this year. Those who have already seen it have been very enthusiastic.

4 2008 TUG meeting in Cork

I am of course delighted that TUG has chosen Cork as the site for the 2008 TUG conference. It will be 18 years since it was last held there, and a lot has changed. There is a web site at `http://tug.org/tug2008/`, and I would like to see plenty of papers on typography and typographic design — so get writing!

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

Book Reviews

Alphabet Stories by Hermann Zapf

Hans Hagen & Taco Hoekwater

Hermann Zapf

Introduction

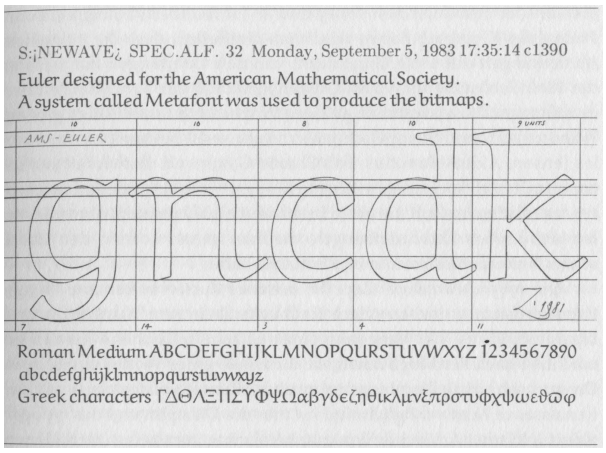
It pays off to be a Dante member! Some time ago each member received a copy of Hermann Zapf's monograph 'Alphabetgeschichten', a gift from Hermann himself. For many users of computers the name 'Zapf' may ring a bell because of the omnipresent Zapf dingbats fonts. But with Hermann Zapf being one of the greatest designers of our time, there is much more to learn about him.

Being an honorary member of Dante, Hermann is quite familiar with TEX and friends, and he is in contact with several TEXies. He worked with Donald Knuth on the book '3:16', a calligraphic masterpiece. He is also responsible for the design of the Euler font family (we will tell you more about this in an upcoming issue). In the recent font projects (Latin Modern and TEX Gyre) we consult Hermann on matters that we are unsure about.

Two versions

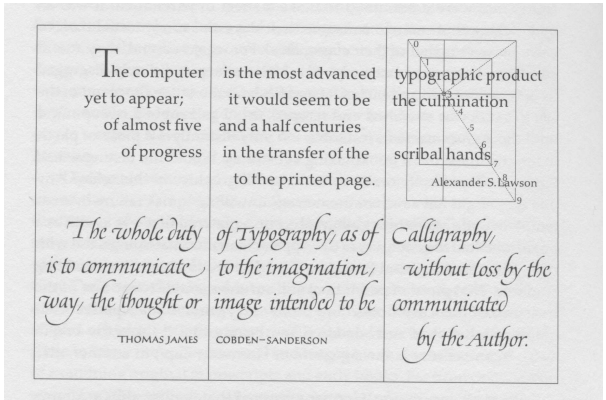
There are two versions of this book, the German version and an English translation and it is a pleasure to have both, especially because they are not entirely the same. The German version has a few more pages than the English translation. And not only because of the language, there are also true differences in the contents.

Editor's Note: First published in *MAPS* 35, 2007. Reprinted with permission.



Born on November 8, 1918, Hermann has grown up in and been a witness to turbulent times. The German version sheds more light on how difficult it was to survive in these times and how much art was lost in that period. He wrote down nice anecdotes about this era, for instance how the ability to write in 1 mm script impressed his army superiors so much that it kept him out of trouble. Both books have some differences in the graphics that go with that period and in the English version some quotes are shortened.

The English book catches up on its last pages. Since 1977 Hermann Zapf has been an associate professor at the Rochester Institute of Technology. In the postscript to this version the curator describes the influence Hermann has had on them in the past 30 years. At the time we write this review, Hermann is visiting this institute, where he is involved in a calligraphic and typographic display on 27 glass panels surrounding the new facilities.



If you manage to lay hands on a copy, you will notice that it's printed on thick cream-colored paper and very well bound in a dark blue hard cover with gold initials on the front. At the traditional Dante Christmas Party in 2006 in Darmstadt, Herman told the audience that nowadays it's not trivial to get such paper in the quantities needed: most paper plants only produce paper of moderate quality in any bulk. But here, large quantities of special paper were needed; keep in mind that he gave away a free copy to each of the more than 2000 Dante members.

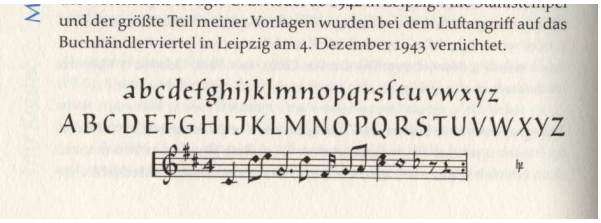
Other interesting differences between the versions are in paragraph breaks and whitespace. As with Dutch, German needs a few more words than English to express ideas, but the general impression is that the German version is the most informative. Other subtle differences are in the technical terms used. The English version qualifies Palatino Sans as 'sans serif', but the German text talks about 'Grotesk'.

A lifetime

One possible reason why Hermann has always been able to catch up with technology and could adapt quite well to the transition from lead to computer, was that originally he wanted to be an 'Elektroingenieur', but calligraphy attracted him more.

Hermann was never stuck on characters only. The book starts with a colorful full-page illustration of flowers and small beetles.

Also, in his early period he created a few 'Notenschriften'. The book shows many examples of handwriting and the grand finale is Zapfino, which is available as a OpenType font with many (complex) features.

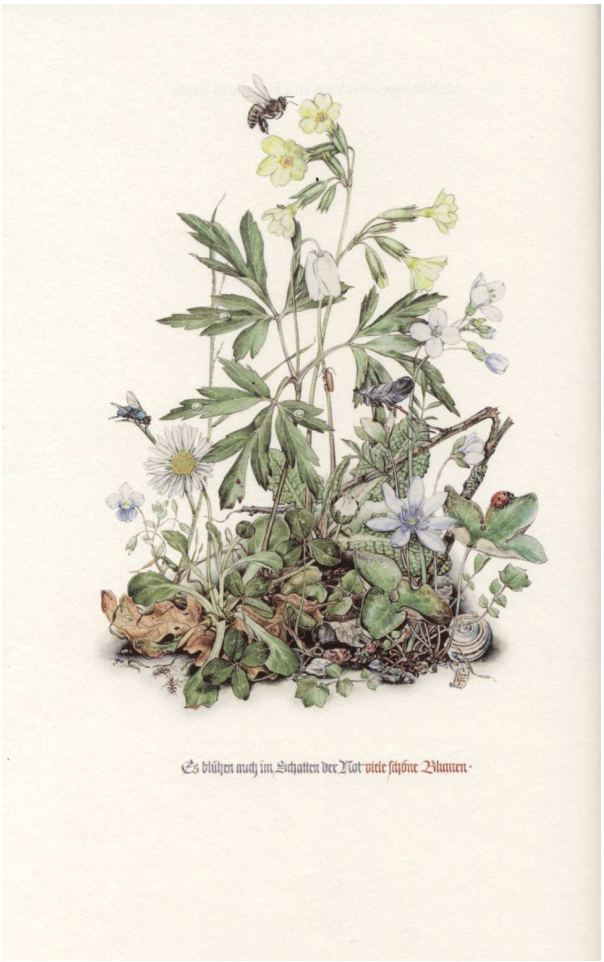

und der größte Teil meiner Vorlagen wurden bei dem Luftangriff auf das Buchhändlerviertel in Leipzig am 4. Dezember 1943 vernichtet.

abcdefghijklmnopqrstuvwxyz

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Greek, Arabic, you name it . . . he draws it. Among his most well known fonts are Optima and Palatino. Both fonts date back half a century when lead was still leading, but they were recently redesigned to take advantage of new technologies. Last year a sans serif family named Palatino Sans was added, and an Arabic variant is in the making.

The first Optima was drafted on thousand lire notes in 1950. In 1975, this font was used for the Vietnam Veterans Memorial in Washington.


Es blühen auch im Schatten der Not viele schöne Blumen.

Hermann spent quite some time in the USA, running his own company there, teaching at several designer schools and working with Donald Knuth. He is still associated with the Rochester Institute of Technology in New York.

In pdfTEX there is a feature that informally is called 'hz-optimization'. This feature is inspired by the work of Hermann on the 'hz-Programm' and in the book Hàn Thế Thành's work and Hermann's communication with Hàn Thế Thành are explicitly mentioned.

Although an old printing press has a prominent position in his house in Darmstadt, Hermann has always been involved in new technologies. He went from typesetting in lead to using phototypesetters to computer based typesetting. The Zapfino font, that adapts its choice of glyphs to the circumstances, is a prime example of this. Steve Jobs of Apple Computers made sure that on this platform Zapfino behaves as it should.

For those who use the dingbats there is good news as well. The Zapf Essentials are the improved and extended version of these symbols. We now finally have everything available that Hermann originally had in mind when he began drafting this symbol set.

The book also shows samples of Zapfino Ink, yet another innovation. Here color and shades make their way into the font but we have to wait till the font technologies are ready for that. The book tells us that this is being worked on.

The book is typeset in Palatino Nova with displayed quotations in the brand new Palatino Sans. In the not too wide margin keywords are typeset. These are rotated 90 degrees and printed in blue, which adds a very nice touch to the book's typographic feel. Especially so where the keyword in question is actually a font name, because each of those is typeset in the font that is indicated.

Afterword

At the Dante Christmas Party 2006 we showed Hermann some of his work on a digital ink device and he seemed quite impressed with what new technologies can provide. However we fully agree with the following quote from his monograph:

> *Ein gedruckter Buchstabe und ein schön gestaltetes Buch sind etwas Beständiges, Bleibendes im Vergleich zu dem schnellen Zugriff zu einer Information im Internet und dessen Flüchtigkeit der Wiedergabe am Bildschirm. Es ist das etwas schwer zu beschreibende eigenartige Erlebnis des Lesers, wenn er ein Buch in seiner Händen hält. Ein Buch spricht die Sinne an, der Druck auf dem Papier, das Umblättern der Seiten, ganz im Gegensatz zu der abstrakten elektronischen Darstellung eines Textes.*[1]

Both language versions of the monograph can be ordered directly from the Merchandise section of the Linotype website, `http://www.linotype.com/26/merchandise.html`.

If you prefer to order elsewhere, the ISBN number is 3-9810319–5-4 for 'Alphabetgeschichten', or 3-98103129-6-2 for 'Alphabet Stories'.

◇ Hans Hagen & Taco Hoekwater
NTG
`http://www.ntg.nl`

[1] The printed letter — or a well-designed book — is something very unique compared to the fleeting resolution of a screen and quick access on the Internet. A book offers a happy feeling in the hands of the reader and is quite different from an abstract presentation of text.

Fonts

An exploration of the Latin Modern fonts

Will Robertson

Abstract

The Latin Modern fonts are a newly-created set of fonts with the principal aim of providing glyphs for as many languages as possible. There is a multitude of little-known font shapes in the package, however, and these will be explored here.

1 Introduction

The Latin Modern family is a relatively recent collection of fonts authored by Bogusław Jackowski and Janusz M. Nowacki [1]. They are intended as the successors to Donald Knuth's Computer Modern fonts for the Unicode age, to provide the means for typesetting as many languages as possible that use the Latin-based alphabet. The collection is vast: it contains 72 text fonts, each containing almost seven hundred glyphs, at time of writing, with more probable in the future. That's some 50,000 glyphs in total! A very small number of the glyphs are shown in figure 1, chosen mostly at random for their interesting shapes. The maths fonts in the collection are not considered in this article, as they are equivalent to the Computer Modern fonts they are based on.

The Latin Modern fonts have been created with the MetaType1 system [2], whose programmatic nature makes the idea of dealing with such a huge number of glyphs even possible. The number of fonts in the collection is greater than the BlueSky Computer Modern Type 1 fonts [3] now used by default by all current LaTeX distributions, but fewer than in the enormous CM-Super collection (which also provides many glyphs for multilingual typesetting), whose fonts have been auto-traced from bitmaps and hence are of slightly inferior quality [4]. With the most recent releases, OpenType versions of the fonts have been made available for more general use. In this article, we shall look at the fonts the Latin Modern family provides and how they may be accessed in LaTeX.

2 NFSS refresher

To provide context, some brief details of LaTeX's font selection scheme are expounded here. Refer to the documentation [5] for further information.

Editor's note: Reprinted from *The PracTEX Journal* 2006-1 (`http://tug.org/pracjourn`), by permission.

Figure 1: Ten of the 50000-odd glyphs in the Latin Modern collection.

Three main families are defined for a document: the default roman, sans serif, and typewriter fonts. These are selected with the `\rmfamily`, `\sffamily`, and `\ttfamily` commands, respectively. Font families are requested with `\fontfamily{...}`; all such `\font...` commands (more to be seen) must be followed by `\selectfont`, if nothing else, to perform the actual font selection.

Variations along two other font axes (other than family) are possible: series and shape. The series axis is used to express weight and width, such as bold or condensed, and combinations thereof. We will be using the `\fontseries{...}` command later to look at various weights of the Latin Modern fonts. The shape axis is used to express italics and small caps, among other more esoteric options. We shall be content in the shape axis to use the commands `\itshape`, `\slshape`, and `\scshape`[1] to choose between the italic, oblique, and small caps shapes. Note that when only slanted shapes are available, `\itshape` will generally also select them.

How do we discover all the codes used to express the families, series and shapes for each font? These are all defined within font definition (`.fd`) files, which are supplied one per font encoding. The most common encoding is `T1`, which provides glyphs for many, but not all, European languages. To discover the font shapes available in the Latin Modern collection, then, these files must be located within the TeX distribution. They are found in the `texmf/tex/latex/lm` directory (where this is located will be system dependent), and investigation here will yield all of Latin Modern's secrets.

The encodings currently supported by the Latin Modern fonts in LaTeX are: `T1`, for most European languages; `TS1`, a large collection of miscellaneous symbols to accompany `T1`. `QX`, a variant of `T1` that is more suitable for Slavonic languages (also including the fk ligature, cf. fk); `LY1`, an alternative to `T1` that supports a mixture of common symbols and accented letters; `T5`, for Vietnamese; `OT1`, for emulating TeX's original ad-hoc font encoding; `OT4`, an obsolete encoding based on OT1 that supports Polish; `IL2`, a 'nonstandard' encoding suitable for Czech fonts; and, `L7X`, a 'nonstandard' encoding for Lithuanian.

[1] Or `\textit{...}`, `\textsl{...}`, `\textsc{...}`, respectively, which change the font of their argument instead.

3 The same-old

Everyone is familiar with the default TEX fonts. The Latin Modern fonts are selected with, in the preamble,[2]

```
\usepackage{lmodern}
\usepackage[T1]{fontenc}
```

which should make barely any visible changes to already existing documents; these fonts are an *extension* of Computer Modern, not a new design.

To begin, the three default families are shown, using common LATEX font selecting commands. In the examples shown in this article, indented entries indicate that the previous outdented command(s) are still active.

Roman Perhaps simply because he could, Knuth included a large amount of variation in the fonts he designed for TEX. Certainly, no one since has really matched his efforts. The descendants of his fonts still bear this curious hallmark: the Latin Modern Roman family contains both slanted *and* italic shapes.

`\rmdefault`	LM Roman
`\itshape`	*LM Roman Italic*
`\slshape`	*LM Roman Oblique*
`\scshape`	LM Roman Small Caps
`\bfseries`	**LM Roman Bold Extended**
`\itshape`	***LM Roman Bold Italic Extended***
`\slshape`	***LMR Bold Oblique Extended***

Sans serif Variations here must wait until later; here are the 'standard four'. Note that the sans serif family does not have a true italic, nor small caps.

`\sffamily`	LM Sans
`\slshape`	*LM Sans Oblique*
`\bfseries`	**LM Sans Bold**
`\slshape`	***LM Sans Bold Oblique***

Typewriter The italic shape here is perhaps a little unpleasant, and the fact that it has small caps is quite unusual considering that the sans serif family does not.

`\ttfamily`	`LM Typewriter`
`\itshape`	*`LM Typewriter Italic`*
`\slshape`	*`LM Typewriter Oblique`*
`\scshape`	`LM Typewriter Small Caps`
`\bfseries`	**`LM Typewriter Dark`**
`\slshape`	***`LM Typewriter Dark Oblique`***

The majority of the shapes demonstrated above are available in the Computer Modern fonts (that is, the current LATEX defaults). The bold ('Dark') typewriter fonts above, however, are completely new to Latin Modern. While the original METAFONT fonts are parameterised such that changes like this were easily possible, its bitmap output format is very outdated and rarely used these days.

[2] Change `T1` to another option (`LY1`, `QX`, `T5`, etc.), or combination thereof, depending on which glyphs you require/which language(s) you are typesetting.

4 Interlude — optical sizes

In the old days of printing, fonts were made of metal and were literally one to a size. The characters in a font for the body text of a book would look noticeably different to that same font at a larger size for titling. Nowadays, computer-based fonts can be scaled linearly to any size imaginable, but well designed fonts are still made available with variations based on the intended size of the output. In brief, the smaller a font is, the less fine its intricacies must be in order to survive the transfer from (possibly imperfect) printed page or low-resolution screen to eye. Conversely, a font designed to be large can be more delicately rendered.

For the original Computer Modern fonts, designed in METAFONT, the optical size could be chosen exactly for any size. Due to constraints on early computers, specific sizes were chosen as canonical, which were then inherited when they were converted to the PostScript Type 1 format. The Latin Modern fonts, in turn, also preserve these canonical sizes for all of the 'major' shapes, although such a profusion of optical sizes is almost certainly unnecessary, since there needn't be such a great range of font sizes in a single document.

The set of optical sizes for Latin Modern Roman is shown in figure 2, the largest number for any of the Latin Modern families. The non-linear nature of the scaling is immediately apparent, and it is quite clear how the characteristics change from robust to delicate, most significantly in the widths and stroke thicknesses of the characters, as the design size increases.

The Latin Modern fonts with a range of optical sizes are: roman upright, italic, oblique, and bold

Latin Modern Roman, design size 5 pt
Latin Modern Roman, design size 6 pt
Latin Modern Roman, design size 7 pt
Latin Modern Roman, design size 8 pt
Latin Modern Roman, design size 9 pt
Latin Modern Roman, design size 10 pt
Latin Modern Roman, design size 12 pt
Latin Modern Roman, design size 17 pt

Figure 2: The optical size range of Latin Modern Roman, each font at 10 pt.

extended; sans upright and oblique; and typewriter upright. These optical size variations constitute 32 of 69 fonts in the collection.

5 Non-default weights

As previously mentioned, the Latin Modern collection shares with the Computer Modern fonts some shapes that are not often used in practice, probably due to the fact that they can't be accessed with the 'normal' NFSS commands such as `\emph` and `\textbf`.

5.1 Other bold shapes

A non-extended version of the roman bold exists. Unfortunately, it is available in but a single design size (unlike its extended counterpart), and lacks true italics.

`\bfseries` **LM Roman Bold Extended**
`\fontseries{b}\selectfont` **LM Roman Bold**
`\fontseries{b}\slshape` ***LM Roman Bold Oblique***

The sans serif family also has a 'secret' bold shape:

```
\sffamily
  \bfseries          LM Sans Bold
  \fontseries{sbc}
    \selectfont      LM Sans Demi Condensed
    \slshape         LM Sans Demi Condensed Oblique
```

5.2 Italic small caps

The slantsc package allows `\slshape` and `\scshape` to be combined in order to select oblique small caps. (Or `\itshape` for truly italic small caps if they exist.) With `\usepackage{slantsc}`, it is possible to select

`\scshape\slshape` *LM Roman Oblique Small Caps*
`\ttfamily\scshape\slshape`
LM Typewriter Oblique Small Caps

Oblique or italic small caps are scarce in traditional typesetting, but their use is becoming more popular in modern times.

5.3 The new typewriter shapes

Quite recently in the lifetime of the Latin Modern collection, the typewriter fonts have been supplemented with extra shapes, including the 'Typewriter Dark' fonts previously seen. Also present are light and condensed light shapes, the latter being a $\frac{2}{3}$ reduction in width; that is, 120 characters in condensed light will fit in the space for 80 regular typewriter letters. Note that every character in every weight and shape of the typewriter fonts has the same width so that the letter grid remains constant when switching between styles.

```
\ttfamily
  \fontseries{b}
    \selectfont    LM Typewriter Dark
    \slshape       LM Typewriter Dark Oblique
  \fontseries{l}
    \selectfont    LM Typewriter Light
    \slshape       LM Typewriter Light Oblique
  \fontseries{lc}
    \selectfont    LM Typewriter Light Condensed
    \slshape       LM Typewriter Light Condensed Oblique
```

One may wonder why the light weights were produced. As the medium typewriter face is relatively heavy, it does not have much contrast with the new dark weight; compare the example on page 3 with the one on the previous page. So, in situations in which the bold face is to be used, the light face should be selected as the 'normal' typewriter weight. See figure 3 for preamble code to effect this.

```
\DeclareFontFamily{T1}{lmtt}{}
\DeclareFontShape{T1}{lmtt}
  {m}{n}{<-> ec-lmtl10}{}
\DeclareFontShape{T1}{lmtt}
  {m}{\itdefault}{<-> ec-lmtlo10}{}
\DeclareFontShape{T1}{lmtt}
  {\bfdefault}{n}{<-> ec-lmtk10}{}
\DeclareFontShape{T1}{lmtt}
  {\bfdefault}{\itdefault}{<-> ec-lmtko10}{}
```

Figure 3: Code to select lightface typewriter by default. For the `T1` encoding; adapt as required for the other encodings by looking in the `...lmtt.fd` files, as discussed in section 2.

6 Other families

As well as the secret weights mentioned above, there are entire *families* in the Latin Modern collection of which many people may be unaware.

6.1 Sans extended

The family 'Latin Modern Sans Extended' (sometimes referred to as 'Sans Quotation' due to Knuth's original use for it) is an extended version of the default sans serif family, intended for use at small font sizes (its nominal design size is 8 pt).

```
\renewcommand\sfdefault{lmssq}
```

```
\sffamily          LM Sans Extended
  \slshape         LM Sans Extended
  \bfseries        LM Sans Extended
    \slshape       LM Sans Extended
```

The variation in sans bold is interesting with regard to the condensed sans shown in section 5.1, but the shapes aren't entirely suitable for combination since they have different x-heights arising from their different design sizes:

Condensed Bold Extended

```
\DeclareFontFamily{T1}{lmvtt}{}
\DeclareFontShape{T1}{lmvtt}
  {m}{n}{<-> ec-lmvtl10}{}
\DeclareFontShape{T1}{lmvtt}
  {m}{\itdefault}{<-> ec-lmvtlo10}{}
\DeclareFontShape{T1}{lmvtt}
  {\bfdefault}{n}{<-> ec-lmvtk10}{}
\DeclareFontShape{T1}{lmvtt}
  {\bfdefault}{\itdefault}{<-> ec-lmvtko10}{}
```

Figure 4: Preamble code to select the lightface variable width typewriter by default.

6.2 Typewriter proportional

As the era of teletext computers becomes ever more distant, perhaps the idea of a fixed width font can be thought to be archaic. The Latin Modern Typewriter family has an accompanying variable width design, for those who wish to use it:

```
\renewcommand\ttdefault{lmvtt}
\ttfamily          LMTT Proportional
  \slshape         LMTT Proportional Oblique
  \fontseries{l}
    \selectfont    LMTT Proportional Light
    \slshape       LMTT Proportional Light Oblique
  \fontseries{b}
    \selectfont    LMTT Proportional Dark
    \slshape       LMTT Proportional Dark Oblique
```

It can be seen that here, as in the fixed-width typewriter fonts, every alphabet has the same horizontal width. Again, if the bold face is to be used for contrast, better results will be achieved by selecting the light face as default. This can be effected in a similar manner as before (section 5.3, refer in this case to `t1lmvtt.fd`); see figure 4.

6.3 Odd shapes

These fonts exist as examples to demonstrate the 'meta-ness' of the Computer Modern family, in that obliqueness of the italics and the stem height of the roman, to name but two parameters in the design, may be varied orthogonally. Their use is not particularly widespread.

The Dunhill family is named after the cigarette, for obvious reasons:

```
\fontfamily{lmdh}\selectfont
```
Latin Modern Dunhill

```
\fontfamily{lmdh}\slshape
```
LM Dunhill Slanted

There is also an 'upright italic' font, which I find quite unusual:[3]

```
\fontshape{ui}\selectfont
```
Latin Modern Unslanted italic

7 Conclusions

This concludes our tour of the different shapes of the Latin Modern font collection, which are the more multilingual replacements of the vector Computer Modern fonts. They have been exhibited in the belief that they are not as well known as they deserve, for much time and effort has been spent to supplement each of the fonts with hundreds of extra glyphs.

We have seen some shortfalls and awkwardness with LaTeX's font selection scheme in being able to select, in a straightforward manner, the large variety of shapes and weights that the collection offers. Brief examples detailing how to overcome these problems have been given, but more work is required for a flexible generic solution. In the future, we look forward to the creation of a better user interface for this purpose, either specifically for these fonts, or in general with a 'newer' font selection scheme.

Bibliography

[1] Bogusław Jackowski and Janusz M. Nowacki. Latin Modern: Enhancing Computer Modern with accents, accents, accents. *TUGboat*, 24(1): 64–74, 2003. `http://www.tug.org/TUGboat/Articles/tb24-1/jackowski.pdf`.

[2] Bogusław Jackowski, Janusz M. Nowacki, and Piotr Strzelczyk. Programming PS Type 1 fonts using MetaType1: Auditing, enhancing, creating. *TUGboat*, 24(3):575–581, 2003. `http://www.tug.org/TUGboat/Articles/tb24-3/jackowski.pdf`.

[3] Computer Modern and AMSFonts in Type 1 (PostScript) form. `http://www.ams.org/tex/type1-fonts.html`.

[4] Vladimir Volovich. CM-Super: Automatic creation of efficient Type 1 fonts from METAFONT fonts. *TUGboat*, 24(1):75–78, 2003. `http://www.tug.org/TUGboat/Articles/tb24-1/volovich.pdf`.

[5] LaTeX3 Project Team. *LaTeX 2ε font selection*. `http://www.latex-project.org/guides/fntguide.pdf`.

[3] This will work in the next release of the Latin Modern fonts; at time of writing the font exists but the font definition for LaTeX is missing.

Creation of a PostScript Type 1 logo font with MetaType 1

Klaus Höppner

Abstract

MetaType 1 is a tool created by Bogusław Jackowski, Janusz Nowacki, and Piotr Strzelczyk for creating PostScript Type 1 fonts. It uses METAPOST, t1utils and some AWK scripts to start from a METAPOST source with some special macros, resulting in the AFM, TFM and PFB files needed to use the font as any other PostScript font.

MetaType 1 was used to create the Latin Modern fonts, derived from Computer Modern fonts but including many more accented characters and nowadays part of most TEX distributions. Other new fonts such as Iwona and Kurier have also been created by the developers of MetaType 1.

I came into contact with METAPOST when I wanted to convert an existing logo font from METAFONT to PostScript Type 1. Unfortunately there doesn't yet exist a tutorial or cookbook for using MetaType 1. So I started to play with the example fonts supplied as part of MetaType 1 and to read the comments in the source. This tutorial will give an example and the lessons I learned.

1 Introduction

When Donald E. Knuth invented TEX, he also created his own description language for high quality fonts. It was named METAFONT. So the process from a TEX source to some paperwork was as follows: Compile the TEX source to get a DVI file that contains references to the fonts that were used in the document — in fact the only thing that TEX knows about a font is its metrics. To produce the document on paper, the DVI driver invoked METAFONT (the program) to convert the METAFONT source of the font, i. e. the geometrical description of the font outlines, to a bitmapped font suited for the resolution and technical details of the printer by using the METAFONT mode for this special printer.

While this approach works fine if you work alone and just send your documents to your personal printer, it has some disadvantages if you want to exchange documents electronically. Normally, distributing DVI isn't the best idea, since it requires that the recipient has a TEX system installed including all fonts that were used in your document — not to mention any graphics included in your document. So in most cases you will send a PostScript file or nowadays a PDF file. In this case, all the fonts from METAFONT sources will be embedded as bitmapped PostScript Type 3 fonts. When the recipient prints your document, it may look fine, but it may look poor if the METAFONT mode used to create the bitmapped font didn't match the printer, and the document will probably look very poor on the screen (especially in old versions of Acrobat Reader).

So when exchanging documents, it is preferable to embed the fonts as outline fonts. For these, the usual format used in the TEX world is PostScript Type 1 (though this is gradually being replaced by OpenType). The Type 1 format uses a subset of the well established PostScript language.[1]

Meanwhile, most of the fonts used in the TEX world are available as PostScript Type 1 fonts, starting with the Type 1 version of Knuth's CM fonts up to the Latin Modern fonts that augment CM with a complete set of diacritic characters.

2 MetaType 1

MetaType 1 is the tool that was used to create the Latin Modern fonts from the METAFONT sources of CM fonts, and for the creation of completely new fonts such as Iwona.

MetaType 1 relies on METAPOST, a variant of METAFONT producing small pieces of PostScript as output, written by John Hobby. Bogusław Jackowski, Janusz Nowacki, and Piotr Strzelczyk wrote a set of METAPOST macros and added some AWK scripts to create the input files that can be converted to Type 1 with t1utils. Thus, one advantage of MetaType 1 is that it uses a source format that is very similar to the old METAFONT sources.

3 Our example

I came into touch with MetaType 1 when I wasn't satisfied with the DANTE logo being typeset from the old METAFONT source with all the disadvantages mentioned above. So I wanted to give MetaType 1 a try to convert the DANTE logo font into a PostScript Type 1 font.

Fortunately, the DANTE logo font contains just the characters needed to set the logo:

dante

So, it was just five characters for which the METAFONT source had to be made suitable to be processed with MetaType 1.

Unfortunately, I found out that the available documentation for MetaType 1 was rather limited: articles from conference talks [1, 2], the commented

[1] It is sometimes said that Type 1 fonts are outline fonts while Type 3 are bitmap fonts. That's not true, since Type 3 fonts may comprise both outlines and bitmaps.

source for the MetaType 1 macros and two sample fonts that are part of the MetaType 1 distribution.

But in the end, I found my way, and as you will see, was able to create my own Type 1 font. To make things a bit simpler for this tutorial, I will show the steps I made for a small test font with just two characters, "a" and "t", simplified compared to the original characters from the DANTE logo font. Hopefully it will make the presented source more understandable, even if you haven't programmed in METAPOST before.

3.1 Installation

Installing MetaType 1 was easy enough. I downloaded the ZIP archive file from CTAN [3] and copied the files to the appropriate locations of my local texmf tree: the `.mp` files into `metapost/mt1`, the `.mft` files into `mft/mt1`, the `.sty` files into `macros/generic/mt1`, and finally the `.awk` and `.dat` files into `scripts/mt1`.[2]

The main problem in my case was that MetaType 1 was shipped with a set of DOS batch files that are used to create the fonts, but I was using GNU/Linux. So I looked into these files to find out what they do — in fact they were rather simple, just calling METAPOST to produce a small PostScript file for every glyph in the font and then using some AWK scripts to merge and assemble these files into a raw PostScript font that is converted into PostScript Type 1 with t1asm (part of t1utils). So several immediate files and steps are involved, but the workflow is straightforward. Eventually, I wrote a small Makefile that does the job on a Unix system, as shown in listing 3. From this point, I could create the TFM, PFB and MAP files for a font with the command `make FONT=myfont`.

I also manually created an FD file for using the font in LaTeX. These files could all be installed into the appropriate locations inside a `texmf` tree. Testing of a font is convenient in pdfTeX since one can use a MAP file locally in a document using the `\pdfmapfile` primitive, while for a real font one normally will install the MAP file using the `updmap` script (or equivalent).

3.2 The first font

After these prerequisites were done, I could start with my first font. I copied the file `tapes.mp` (a sample font that is part of the MetaType 1 distribution) into `myfont.mp`, found several settings with font parameters starting with `pf_info_*`, changed them where appropriate (font name, family, creator, etc.) and kept the rest unchanged.

[2] This location isn't required since these files aren't found by the Kpathsea library, but instead via an environment variable, but at least this location seemed to be meaningful.

Listing 1: First definition of "a" and "t".

```
encode ("a") (ASCII "a");
introduce "a" (store+utilize) (0) ();
beginglyph("a");
path pa, pb, pc;
z0 = (round_hdist+radius,radius);
z1 = (round_hdist+2radius-strength,0);
pa = fullcircle scaled 2 radius shifted z0;
pb = reverse fullcircle
     scaled (2radius-2strength) shifted z0;
pc = unitsquare xscaled strength
     yscaled 2radius shifted z1;
Fill pa;
unFill pb;
Fill pc;
fix_hsbw(2radius+round_hdist+hdist,0,0);
endglyph;

encode ("t") (ASCII "t");
introduce "t" (store+utilize) (0) ();
beginglyph("t");
path pa, pb;
z0 = (hdist+3.5strength,1.5strength);
x1 = hdist + 2strength;
x2 = x1 + strength;
y1 = y2 = height;
z3 = (hdist,height-3strength);
pa = z1
     -- (halfcircle rotated 180
         scaled 3strength shifted z0)
     -- (reverse halfcircle rotated 180
         scaled strength shifted z0)
     -- z2 -- cycle;
pb = unitsquare xscaled 5strength
     yscaled strength shifted z3;
Fill pa;
Fill pb;
fix_hsbw(2hdist+5strength,0,0);
endglyph;
```

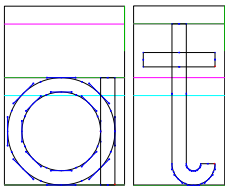
Then I defined the first two characters according to the following rule:

> *Characters consist of closed paths, filled or unfilled paths, where filled paths always turn counter clockwise and unfilled paths always clockwise.*

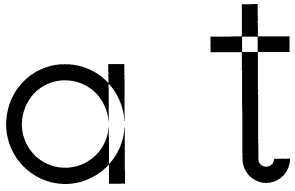
So when designing the letter "a", I defined an outer circle that was filled and then an inner circle to be unfilled and then a rectangular shape as vertical stem. And the letter "t" was just built from a vertical stem (with a hook at the right bottom) and a horizontal bar. The definitions for the characters are shown in listing 1.

Please notice in the definition of letter “a”, that the path for the outer circle is a (counter clockwise) fullcircle, while the inner circle is a reverse fullcircle, since the former one is filled while the latter one is unfilled. Filling and unfilling of the paths is done by the macros `Fill` and `unFill`; these macros warn you if the turning direction of the path is wrong.

Proofs for the glyphs are produced by compiling the file `myfont.mp` with METAPOST. As you can see, they really do look like an “a” and a “t”:



Now let’s see how the Type 1 font looks:



Something went wrong. After taking a closer look, it becomes obvious. The regions where filled paths overlap become unfilled. This is due to the fact that filling of paths is done with an *exclusive-or* fill, i. e. when filling a path, regions inside that are already black become white. As this isn’t what we want to achieve, we formulate another rule to keep in mind:

Paths must not overlap!

Although it is possible with pure METAPOST to find the intersection points of paths to remove overlapping parts, this tends to be painful. Since MetaType 1 was used to attach cedilla and ogonek accents to various characters in the extension of CM to LM, this painful work of finding the outline of two overlapping paths was encapsulated into a macro that is part of MetaType 1, named `find_outlines`. Let’s see how this macro is utilized for the letter “a”:

```
find_outlines(pa,pc)(r);
Fill r1;
```

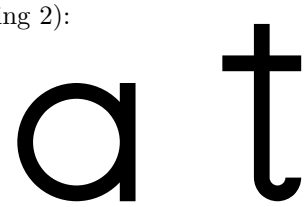
It finds the outline of the two overlapping paths `pa` and `pc`, with the result written in the path array `r`. The result is an array because the outline of the paths may consist of more than one path, but in our case it is just one path, accessible as `r1`. The same is applied for the letter “t” (just the names of the two paths slightly differ).

When filling the new outlines instead of the overlapping paths, we now get the following result:

So, obviously finding the outline path for the “t” worked, but it failed for the “a”. Why? Because in the case of the “a”, both paths touch in one point without crossing at the right side of the vertical stem, i. e. they have an intersection point with the same direction vector. This confuses the macro that finds the outlines since it doesn’t know which path to follow — and in this case it chooses wrong. So, let’s bear in mind another rule:

Paths must not touch tangentially!

To resolve the problem, we use a simple trick: Shift the vertical stem a tiny amount to the right, so that the paths don’t touch anymore. In METAPOST you can use `eps` as a tiny positive number (in mathematics, an arbitrary small number is usually denoted by ϵ). The following lovely characters are the result (the METAPOST definitions are shown in listing 2):



3.3 Kerning

Our glyphs are ready, but a normal font has more features, such as kerning pairs and ligatures. In the former case, for a pair of characters the horizontal spacing between them is changed, while in the latter case a character pair is replaced by another glyph.

Defining a kerning pair in MetaType 1 is simple. *After* the definition of the glyphs, we can add a kerning table. In our case it looks like this:

```
LK("a") KP("t")(-3ku); KL;
```

In the list of ligatures and kernings for the letter “a” we define a kerning of −3 ku if it is followed by the letter “t” to remove the optical gap between them (the kerning unit ‘ku’ is defined elsewhere in the METAPOST source). The effect of kerning is shown in figure 1.

Ligatures don’t make sense for our sample font, so I leave them out for this tutorial. In principle they work similarly; you merely define from which slot in the font the replacement for a specified character pair is to be taken.

Listing 2: Definition of “a” and “t” with outlines.

```
encode ("a") (ASCII "a");
introduce "a" (store+utilize) (0) ();
beginglyph("a");
path pa, pb, pc, r;
z0 = (round_hdist+radius,radius);
z1 = (round_hdist+2radius-strength+eps,0);
pa = fullcircle scaled 2 radius shifted z0;
pb = reverse fullcircle
     scaled (2radius-2strength) shifted z0;
pc = unitsquare xscaled strength
     yscaled 2radius shifted z1;
find_outlines(pa,pc)(r);
Fill r1;
unFill pb;
fix_hsbw(2radius+round_hdist+hdist,0,0);
endglyph;

encode ("t") (ASCII "t");
introduce "t" (store+utilize) (0) ();
beginglyph("t");
path pa, pb, r;
z0 = (hdist+3.5strength,1.5strength);
x1 = hdist + 2strength;
x2 = x1 + strength;
y1 = y2 = height;
z3 = (hdist,height-3strength);
pa = z1
     -- (halfcircle rotated 180
         scaled 3strength shifted z0)
     -- (reverse halfcircle rotated 180 scaled
         strength shifted z0)
     -- z2 -- cycle;
pb = unitsquare xscaled 5strength
     yscaled strength shifted z3;
find_outlines(pa,pb)(r);
Fill r1;
fix_hsbw(2hdist+5strength,0,0);
endglyph;
```

3.4 Hinting

When you embed fonts as outline fonts, you leave the task of rasterizing the glyphs to your output device (printer or viewer). Unfortunately, this final result may look rather poor, especially on low resolution devices such as screens. Imagine the letter “H” and how it is rasterized into pixels. If we’re unlucky, the left and right vertical stem will have a different width. On a printer with 1200 dpi it’s nearly unnoticeable, but on the screen a difference of one pixel makes it look quite ugly.

To prevent this, high quality fonts use a mechanism called “hinting” to help the rasterizer (e. g. the PostScript RIP in a printer) to keep vertical or horizontal stems the same width.

Figure 1: Our font without (top) and with (bottom) kerning.

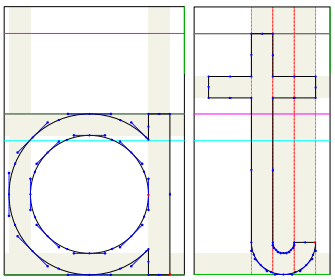


Figure 2: Hinting informations (shaded areas).

MetaType 1 supports hinting by providing the macros `fix_hstem` and `fix_vstem` that try to find horizontal or vertical stems of a given width and add hinting information for them. For example, since we know that our letters “a” and “t” have stems of the width `strength`, we add hinting information by

```
fix_hstem(strength,pa,pb);
fix_vstem(strength,pa,pb);
```

You can see what hinting information was found as shaded areas in the proofs (figure 2).

4 Conclusions

I found that MetaType 1 is a suitable tool to create PostScript Type 1 fonts. Though there is a lack of beginning documentation, I was able to create a first font quite quickly by relying on an existing METAFONT source. Of course, knowledge of METAPOST or METAFONT is highly desirable. Understanding hinting is a bit more difficult, but finally possible.

References

[1] Bogusław Jackowski, Janusz M. Nowacki, Piotr Strzelczyk, *MetaType 1: A MetaPost-based engine for generating Type 1 fonts*, Proc. of EuroTEX 2001, published in MAPS 26, 2001, 111–119.

[2] Bogusław Jackowski, Janusz M. Nowacki, Piotr Strzelczyk, *Programming PostScript Type 1 fonts using MetaType 1: Auditing, enhancing, creating*, *TUGboat*, volume 24 (2003), no. 3.

[3] `http://mirror.ctan.org/fonts/utilities/metatype1/`

Listing 3: Makefile for font creation with MetaType 1.

```
METATYPE1 = /home/klaus/texmf/scripts/mt1

.PHONY: pfb tfm proof all

all: pfb tfm
proof: $(FONT).pdf

pfb: $(FONT).pfb

tfm: $(FONT).tfm

%.p: %.mp
    mpost "\generating:=0; \input $<"
    gawk -f $(METATYPE1)/mp2pf.awk \
         -vCD=$(METATYPE1)/pfcommon.dat \
         -vNAME=`basename $< .mp`

%.pn: %.p
    gawk -f $(METATYPE1)/packsubr.awk \
         -vVERBOSE=1 -vLEV=5 -vOUP=$@ $<

%.pfb: %.pn
    t1asm -b $< $@

%.tfm: %.mp
    mpost "\generating:=1; \input $<"

%.pdf: %.ps
    ps2pdf $< $@

%.ps: %.dvi
    dvips -o $@ $<

%.dvi: %.tex
    tex $<

%.tex: %.mp
    mpost $<
    cp $< _t_m_p.mp
    mft _t_m_p.mp -style=mt1form.mft
    echo '\input mt1form.sty' > $@
    test -f piclist.tex && cat piclist.tex >> $@
    test -f _t_m_p.tex && cat _t_m_p.tex >> $@
    echo '\endproof' >> $@
```

Listing 4: The complete font.

```
% A sample font for PRACTEX2006
input fontbase;

% Global parameters for all characters
size := 1000; depth := 0; math_axis := 1/2size;
radius := 300; hight := 900; strength := 80;
ku := 18; hdist := 3ku; round_hdist := 1ku;

% Font settings
pf_info_familyname "MyFont";
pf_info_fontname "MyFont-Regular";
pf_info_weight "Normal";
pf_info_version "0.01";
pf_info_capheight hight;
pf_info_xheight 2radius;
pf_info_space 10ku;
pf_info_adl size, 0, 0;
pf_info_author "Made by KH for PRACTEX2006"
pf_info_overshoots (1000,10), (0, -10);
pf_info_encoding "at";
pf_info_creationdate;

beginfont

encode ("a") (ASCII "a");
introduce "a" (store+utilize) (0) ();
beginglyph("a");
path pa, pb, pc, r;
z0 = (round_hdist+radius,radius);
z1 = (round_hdist+2radius-strength+eps,0);
pa = fullcircle scaled 2 radius shifted z0;
pb = reverse fullcircle scaled (2radius-2strength)
     shifted z0;
pc = unitsquare xscaled strength yscaled 2radius
     shifted z1;
find_outlines(pa,pc)(r);
Fill r1;
unFill pb;
fix_hstem(strength,pa,pb,pc);
fix_vstem(strength,pa,pb,pc);
fix_hsbw(2radius+round_hdist+hdist,0,0);
endglyph;

encode ("t") (ASCII "t");
introduce "t" (store+utilize) (0) ();
beginglyph("t");
path pa, pb, r;
z0 = (hdist+3.5strength,1.5strength);
x1 = hdist + 2strength;
x2 = x1 + strength;
y1 = y2 = hight;
z3 = (hdist,hight-3strength);
pa = z1 -- (halfcircle rotated 180
           scaled 3strength shifted z0)
     -- (reverse halfcircle rotated 180
         scaled strength shifted z0)
     -- z2 -- cycle;
pb = unitsquare xscaled 5strength yscaled strength
     shifted z3;
find_outlines(pa,pb)(r);
Fill r1;
fix_hstem(strength,pa,pb);
fix_vstem(strength,pa,pb);
fix_hsbw(2hdist+5strength,0,0);
endglyph;

LK("a") KP("t")(-3ku); KL;
endfont.
```

Writing ETX format font encoding specifications

Lars Hellström

Abstract

This paper explains how one writes formal specifications of font encodings for LaTeX and suggests a ratification procedure for such specifications.

1 Introduction

One of the many difficult problems any creator of a new typesetting system encounters is that of *font construction* — to create fonts that provide all the information that the typesetting system needs to do its job. From the early history of TeX, we learn that this problem is so significant that it motivated the creation of TeX's companion and equal METAFONT, whose implementation proved to be an even greater scientific challenge than TeX was. It is also a tell-tale sign that the `fonts` subtree of the teTeX distribution is about three times as large as the `tex` subtree: fonts are important, and not at all trivial to generate.

The most respected and celebrated part of font construction is *font design* — the creation from practically nothing of new letter (and symbol) shapes, in pursuit of an artistic vision — but it is also something very few people have the time and skill to carry through. More common is the task of *font installation*, where one has to solve the very concrete problem of how to set up an existing font so that it can be used with (La)TeX. The subproblems in this domain ranges from the very technical — how to make different pieces of software "talk" to each other, for example making information in file format A available to program B — to the almost artistic — finding values for glyph metrics and kerns that will make them look good in text — but these extremes tend to be clearly defined even if solving them can be hard, so they are not what will be considered here. Rather, this paper is about a class of more subtle problems that have to do with how a font is organised.

The technical name for such a "font organisation" is a *font encoding*. In some contexts, font encodings are assumed to be mere mappings from a set of "slots" to a set of glyph identifiers, but in TeX the concept entails much more; the various aspects are detailed in subsequent sections. For the moment, it is sufficient to observe that the role that a font encoding plays in a typesetting system is that of a standard: it describes what an author can expect from a font, so that a document or macro package can be written that work with a large class of fonts rather than just for one font family. The world of (La)TeX would be very different if papers published in journal X that is printed in commercial font Y could not use essentially the same sources as the author prepared for typesetting in the free font Z. Fine-tuning of a document (overfull lines, bad page breaks, etc.) depends on the exact font used, but it is a great convenience that one can typeset a well-coded body of text under a rather wide range of layout parameters (of which the main font family is one) values and still expect the result to look decent, often even good. Had font encodings not been standardised, the results might not even have been readable.

When font encodings are viewed as standards, the historical states of most (La)TeX font encodings becomes rather embarrassing, as they lack something as fundamental as proper specifications! The typical origin of a font encoding has been that someone creates a font that behaves noticeably different from other fonts, macro packages are then created to support this new font, and in time other people create other fonts that work with the same macros. At the end of this story the new encoding exists, but it is not clear who created it, and there is probably no document that describes all aspects of the encoding. Later contributors have typically had to rely on a combination of imitation of previous works, folklore, and reverse engineering of existing software when trying to figure out what they need to provide, but the results are not always verifiable. Furthermore the errors in this area are usually silent — the classical error being that a '$' was substituted for a '£' (or vice versa) — which means they can only be discovered through careful proofreading, and then only if a document even exists which exercises all aspects of the font encoding. Since font encodings interact with hyphenation, exhaustive font verification through proofreading is probably beyond the capabilities of any living TeXpert on purely linguistic grounds.

Proper specifications of font encodings makes the task of font installation — and to some extent also the task of font design, as it too is subject to the technicalities of font encodings — much simpler, as there is then a document that authoritatively gives all details of a font encoding. This paper even goes one step further, and proposes (i) a standard format for formal specifications of (La)TeX font encodings and (ii) a process through which such specifications can be ratified as *the* specification of a particular encoding. My hope is that future (La)TeX font encodings will have proper specifications from the

start, as this will greatly simplify making more fonts available in these encodings, and perhaps also make font designers aware of the subtler points of (LA)TEX font design, as many details have been poorly documented.

The proposed file format for encoding specifications is a development of the fontinst [6] ETX format. One reason for this choice was that it is an established format; many of those who are making fonts already use it, even if for a slightly different purpose. Another major reason is that an ETX file is both a LATEX document and a processable data file; this is the same kind of bilinguality that has made the `.dtx` format so useful. Finally the ETX format makes it easy to create experimental font installations when a new encoding is being designed; fontinst can directly read the file, but the file can also be automatically converted to a PostScript encoding vector if that approach seems more convenient. On the other hand, there are some features — most notably the prominent role of the glyph names — of the ETX format that would probably had been done differently in a file format that was built from scratch, but this is necessary for several of the advantages listed above.

2 Points to keep in mind

2.1 Characters, glyphs, and slots

One fundamental difference that must be understood is that between characters and glyphs. A *character* is a semantic entity — it carries some meaning, even if you usually have to combine several characters to make up even one word — whereas a *glyph* simply is a piece of graphics. In printed text, glyphs are used to represent characters and the first step of reading is to determine which character(s) a given glyph is representing.[1]

In the output, TEX neither deals with characters nor glyphs, really (although many of its messages speak of characters), but with *slots*, which essentially are numbered positions in a font. To TEX, a slot is simply something which can have certain metric properties (width, height, depth, etc.) but to the driver which actually does the printing the slot also specifies a glyph. The same slot in two different fonts can correspond to two quite different characters.

For completeness it should also be mentioned that the *input* of TEX is a stream of semantic entities and thus TEX is dealing with characters on that side, but the input is not the subject of this paper.

2.2 Ligatures

In typography, a *ligature* is a glyph which has been formed by joining glyphs that represent two or more characters; this joining can involve quite a lot of deformation of the original shapes. Examples of ligatures are the 'fi' ligature (from 'f' and 'i'), the 'Æ' ligature (from 'A' and 'E'), and the '&' character (from 'E' and 't'), the latter two of which has evolved to become characters of their own. For those ligatures (such as 'fi') that have not evolved to characters, TEX has a mechanism for forming the ligature out of the characters it is composed from, under the guidance of ligature/kerning programs found in the font.

More technically, what happens is that if the `\char` (or equivalent) for one slot is immediately followed by the `\char` (or equivalent) for another (or the same) slot and there is a ligaturing instruction in the `LIGKERN` table of the current font which applies to this slot pair then this ligaturing instruction is executed. This usually replaces the two slots in the pair with a single new slot specified by the ligaturing instruction (it could also keep one or both of the original slots, but that is less common). TEX has no idea about whether these replacements change the meaning of anything, but TEX assumes that it doesn't, and it is up to the font designer to ensure that this is the case.

Apart from forming ligatures in text, the ligaturing mechanism of TEX is traditionally also employed for another task which is much more problematic. Ligatures are also used to produce certain characters which are not part of visible ASCII — the most common are the endash (typed as `--`) and the emdash (typed as `---`). This is a problem because it violates TEX's assumption that the meaning is unchanged; the classical problem with this appears in the `OT2` encoding, where the Unicode character `U+0446` (CYRILLIC SMALL LETTER TSE) could be typed as `ts`, whilst the `t` and `s` by themselves produced Unicode characters `U+0442` (CYRILLIC SMALL LETTER TE) and `U+0441` (CYRILLIC SMALL LETTER ES) respectively. TEX's hyphenation mechanism can however decompose ligatures, so it sometimes happened that the TSE was hyphenated as TE-ES, which is quite different from what was intended. Since this is such an obvious disadvantage, the use of ligatures for forming non-English letters quickly disappeared after 8-bit input encodings became available. The practice still remains in use for punctuation, however, and the font designer must be aware of this.

[1] Some PDF viewers also try to accomplish this, but in general they need extra information to do it right. The generic solution provided is to embed a *ToUnicode CMap* — which is precisely a map from slots to characters — in the PDF font object.

For many font encodings there is a set of ligatures which must be present and replace two or more characters by a single, different character. These ligatures are called *mandatory ligatures* in this paper.

The use of mandatory ligatures in new font encodings is strongly discouraged, for a number of reasons. The main problem is that they create unhealthy dependencies between input and output encoding, whereas these should ideally be totally independent. Using ligatures in this way complicates the internal representation of text, and it also makes it much harder to typeset text where those ligatures are not wanted (such as verbatim text). Furthermore it creates problems with kerning, since the "ligature" has not yet been formed when a kern to the left of it is inserted. Finally, a much better solution (when it is available) is to use an Omega translation process (see [9, Sec. 8–11]), since that *is* independent of the font, different translations can be combined, and they can easily handle even "abbreviations" much more complicated than those ligatures can deal with.

2.3 Output stages

On its way out of LaTeX towards the printed text, a character passes through a number of stages. The following five seem to cover what is relevant for the present discussion:

1. The *LaTeX Internal Character Representation* (LICR); see [8], Section 7.11, for a full description. At this point the character is a character token (e.g. `a`), a text command (e.g. `\ss`), or a combination (e.g. `\H{o}`).
2. *Horizontal material;* this is what the character is en route from TeX's mouth to its stomach. For most characters this is equivalent to a single `\char` command (e.g. `a` is equivalent to `\char97`), but some require more than one, some are combined using the `\accent` and `\char` commands, some involve rules and/or kerns, and some are built using boxes that arbitrarily combine the above elements.
3. *DVI commands;* this is the DVI file commands that produce the printed representation of the character.
4. *Printed text;* this is the graphical representation of the character, e.g. as ink on paper or as a pattern on a computer screen. Here the text consists of glyphs.
5. *Interpreted text;* this is essentially printed text modulo equivalence of interpretation, hence the text doesn't really reach this stage until someone reads it. Here the text consists of characters.

In theory there is a universal mapping from LICR to interpreted text, but various technical restrictions make it impossible to simultaneously support the entire mapping. A LaTeX encoding selects a restriction of this mapping to a limited set which will be "well supported" (meaning kerning and such between characters in the set works), whereas elements outside this set at best can be supported through temporary encoding changes. The encoding also specifies a decomposition of the mapping into one part which maps LICR to horizontal material and one part which maps horizontal material to interpreted text. The first part is realized by the text command definitions usually found in the '⟨*enc*⟩`enc.def`' file for the encoding. The second part is the font encoding, the specification of which is the topic of this paper. It is also worth noticing that an actual font is a mapping of horizontal material to printed text.

An alternative decomposition of the mapping from LICR to interpreted text would be at the DVI command level, but even though this decomposition is realized in most TeX implementations, it has very little relevance for the discussion of encodings. The main reason for this is that it depends not only on the encoding of a font, but also on its metrics. Furthermore it is worth noticing that in e.g. pdfTeX there need not be a DVI command level.

2.4 Hyphenation

There are strong connections between font encoding and hyphenation because TeX's hyphenation mechanism operates on horizontal material; more precisely, the hyphenation mechanism only works on pieces of horizontal material that are equivalent to sequences of `\char` commands. This implies that hyphenation patterns, as selected via the `\language` parameter, are not only for a specific language, they are also for a specific font encoding.

The hyphenation mechanism uses the `\lccode` values to distinguish between three types of slots:

1. lower case letters (`\lccode`$n = n$),
2. upper case letters (`\lccode`$n \notin \{0, n\}$), and
3. non-letters (`\lccode`$n = 0$).

Only the first two types can be part of a hyphenatable word and only lower case letters are needed in the hyphenation patters. This does however place severe restrictions on how letters can be placed in a text font because TeX uses the same `\lccode` values for all text in a paragraph and therefore these values cannot be changed whenever the encoding changes. In LaTeX the `\lccode` table is not allowed to change at all and consequently all

text font encodings must work using the standard set of `\lccode` values.

In ε-TEX each set of hyphenation patterns has its own set of `\lccode` values for hyphenation, so the problem isn't as severe there. The hyphenation mechanism of Omega should become completely independent of the font encoding, although the last time I checked it was still operating on material encoded according to a font encoding.

2.5 Production and specification ETX files

Finally, it is worth pointing out the difference between an ETX file created for the specification of a font encoding and one created for being used in actually producing fonts with this encoding. They are usually not the same. Although specification ETXs certainly may be of direct use in the production of fonts — especially experimental fonts produced as part of the work on a new encoding — they are usually not ideal for the purpose. In particular there is often a need to switch between alternative names for a glyph to accommodate what is actually in the fonts, but such trickeries are undesirable complications in a specification. On the other hand a production ETX file has little need for verbose comments, whereas they are rather an advantage in a specification ETX file.

Therefore one shouldn't be surprised if there are two ETX files for a specific encoding: one which is a specification version and one which is a production version. If both might need to be in the same directory then one should, as a rule of thumb, include a '`spec`' in the name of the specification version.

3 Font encoding specifications

3.1 Basic principles

Most features of the font encoding are categorized as either *mandatory* or *ordinary*. The mandatory features are what macros may rely on, whereas the ordinary simply are something which fonts with this encoding normally provide. Font designers may choose to provide other features than the ordinary, but are recommended to provide the ordinary features to the extent that available resources permit.

Many internal references in the specification are in the form of *glyph names* and the choice of these is a slightly tricky matter. From the point of formal specification, the choices can be completely arbitrary, but from the point of practical usefulness they most likely are not. One of the main advantages of the ETX format for specifications is that such specifications can also be used to make experimental implementations, but this requires that the glyph names in the specification are the same as those used in the fonts from which the experimental implementation should be built. Yet another aspect is that the glyph names are best chosen to be the ones one can expect to find in actual fonts, as that will make things easier for other people that want to make non-experimental implementations later. For this last purpose, a good reference is Adobe's technical note on Unicode and glyph names [3]. For most common glyphs, [3] ends up recommending that one should follow the Adobe glyph list [2], which however has the peculiar trait of recommending names on the form `afii`*ddddd* (rather than the Unicode-based alternative `uni`*xxxx*) for most non-latin glyphs. This is somewhat put in perspective by [1].

3.2 Slot assignments

The purpose of the slot assignments is to specify for each slot the character or characters to which it is mapped. That one slot is mapped to many characters is an unfortunate, but not uncommon, reality in many encodings, as limitations in font size have often encouraged identifications of two characters which are almost the same. It should be avoided in new encodings.

Slot assignments are done using the `\nextslot` command and a `\setslot ... \endsetslot` construction as follows:

```
\nextslot{⟨slot number⟩}
\setslot{⟨glyph name⟩}
  ⟨slot commands⟩
\endsetslot
```

A typical example of this is

```
\nextslot{65}
\setslot{A}
  \Unicode{0041}{LATIN CAPITAL LETTER A}
\endsetslot
```

which gets typeset as

Slot 65 'A'
Unicode character U+0041, LATIN CAPITAL LETTER A.

The `\nextslot` command does not typeset anything; it simply stores the slot number in a counter, for later use by `\setslot`. The `\endsetslot` command increments this counter by one. Hence the `\nextslot` command is unnecessary between `\setslot`s for consecutive slots. Besides `\nextslot`, there is also a command `\skipslots` which increments the slot number counter by a specified amount. The argument of both `\nextslot` and `\skipslots` can be arbitrary fontinst integer expressions (see [5]). All TEX ⟨*number*⟩s that survive full expansion are valid fontinst integer expressions,

but for example `‘\~` is not, as `\~` is a macro which will break before the expression is typeset. These cases can however be fixed by preceding the TEX ⟨*number*⟩ by `\number`, as `\number‘\~` survives full expansion by expanding to `126`.

The main duty of the ⟨*slot commands*⟩ is to specify the target character (or characters) for this slot. The simplest way of doing this is to use the `\Unicode` command, which has the syntax

`\Unicode{`⟨*code point*⟩`}{`⟨*name*⟩`}`

The ⟨*code point*⟩ is the number of the character (in hexadecimal notation, usually a four-digit number) and the ⟨*name*⟩ is the name. Case is insignificant in these arguments. If a slot corresponds to a string of characters rather than to a single character, then one uses the `\charseq` command, which has the syntax

`\charseq{`⟨`\Unicode` *commands*⟩`}`

e.g.

```
\nextslot{30}
\setslot{ffi}
  \charseq{
    \Unicode{0066}{LATIN SMALL LETTER F}
    \Unicode{0066}{LATIN SMALL LETTER F}
    \Unicode{0069}{LATIN SMALL LETTER I}
  }
\endsetslot
```

Several `\Unicode` commands not in the argument of a `\charseq` instead mean that each of the listed characters is a valid interpretation of the slot.

If a character cannot be specified in terms of Unicode code points then the specification should simply be a description in text which identifies the character. Such descriptions are written using the `\comment` command

`\comment{`⟨*text*⟩`}`

It is worth noticing that the ⟨*text*⟩ is technically only an argument of `\comment` when the program processing the ETX file is ignoring `\comment` commands. This means `\verb` and similar catcode-changing commands *can* be used in the ⟨*text*⟩. The `\par` command, on the other hand, is not allowed in the ⟨*text*⟩.

The `\comment` command should also be used for any further piece of explanation of or commentary to the character used for the slot, if the exposition seems to need it. There can be any number of `\comment` commands in the ⟨*slot commands*⟩.

3.3 Ligatures

There are three classes of ligatures in the font encoding specifications: mandatory, ordinary, and odd. Mandatory ligatures must be present in any font which complies with the encoding, whereas ordinary and odd ligatures need not be. No clear distinction can be made between ordinary and odd ligatures, but a non-mandatory ligature should be categorized as ordinary if it makes sense for the majority of users, and as odd otherwise. Hence the ‘fi’ ligature is categorized as ordinary in the `T1` encoding (although it makes no sense in Turkish), whereas the ‘ij’ ligature is odd.

In the ETX format, a ligature is specified using one of the slot commands

```
\Ligature{⟨ligtype⟩}{⟨right⟩}{⟨new⟩}
\ligature{⟨ligtype⟩}{⟨right⟩}{⟨new⟩}
\oddligature{⟨note⟩}{⟨ligtype⟩}
                {⟨right⟩}{⟨new⟩}
```

The `\Ligature` command is used for mandatory ligatures, `\ligature` for ordinary ligatures, and `\oddligature` for odd ligatures. The ⟨*right*⟩ and ⟨*new*⟩ arguments are names of the glyphs being assigned to the slots involved in this ligature. The ⟨*right*⟩ specifies the right part in the slot pair being affected by the ligature, whereas the left part is the one of the `\setslot` ... `\endsetslot` construction in which the ligaturing command is placed. The ⟨*new*⟩ specifies a new slot which will be inserted by the ligaturing instruction. The ⟨*ligtype*⟩ is the actual ligaturing instruction that will be used; it must be `LIG`, `/LIG`, `/LIG>`, `LIG/`, `LIG/>`, `/LIG/`, `/LIG/>`, or `/LIG/>>`. The slashes specify retention of the left or right original character; the `>` signs specify passing over that many slots in the result without further ligature processing. ⟨*note*⟩, finally, is a piece of text which explains when the odd ligature may be appropriate. It is typeset as a footnote.

As an example of ligatures we find the following in the specification of the `T1` encoding:

```
\nextslot{33}
\setslot{exclam}
  \Unicode{0021}{EXCLAMATION MARK}
  \Ligature{LIG}{quoteleft}{exclamdown}
\endsetslot
```

It is typeset as

Slot 33 ‘exclam’
Unicode character `U+0021`, EXCLAMATION MARK.
Mandatory ligature `exclam*quoteleft` → `exclamdown`

With other ⟨*ligtype*⟩s there may be more names listed on the right hand side and possibly a ‘⌊’ symbol showing the position at which ligature processing will start afterwards.

3.4 Math font specialities

There are numerous technicalities which are special to math fonts, but only a few of them are exhibited in ETX files.[2] Most of these have to do with the TEX mechanisms that find sufficiently large characters for commands like `\left`, `\sqrt`, and `\widetilde`.

The first mechanism for this is that a character in a font can sort of say "If I'm too small, then try character ... instead". This is expressed in an ETX file using the `\nextlarger` command, which has the syntax

`\nextlarger{`⟨*glyph name*⟩`}`

The second mechanism constructs a sufficiently large character from smaller pieces; this is known as a 'varchar' or 'extensible character'. This is expressed in an ETX file using an "extensible recipe", the syntax for which is

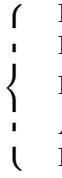
`\varchar` ⟨*varchar commands*⟩ `\endvarchar`

where each ⟨*varchar command*⟩ is one of

`\varrep{`⟨*glyph name*⟩`}`
`\vartop{`⟨*glyph name*⟩`}`
`\varmid{`⟨*glyph name*⟩`}`
`\varbot{`⟨*glyph name*⟩`}`

There can be at most one of each and their order is irrelevant. The most important is the `\varrep` command, as that is the part which is repeated until the character is sufficiently large. The `\vartop`, `\varmid`, and `\varbot` commands are used to specify some other part which should be put at the top, middle, and bottom of the extensible character respectively. Not all extensible recipes use all of these, however.

As an example, here is how a very large left brace is constructed:



Both `\nextlarger` and `\varchar` commands are like `\ligature` in that they describe ordinary features for the encoding; they appear in a specification ETX file mainly to explain the purpose of some ordinary character. There is no such thing as a mandatory `\nextlarger` or `\varchar`, but varchars are occasionally used to a similar effect. In these cases, the character generated by the extensible recipe is something quite different from what a `\char` for that slot would produce. Thus for the slot to produce the expected result it must be referenced using a `\delimiter` or `\radical` primitive, since those are the only ones which make use of the extensible recipe. The effect is that the slot has a *semimandatory* assignment; the result of `\char` is unspecified (as for a slot with an ordinary assignment), but the result for a large delimiter or radical is not (as for a slot with a mandatory assignment).

Thus some math fonts have an extra section "Semimandatory characters" between the mandatory and ordinary character sections. In that section for the `OMX` encoding we find for example

```
\nextslot{60}
\setslot{braceleftmid}
  \Unicode{2016}{DOUBLE VERTICAL LINE}
  \comment{This is the large size of the
     |\Arrowvert| delimiter, a glyphic
     variation on |\Vert|. The
     \texttt{braceleftmid} glyph
     ordinarily placed in this slot must
     not be too tall, or else the
     extensible recipe actually
     producing the character might
     sometimes not be used.}
  \varchar
    \varrep{arrowvertex}
  \endvarchar
\endsetslot
```

which is typeset as

Slot 60 'braceleftmid'
Unicode character U+2016, DOUBLE VERTICAL LINE.
This is the large size of the `\Arrowvert` delimiter, a glyphic variation on `\Vert`. The `braceleftmid` glyph ordinarily placed in this slot must not be too tall, or else the extensible recipe actually producing the character might sometimes not be used.
Extensible glyph:
Repeated `arrowvertex`

3.5 Fontdimens

Each TEX font contains a list of fontdimens, numbered from 1 and up, which are accessible via the `\fontdimen` TEX primitive. Quite a few are also used implicitly by TEX and therefore cannot be left out even if they are totally irrelevant, but as one can always include some extra fontdimens in a font — the only bounds on how many fontdimens there may be are the general bound on the size of a TFM file and the amount of font memory TEX has available — this is usually not a problem.

[2] For an overview of the subject, see for example Vieth [10].

The reason fontdimens are part of font encoding specifications is that the meaning of e.g. `\fontdimen8` varies between different fonts depending on their encoding; thus the encoding specification must define the quantity stored in each `\fontdimen` parameter. This is done using the `\setfontdimen` command, which has the syntax

`\setfontdimen{`⟨*number*⟩`}{`⟨*name*⟩`}`

The ⟨*number*⟩ is the fontdimen number (as a sequence of decimal digits where the first digit isn't zero) and the ⟨*name*⟩ is a symbolic name for the quantity.

The standard list of symbolic names for fontdimen quantities appears below; the listed quantities should always be described using the names in this list. Encoding specifications that employ other quantities as fontdimens should include definitions of these quantities. Those quantities that are defined as "Formula parameter ..." have to do with how mathematical formulae are rendered and are beyond our scope here. For exact definitions of these parameters, the reader is referred to Appendix G of *The TEXbook* [7].

acccapheight The height of accented full capitals.

ascender The height of lower case letters with ascenders.

axisheight Formula parameter σ_{22}.

baselineskip The font designer's recommendation for natural length of the TEX parameter `\baselineskip`.

bigopspacing1 Formula parameter ξ_9.

bigopspacing2 Formula parameter ξ_{10}.

bigopspacing3 Formula parameter ξ_{11}.

bigopspacing4 Formula parameter ξ_{12}.

bigopspacing5 Formula parameter ξ_{13}.

capheight The height of full capitals.

defaultrulethickness Formula parameter ξ_8.

delim1 Formula parameter σ_{20}.

delim2 Formula parameter σ_{21}.

denom1 Formula parameter σ_{11}.

denom2 Formula parameter σ_{12}.

descender The depth of lower case letters with descenders.

digitwidth The median width of the digits in the font.

extraspace The natural width of extra interword glue at the end of a sentence. TEX implicitly uses this parameter if `\spacefactor` is 2000 or more and `\xspaceskip` is zero.

interword The natural width of interword glue (spaces). TEX implicitly uses this parameter unless `\spaceskip` is nonzero.

italicslant The slant per point of the font. Unlike all other fontdimens, it is not proportional to the font size.

maxdepth The maximal depth over all slots in the font.

maxheight The maximal height over all slots in the font.

num1 Formula parameter σ_8.

num2 Formula parameter σ_9.

num3 Formula parameter σ_{10}.

quad The quad width of the font, normally approximately equal to the font size and/or the width of an 'M'. Also implicitly available as the length unit `em` and used for determining the size of the length unit `mu`.

shrinkword The (finite) shrink component of interword glue (spaces). TEX implicitly uses this parameter unless `\spaceskip` is nonzero.

stretchword The (finite) stretch component of interword glue (spaces). TEX implicitly uses this parameter unless `\spaceskip` is nonzero.

sub1 Formula parameter σ_{16}.

sub2 Formula parameter σ_{17}.

subdrop Formula parameter σ_{19}.

sup1 Formula parameter σ_{13}.

sup2 Formula parameter σ_{14}.

sup3 Formula parameter σ_{15}.

supdrop Formula parameter σ_{18}.

verticalstem The dominant width of vertical stems. This quantity is meant to be used as a measure of how "dark" the font is.

xheight The x-height (height of lower case letters without ascenders). Also implicitly available as the length unit `ex`.

3.6 The codingscheme

The final encoding-dependent piece of information in a TEX font is the codingscheme, which is essentially a string declaring what encoding the font has. This information is currently only used by programs that convert the information in a TEX font to some other format and these use it to identify the glyphs in the font. Therefore this string should be chosen so that the contents of the slots in the font can be positively identified. Observe that the encoding specification by itself does not provide enough information for this, since there are usually a couple of slots that do not contain mandatory characters. On the other hand, it is not a problem in this context if the font leaves some of the slots (even mandatory ones) empty as that is anyway easily detected. The only problem is with fonts where the slots are as-

signed to other characters than the ones specified in the encoding.

For that reason, it is appropriate to assign two codingscheme strings to each encoding. The main codingscheme is for fonts where all slots (mandatory and ordinary alike) have been assigned according to the specification or have been left empty. The variant codingscheme is for fonts where some ordinary slots have been assigned other characters than the ones listed in the specification, but where the mandatory slots are still assigned according to the specification or are left empty. The font encoding specification should give the main codingscheme name, whereas the variant codingscheme name could be formed by adding ␣VARIANT to the main codingscheme name.

Technically the codingscheme is specified by setting the `codingscheme` string variable. This has the syntax

```
\setstr{codingscheme}
{⟨codingscheme name⟩}
```

e.g.

```
\setstr{codingscheme}
{EXTENDED TEX FONT ENCODING - LATIN}
```

which is typeset as (line break is editorial)

Default s(`codingscheme`) = `EXTENDED␣TEX␣FONT␣ENCODING␣-␣LATIN`

A codingscheme name may be at most 40 characters long and may not contain parentheses. If the entire ␣VARIANT cannot be suffixed to a main name because the result becomes to long (as in the above example) then use the first 40 characters of the result.

3.7 Overall document structure

The overall structure of a font encoding specification should be roughly the following

```
\relax
\documentclass[twocolumn]{article}
\usepackage[specification]{fontdoc}
⟨preamble⟩
\begin{document}
⟨title⟩
⟨manifest⟩
\encoding
⟨body⟩
\endencoding
⟨discussion⟩
⟨change history⟩
⟨bibliography⟩
\end{document}
```

The commands described in the preceding subsections must all go in the ⟨*body*⟩ part of the document, as that is the only part of the file which actually gets processed as a data file. The part before `\encoding` is skipped and the part after `\endencoding` is never even input, so whatever appears there is only part of the LaTeX document. For the purposes of processing as a data file, the important markers in the file are the `\relax`, the `\endcoding`, and the `\endencoding` commands.

The ⟨*title*⟩ is the usual `\maketitle` (and the like) stuff. The person or persons who appear as author(s) are elsewhere in this paper described as the *encoding proposers*. The ⟨*title*⟩ should also give the date when the specification was last changed.

The ⟨*manifest*⟩ is an important, although usually pretty short, part of the specification. It is a piece of text which explains the purpose of the encoding (in particular what it can be used for) and the basic ideas (if any) which have been used in its construction. It is often best marked up as an abstract.

The ⟨*discussion*⟩ is the place for any longer comments on the encoding, such as analyses of different implementations, comparisons with other encodings, etc. This is also the place to explain any more general structures in the encoding, such as the arrow kit in the proposed `MS2` encoding [4]. In cases where the specification is mainly a formulation of what is already an established standard the ⟨*discussion*⟩ is often rather short as the relevant discussion has already been published elsewhere, but it is anyway a service to the reader to include this information. References to the original documents should always be given.

It might be convenient to include an FAQ section at the end of the discussion. This is particularly suited for explaining things where one has to look for a while and consult the references to find the relevant information.

The ⟨*change history*⟩ documents how the specification has changed over time. It is preferably detailed, as each detail in an encoding is important, but one should not be surprised if it is nevertheless rather short due to there not having been that many changes.

The ⟨*bibliography*⟩ is an important part of the specification. It should at the very least include all the sources which have been used in compiling the encoding specification, regardless of whether they are printed, available on the net, merely "personal communication", or something else. It is also a service to the reader to include in the bibliography some more general references for related matters.

The ⟨*preamble*⟩ is just a normal LaTeX preamble and there are no restrictions on defining new commands in it, although use of such commands in the ⟨*body*⟩ part is subject to the same restrictions as use of any general LaTeX command. The preamble should however *not* load any packages not part of the required suite of LaTeX packages, as that may prevent users who do not have these packages from typesetting the specification. Likewise, the specification should *not* require that some special font is available. Glyph examples for characters are usually better referenced via Unicode character charts than via special fonts.

An exception to this rule about packages is that the specification must load the fontdoc package, as shown in the outline above, since that defines the `\setslot` etc. commands that should appear in the ⟨*body*⟩. This should not cause any problems, as the fontdoc package can preferably be kept in the same directory as the collection of encoding specifications (see below). The `specification` option should be passed to the package to let it know that the file being processed is an encoding specification — otherwise `\Ligature` and `\ligature` will get the same formatting, for one. It is not actually necessary to use the article document class, and neither must it be passed the `twocolumn` option, but it is customary to do so. In principle any other document class defined in required LaTeX will do just as well.

If you absolutely think that using some non-required package significantly improves the specification, then try writing the code so that it loads the package only if it is available and provide some kind of fallback definition for sites where it is not. E.g. the url package could be loaded as

```
\IfFileExists{url.sty}{\usepackage{url}}{}
\providecommand\url{\verb}
```

The `\url` command defined by this is not equivalent to the command defined by the url package, but it can serve fairly well (with a couple of extra overfull lines as the only ill effect) if its use is somewhat restricted.

Finally, a technical restriction on the ⟨*preamble*⟩, ⟨*title*⟩, and ⟨*manifest*⟩ is that they must not contain any mismatched `\if`s (of any type) or `\fi`s, as TeX conditionals will be used for skipping those parts of the file when it is processed as a data file. If the definition of some macro includes mismatched `\if`s or `\fi`s (this will probably occur only rarely) then include some extra code so that they do match.

3.8 Encoding specification body syntax

The ⟨*body*⟩ part of an encoding specification must comply to a much stricter syntax than the rest of the file. The ⟨*body*⟩ is a sequence of ⟨*encoding command*⟩s, each of which should be one of the following:

`\setslot{`⟨*glyph name*⟩`}` ⟨*slots commands*⟩
`\endsetslot`
`\nextslot{`⟨*number*⟩`}`
`\skipslots{`⟨*number*⟩`}`
`\setfontdimen{`⟨*number*⟩`}{`⟨*name*⟩`}`
`\setstr{codingscheme}`
`{`⟨*codingscheme name*⟩`}`
`\needsfontinstversion{`⟨*version number*⟩`}`

The `\needsfontinstversion` command is usually placed immediately after the `\encoding` command. The ⟨*version number*⟩ must be at least `1.918` for many of the features described in this file to be available, and at least `1.928` if the `\charseq` command is used.

The ⟨*slot commands*⟩ are likewise a sequence of ⟨*slot command*⟩s, each of which should be one of the following:

`\Unicode{`⟨*code point*⟩`}{`⟨*name*⟩`}`
`\charseq{`⟨`\Unicode` *commands*⟩`}`
`\comment{`⟨*text*⟩`}`
`\Ligature{`⟨*ligtype*⟩`}{`⟨*right*⟩`}{`⟨*new*⟩`}`
`\ligature{`⟨*ligtype*⟩`}{`⟨*right*⟩`}{`⟨*new*⟩`}`
`\oddligature{`⟨*note*⟩`}{`⟨*ligtype*⟩`}`
`{`⟨*right*⟩`}{`⟨*new*⟩`}`
`\nextlarger{`⟨*glyph name*⟩`}`
`\varchar` ⟨*varchar commands*⟩ `\endvarchar`

where ⟨*varchar commands*⟩ is similarly a sequence of ⟨*varchar command*⟩s, each of which should be one of the following:

`\varrep{`⟨*glyph name*⟩`}`
`\vartop{`⟨*glyph name*⟩`}`
`\varmid{`⟨*glyph name*⟩`}`
`\varbot{`⟨*glyph name*⟩`}`

Finally, one can include any number of ⟨*comment command*⟩s between any two encoding, slot, or varchar commands. The comment commands are

`\begincomment` ⟨*LaTeX text*⟩ `\endcomment`
`\label{`⟨*reference label*⟩`}`

The ⟨*LaTeX text*⟩ can be pretty much any LaTeX code that can appear in conditional text. (`\begincomment` is either `\iffalse` or `\iftrue` depending on whether the encoding specification is processed as a data file or typeset as a LaTeX document respectively. `\endcomment` is always `\fi`.) The `\label` command is just the normal LaTeX `\label` command; when it is used in a ⟨*slot commands*⟩ it references that particular slot (by number and glyph name).

The full syntax of the ETX format can be found in the fontinst manual [5], but font encoding specifications only need a subset of that.

3.9 Additional fontdoc features

The `\textunicode` command is an "in comment paragraph" form of `\Unicode`. Both commands have the same syntax, but `\textunicode` is only allowed in "comment" contexts. A typical use of `\textunicode` is

```
\comment{An ...
... this is
\textunicode{2012}{FIGURE DASH}; in ...
}
```

which is typeset as

An ... this is U+2012 (FIGURE DASH); in ...

The fontdoc package inputs a configuration file `fontdoc.cfg` if that exists. This can be used to pass additional options to the package. The only currently available options that may be of interest are the `hypertex` and `pdftex` options, which hyperlinks each `U+...` generated by `\Unicode` or `\textunicode` (using HyperTEX or pdfTEX conventions[3] respectively) to a corresponding glyph image on the Unicode consortium website. To use this feature one should put the line

```
\ExecuteOptions{hypertex}
```

or

```
\ExecuteOptions{pdftex}
```

in the `fontdoc.cfg` file. *Please* do not include this option in the `\usepackage{fontdoc}` of an encoding specification file as that can be a severe annoyance for people whose TEX program or DVI viewers do not support the necessary extensions.

4 Font encoding ratification

This section describes a suggested ratification process for font encoding specifications. As there are fewer technical matters that impose restrictions on what it may look like, it is probably more subjective than the other parts of this paper.

A specification in the process of being ratified can be in one of three different stages: *draft*, *beta*, or *final*. Initially the specification is in the draft stage, during which it will be scrutinized and can be subject to major changes. A specification which is in the beta stage has received a formal approval but the encoding in question may still be subject to some minor changes if weighty arguments present themselves. Once the specification has reached the final stage, the encoding may not change at all.

[3] One could just as well do the same thing using some other convention if a suitable definition of `\FD@codepoint` is included in `fontdoc.cfg`. See the fontinst sources [6] for more details.

4.1 Getting to the draft stage

The process of taking an encoding to the draft stage can be summarized in the following steps. Being in the draft stage doesn't really say anything about whether the encoding is in any way correct or useful, except that some people (the encoding proposers) believe it is and are willing to spend some time on ratifying it.

Write an encoding specification The first step is to write a specification for the font encoding in question. This document must not only technically describe the encoding but also explain what the encoding is for and why it was created. See Subsection 3.7 for details on how the document is preferably organised.

Request an encoding name The second step is to write to the LaTeX3 project and request a LaTeX encoding name for the encoding. This mail should be in the form of a LaTeX bug report, it must be sent to

`latex-bugs@latex-project.org`,

and it must include the encoding specification file. Suggestions for an encoding name are appreciated, but not necessarily accepted. The purpose of this mail is *not* to get an approval of the encoding, but only to have a reasonable name assigned to it.

Upload the specification to CTAN The third step is make the encoding specification publicly available by uploading it to CTAN. Encoding specifications are collected in the

`info/encodings`

directory (which should also contain the most recent version of this paper). The name of the uploaded file should be '⟨*encoding name*⟩`draft.etx`'. The reason for this naming is that it must be clear that the specification has not yet been ratified.

Announce the encoding When the upload has been confirmed, it is time to announce the encoding by posting a message about it to the relevant forums. Most important is the `tex-fonts` mailing list, since that is where new encodings should be debated. Messages should also be posted to the `comp.text.tex` newsgroup and any forums related to the intended use of the encoding: an encoding for Sanskrit should be announced on Indian TEX

users forums, an encoding for printing chess positions should be announced on some chess-with-TEX user forum, etc., to the extent that such forums exist.

The full address of the `tex-fonts` mailing list is

`tex-fonts@math.utah.edu`

This list rejects postings from non-members, so you need to subscribe to it before you can post your announcement. This is done by sending a 'subscribe me' mail to

`tex-fonts-request@math.utah.edu`

The list archives can be found at

HTTP://`www.math.utah.edu/mailman/listinfo/tex-fonts`

A tip is to read through the messages from a couple of months before you write up your announcement, as that should help you get acquainted with the normal style on the list. Please do not send messages encoded in markup languages (notably, HTML, XML, and word processor formats) to the list.

Experimental encodings There is a point in going through the above procedure even for experimental encodings, i.e., encodings whose names start with an `E`. Of course there is no idea in ratifying a specification of an experimental encoding, as it is very likely to frequently change, but having a proper name assigned to the encoding and uploading its specification to CTAN makes it much simpler for other people to learn about and make references to the encoding.

4.2 From draft to beta stage

The main difference between a draft and beta stage specification respectively is that beta stage specifications have been scrutinized by other people and found to be free of errors. The practical implementation of this is that a debate is held (in the normal anarchical manner of mailing list debates) on the `tex-fonts` mailing list. In particular the following aspects of the specification should be checked:

1. *Is the encoding technically correct?* There are many factors which affect what TEX does and it is easy to overlook some. (The `\lccode`s seem to be particularly troublesome in this respect.) Sometimes fonts simply cannot work as an encoding specifies they should and it is important that such defects in the encoding are discovered on an early stage.
2. *Are there any errors in the specification?* A font encoding specification is largely a table and typos are easy to make. Proofreading may be boring, but it is very, very important.
3. *Is the specification sufficiently precise?* Are there any omissions, ambiguities, inaccuracies, or completely irrelevant material in the specification? There shouldn't be.

During the debate, the encoding proposers should hear what other people have to say about the encoding draft, revise it accordingly when some flaw is pointed out, and upload the revised version. This cycle may well have to be repeated several times before everyone is content. It is worth pointing out that in practice the debate should turn out to be more of a collective authoring of the specification than a defense of its validity. There is no point in going into it expecting the worst.

Unfortunately, it might happen that there never is a complete agreement on an encoding specification — depending on what side on takes, either the encoding proposers refuse to correct obvious flaws in it, or someone on the list insists that there is a flaw although there is obviously not — but hopefully that will never happen. If it anyway does happen then the person objecting should send a mail whose subject contains the phrase "formal protest against XXX encoding" (with XXX replaced by whatever the encoding is called) to the list. Then it will be up to the powers that be to decide on the fate of the encoding (see below).

Summarize the debate When the debate on the encoding is over — e.g. a month after anyone last posted anything new on the subject — then the encoding proposers should summarize the debate on the encoding specification draft and post this summary as a follow-up on the original mail to `latex-bugs`. This summary should list the changes that have been made to the encoding, what suggestions there were for changes which have not been included, and whether there were any formal protests against the encoding. The summary should also explain what the proposers want to have done with the encoding. In the usual case this is having it advanced to beta stage, but the proposers might alternatively at this point have reached the conclusion that the encoding wasn't such a good idea to start with and therefore withdraw it, possibly to come again later with a different proposal.

In response to this summary, the LATEX-project people may do one of three things:

- If the proposers wants the encoding specification advanced and there are no formal protests

against this, then the encoding should be advanced to the beta stage. The LaTeX-project people do this by adding the encoding to the list of approved (beta or final stage) encodings that they [presumably] maintain.

- If the proposers want to withdraw the encoding specification then the name assigned to it should once again be made available for use for other encodings.
- If the proposers want the encoding specification advanced but there is some formal protest against this, then the entire matter should be handed over to some suitable authority, as a suggestion some technical TUG committee, for resolution.

Update the specification on CTAN When the specification has reached the beta stage, its file on CTAN should be updated to say so. In particular the file name should be changed from '⟨*encoding name*⟩`draft.etx`' to '⟨*encoding name*⟩`spec.etx`'.

Modifying beta stage encodings If a beta stage encoding is modified then the revised specification should go through the above procedure of ratification again before it can replace the previous '⟨*encoding name*⟩`spec.etx`' file on CTAN. The revised version should thus initially be uploaded as ⟨*encoding name*⟩`draft.etx`, reannounced, and redebated. It can however be expected that such debates will not be as extensive as the original debates.

4.3 From beta stage to final stage

The requirements for going from beta stage to final stage are more about showing that the encoding has reached a certain maturity than about demonstrating technical merits. The main difference in usefulness between a beta stage encoding and a final stage encoding is that the latter can be considered safe for archival purposes, whereas one should have certain reservations against such use of beta stage encodings.

It seems reasonable that the following conditions should have to be fulfilled before a beta stage encoding can be made a final stage encoding:

- At least one year must have passed since the last change was made to the specification.
- At least two people other than the proposer must have succeeded in implemented the encoding in a font.

It is quite possible that some condition should be added or some of the above conditions reformulated.

References

[1] Adobe Systems Incorporated: *Adobe Standard Cyrillic Font Specification*, Adobe Technical Note #5013, 1998; HTTP://`partners.adobe.com/asn/developer/pdfs/tn/5013.Cyrillic_Font_Spec.pdf`.

[2] Adobe Systems Incorporated: *Adobe Glyph List*, text file, 1998, HTTP://`partners.adobe.com/asn/developer/type/glyphlist.txt`.

[3] Adobe Systems Incorporated: *Adobe Solutions Network: Unicode and Glyph Names*, web page, 1998, HTTP://`partners.adobe.com/asn/developer/type/unicodegn.html`.

[4] Matthias Clasen and Ulrik Vieth: *Towards a New Math Font Encoding for (LA)TEX*, March 1998, presented at EuroTEX'98; HTTP://`tug.org/twg/mfg/papers/current/mfg-euro-all.ps.gz`.

[5] Alan Jeffrey, Rowland McDonnell, Ulrik Vieth, and Lars Hellström: *fontinst — font installation software for TEX* (manual), 2004, CTAN:`fonts/utilities/fontinst/doc/fontinst.tex`.

[6] Alan Jeffrey, Sebastian Rahtz, Ulrik Vieth, and Lars Hellström: *The fontinst utility*, documented source code, v 1.9xx, CTAN:`fonts/utilities/fontinst/source/`.

[7] Donald E. Knuth, Duane Bibby (illustrations): *The TEXbook*, Addison-Wesley, 1991; volume A of *Computers and Typesetting*.

[8] Frank Mittelbach and Michel Goossens, with Johannes Braams, David Carlisle, and Chris Rowley: *The LaTeX Companion* (second edition), Addison–Wesley, 2004; ISBN 0-201-36299-6.

[9] John Plaice and Yannis Haralambous: *Draft documentation for the Omega system*, version 1.12, 1999; HTTP://`omega.cse.unsw.edu.au:8080/doc-1.12.ps`.

[10] Ulrik Vieth: *Math typesetting in TEX: The good, the bad, the ugly*, in the proceedings of EuroTEX 2001; HTTP://`www.ntg.nl/eurotex/vieth.pdf`.

◇ Lars Hellström
LaTeX3 project

ConTEXt basics for users: Font styles

Aditya Mahajan

Abstract

This article presents a summary of different ways of changing font styles in ConTEXt.

1 Introduction

The *TUGboat* editors recently invited me to write a regular column in *TUGboat* explaining some of the basic features of ConTEXt. This column is meant for ConTEXt beginners, and will explain how basic elements of ConTEXt work. I will explain it from the practicable point of view, that is, do this, and you will get this; to understand what is happening behind the scenes you need to read the ConTEXt manuals[1] and the ConTEXt sources.[2]

In this first installment, I will discuss how to use the various font styles in ConTEXt. Fonts are one of the most complicated parts of TEX. Fortunately, the macro developers take care of the dirty stuff, and most of the user interface is clean. Nevertheless, understanding the various options of the user interface can be intimidating. As a beginner, one does not want to know all the nitty-gritty details, but just the basic features. We hope to present these in this article.

In ConTEXt there are five ways to switch fonts:

1. font style (`\rm`, `\ss`, etc.),
2. font size (`\tfa`, `\tfb`, etc.),
3. alternative font style (`\bold`, `\sans`, etc.),
4. a complete font change (`\setupbodyfont`, `\switchtobodyfont`).

I will briefly explain each of these.

2 Font styles

There are three types of font families: serif, sans serif, and teletype. To switch between these families, use `\rm` for serif, `\ss` for sans serif, and `\tt` for teletype.

Each of these families come in different styles: upright, bold, italic, slanted, bold-italic, bold-slanted, and small caps. To switch to a different style, use `\tf` for upright, `\bf` for bold, `\it` for italic, `\sl` for slanted, `\bi` for bold-italic, `\bs` for bold-slanted, and `\sc` for small-capped.

You can generally combine font families and font styles, so if you want to switch to bold sans serif, you can use either `\bf\ss` or `\ss\bf`.

There is a font switch `\em` to *emphasize* text. This is somewhat special: it does automatic italic correction and changes the style depending on the current font style. For example, if the current font style is upright, `\em` switches to slanted; and if the current font style is slanted, `\em` switches to upright.

ConTEXt uses the Latin Modern fonts by default; these fonts look similar to the original Computer Modern fonts, but have a much larger character repertoire. As it happens, in the Latin Modern (and Computer Modern) fonts, the *slanted* font does not stand out from the upright font enough for some tastes; so, many people prefer to use the *italic* font for emphasis. To do that use

```
\definebodyfontenvironment[default][em=italic]
```

A font switch remains valid for the rest of the group. So, if you want to temporarily switch to a different font, use the font style command inside a group. The easiest way to start a group is to enclose the text within braces (also called curly brackets), for example

```
This is serif text
{\ss This is sans serif}
{\tt and this is typewriter}
```

which gives (notice the braces in the above lines)

This is serif text
This is sans serif
`and this is typewriter`

3 Font sizes

Occasionally one needs to change the font size. ConTEXt provides two series of commands for that. To increase the font you can use `\tfa` to scale the font size by a factor of 1.2, `\tfb` to scale by a factor of $(1.2)^2 = 1.44$, `\tfc` to scale by $(1.2)^3 = 1.728$ and `\tfd` to scale by $(1.2)^4 = 2.074$.

To decrease the font size, you can use `\tfx` to scale the font by a factor of 0.8 and `\tfxx` to scale by a factor of 0.6. The scale factors can be a function of the current font size and can be changed by `\definebodyfontenvironment`.

For example, if you want `\tfa` to be equal to 12pt when you are using 10pt font, and be equal to 14pt when you are using 11pt font, then add

```
\definebodyfontenvironment [10pt] [a=12pt]
\definebodyfontenvironment [11pt] [a=14pt]
```

The `\definebodyfontenvironment` command is de-

[1] http://pragma-ade.com/show-man-1.htm

[2] http://www.logosrl.it/context/modules/

scribed in detail in the ConTEXt manual and the `font-ini.tex` source file.

Font size can be combined with font styles. As a shortcut, you can use `\bfa` to get bold font scaled by 1.2, `\bfx` to get a bold font scaled by 0.8 and similar commands for other font styles.

These font size switches are meant for changing the font size of a few words: they do not change the interline spacing and math font sizes. So, if you want to change the font size of an entire paragraph, use `\switchtobodyfont` described below in Section 5. However, it is fine to use them as style directives in setup commands, that is, using them as an option for `style=...` in any setup command that accepts the `style` option.

4 Alternative font styles

While learning a document markup language like ConTEXt, it can be hard to remember all the commands. ConTEXt provides easy to remember alternative font styles. So for bold you can use `\bold`, for italic you can use `\italic`, for slanted you can use `\slanted`, and so on. You can probably guess what the following do:

```
\normal             \slanted
\boldslanted        \slantedbold
\bolditalic         \italicbold
\small              \smallnormal
\smallbold          \smallslanted
\smallboldslanted   \smallslantedbold
\smallbolditalic    \smallitalicbold
\sans               \sansserif
\sansbold           \smallcaps
```

In addition, the commands `\smallbodyfont` and `\bigbodyfont` can be used to change the font size.

These alternative font styles are pretty smart. You can either use them as font style switches inside a group, or as a font changing command that takes an argument. For example,

```
This is {\bold bold} and so is \bold{this}.
```

gives

This is **bold** and so is **this**.

These alternative font styles can also be used for all `style=...` options, and while using them as style options, you can just give the command name, for example:

```
\setuphead[section][style=bold]
```

5 Complete font change

If you need to change to a different font size and take care of interline spacing, you can use `\switchtobodyfont`. For example, to switch to 12pt you can use `\switchtobodyfont[12pt]`.

ConTEXt provides two relative sizes, called 'big' and 'small'. So, to go to a bigger font size, you can use `\switchtobodyfont[big]` and to go to a smaller font size, `\switchtobodyfont[small]`. The exact sizes that are used for big and small can be set using `\definebodyfontenvironment`.

The `\setupbodyfont` command accepts all the same arguments as `\switchtobodyfont`. The difference between the two is that `\setupbodyfont` also changes the font for headers, footers and other page markings, while `\switchtobodyfont` does not. So you should use `\setupbodyfont` for global font definitions to apply to the whole document, and `\switchtobodyfont` for local font changes. The effect of `\switchtobodyfont` can be localized within a group as usual.

6 Different typefaces

So far we have discussed style and size changes within a given typeface family. If you want to use a different typeface altogether, such as Times or Palatino, the Pragma web site has recipes covering all the commonly available typefaces,[3] while a separate manual describes how to write support for new typefaces.[4] (For the latter, see also Idris Hamid's article in this issue of *TUGboat*.)

The recipes as given work with the standalone ConTEXt distribution, but not with TEX Live et al.[5] To use the recipes with other distributions, try adding one of `\usetypescript[berry][ec]` or `\usetypescript[adobekb][ec]`.

7 Conclusion

There are many other ways of choosing font styles in ConTEXt. If these basic styles do not satisfy your needs, have a look at the manual, or ask on the ConTEXt mailing list.[6]

◇ Aditya Mahajan
University of Michigan
`adityam (at) umich dot edu`

[3] `http://pragma-ade.com/general/manuals/showfont.pdf`

[4] `http://pragma-ade.com/general/manuals/mfonts.pdf`

[5] `http://pragma-ade.com/general/technotes/tfmetrics.pdf` explains why ConTEXt uses separate font metrics, and gives some differences between the sets.

[6] `http://wiki.contextgarden.net/ConTeXt_Mailing_Lists`

Installing ConTEXt expert fonts: Minion Pro

Idris Samawi Hamid

Abstract

Installing fonts for ConTEXt can be an intimidating business. In this issue we take on a real monster: a collection of Adobe Minion Pro expert fonts. We hope our installation of this collection will provide an illustrative example for ConTEXt users, and help to ease the pain of installing new fonts (if you can install Minion Pro, Myriad Pro and Poetica, you can install just about anything!).

1 Introduction

Fonts can be a messy business in TEX (and, by extension, ConTEXt), and it's easy to get intimidated. One reason for this is TEX's flexibility; TEX allows you to create very sophisticated ways to take advantage of a font and to create, from one or more given font families, typeface collections tailored to your needs. Another reason is a (hopefully temporary) lack of standardization of map and encoding files between pdfTEX, dvips, and dvipdfmx. This second reason is not really a ConTEXt problem per se, though it certainly affects getting fonts working in ConTEXt.

Furthermore, ConTEXt handles fonts and font families by means of *typescripts*; these can be a bit disorienting to someone coming from LATEX and the New Font Selection Scheme (NFSS). On the other hand, after initial hesitation (having myself migrated from the LATEX world), I have concluded that the typescript approach is much more powerful and transparent than NFSS.

For a present book project, I decided to use a very complicated set of fonts from Adobe: Minion Pro (roman or serif), Myriad Pro (sans serif) and Poetica (calligraphy); all by Robert Slimbach. This set also includes a number of expert fonts with non-standard encodings. Together — and aside from mathematics — this set can provide a very nice alternative to the Computer/Latin Modern family, and one particularly suited for the humanities. These fonts also provide some of the few really excellent examples of *multiple master* (MM) technology, by Adobe. The promise of MM font technology was to provide a means of creating a series of finely optically scaled styles and alternative of a font from a single font file.[1]

Editor's Note: First published as a ConTEXt MyWay issue. Reprinted with permission.

On the other hand, despite its promise the system was never widely used and Adobe apparently no longer fully supports it.

In the present experiment we will focus on installing Minion Pro. I will not attempt to fine tune the weights; I will just use the defaults (mostly two weights per variation, plus a semibold style).[2]

There is also a Minion Pro Opticals family, which I received recently. Although this tutorial is based on the older Minion Pro familiar to advanced LATEX users, Appendix 1 explains how to set up Minion Pro Opticals. It should be easy to follow for anyone who has read the earlier sections, and provides a nice example of a truly advanced typescript.

Our work may be divided into three parts:

1. preparing the raw fonts;
2. installing the fonts; and
3. configuring typescripts and map files to use the fonts.

Ok, let's get to work!

2 Preparing the fonts

Fonts generally will come in one of three forms: Type 1 (`*.pfb`), TrueType (`*.ttf`), or OpenType (`*.otf`). TEX was generally restricted to Type 1 fonts till recently. pdfTEX supports the other two to some degree. dvipdfmx supports large Type 1 files (>256 characters per font); I don't know the status of its present or planned support for the other two.

Some fonts (like standard Type 1 fonts) contain only a standard palette of 256 character-slots. In general, such fonts do not contain expert characters or glyphs such as 'ff', 'ffi', and 'ffl'. Given a standard font, we need to combine information from at least one other corresponding font to get a complete and professional typeface for that standard font. There are three ways to prepare the raw fonts for installation. One may use:

1. the fontinst package (for Type 1 `*.pfb's);
2. FontForge (formerly PfaEdit) (for Type 1 `*.pfb's); and
3. pre-prepared fonts, with standard, expert, and variant glyphs all in one font (TrueType and, more and more, OpenType).

[1] For details, see "Designing Multiple Master Typefaces", by Adobe: `http://partners.adobe.com/public/developer/en/font/5091.Design_MM_Fonts.pdf`.

[2] We use the expressions 'style', 'variation', and 'family' in the senses employed in *ConTEXt: the Manual*, page 91. Adobe Minion Pro is a font *family* or typeface *family*, roman and sans serif are *styles*, bold and italic are *style variations*. In the ConTEXt world, the expression 'typeface' is often used to mention a user-defined collection of fonts, often drawn from various families.

If your fonts are already in a pre-prepared format, then you may just skim the first two subsections below.

2.1 fontinst

ConTEXt has its own font installation script, TEXFONT. From page 1 of the TEXFONT manual (`mtexfont.pdf`):

> The script only covers 'normal' fonts... Special fonts, like expert fonts, assume a more in depth knowledge of font handling. We may deal with them in the future. The more demanding user can of course fall back on more complicated tools like fontinst.

Although written in plain TEX, the interface to fontinst is somewhat LATEX-oriented. So its syntax largely follows the NFSS. This is no problem for ConTEXt: we only need the virtual fonts and tfm's produced by fontinst, and we ignore the `*.fd` file. Below we outline the procedure for preparing the fonts for installation using fontinst.[3]

Assuming that you are starting with 256-character Type 1 fonts, you may rename them according to the older Berry convention.[4] We don't need that convention with today's operating systems but we will use it as a starting point. This is since LATEX already has a setup for Minion Pro that uses the Berry fontname scheme and some readers may already have the raw fonts in this format.

The Minion Pro that I have contains 31 fonts. Here is a descriptive listing of the Type 1 Minion Pro family (continued lines are editorial):

```
pmnb7d.pfb      Minion Bold Oldstyle Figures
pmnb8a.pfb      Minion Bold
pmnb8x.pfb      Minion Bold Expert

pmnbi7d.pfb     Minion Bold Italic Oldstyle Figures
pmnbi8a.pfb     Minion Bold Italic
pmnbi8x.pfb     Minion Bold Italic Expert

pmnc7d.pfb      Minion Black Oldstyle Figures
pmnc8a.pfb      Minion Black
pmnc8x.pfb      Minion Black Expert

pmnr8a.pfb      Minion Regular
pmnr8x.pfb      Minion Regular Expert
pmnrc8a.pfb     Minion Regular Small Caps
                & Oldstyle Figures

pmnrd8a.pfb     Minion Regular Display
pmnrd8x.pfb     Minion Regular Display Expert
pmnrdc8a.pfb    Minion Regular Display Small Caps
                & Oldstyle Figures

pmnrdi8a.pfb    Minion Italic Display
pmnrdi8x.pfb    Minion Italic Display Expert
pmnrdic8a.pfb   Minion Italic Display Small Caps
                & Oldstyle Figures
pmnrdiw8a.pfb   Minion Italic Display Swash

pmnri8a.pfb     Minion Italic
pmnri8x.pfb     Minion Italic Expert
pmnric8a.pfb    Minion Italic Small Caps
                & Oldstyle Figures
pmnriw8a.pfb    Minion Italic Swash

pmnrp8a.pfb     Minion Ornaments

pmns8a.pfb      Minion Semibold
pmns8x.pfb      Minion Semibold Expert
pmnsc8a.pfb     Minion Semibold Small Caps
                & Oldstyle Figures

pmnsi8a.pfb     Minion Semibold Italic
pmnsi8x.pfb     Minion Semibold Italic Expert
pmnsic8a.pfb    Minion Semibold Italic Small Caps
                & Oldstyle Figures
pmnsiw8a.pfb    Minion Semibold Italic Swash
```

Let us begin our analysis of Minion; we need to make a few decisions. We'll just make a note of them for later; it helps to stay organized with all the accounting involved in the typescripts:

We first note that, aside from the ornamental font, there are 5 main style variations: medium, semibold, bold, black, and italic. Medium has a display version, italic has a display version, bold has an italic version, and semibold has an italic version, for a total of nine variations. We need to make some sense of this in terms of optical scaling. For our future typescript, we will initially group some of these as follows:

- For `\tf`, let's try medium for sizes < 17.3pt, and medium display for sizes ≥ 17.3pt;
- For `\bf`, try bold for sizes ≥ 8pt, and black for sizes ≤ 8pt (there is no display size for bold). Similarly for `\bi`;
- For `\it`, we try italic for sizes < 17.3pt, and italic display for sizes ≥ 17pt;[5]
- We will leave semibold as its own alternative, although I did once try treating semi-

[3] For a wealth of information about fontinst and virtual fonts, see Alan Hoenig's book *TEX Unbound*. A more recent and up-to-date manual is *The Font Installation Guide*, by Philipp Lehman. It is available in `CTAN:/info/type1fonts/fontinstallationguide`.

[4] For details, see Hoenig, pages 132–134, and Lehman, pages 11–13.

[5] This is all intentionally experimental. Lehman, page 63, has more professional suggestions, but I think it's important to reflect for ourselves. Probably you will one day have to install a font where no one has made predeterminations about this sort of thing.

bold (\sb) as an option for small or caption-sizes ($\leq$ 8pt). I think this was a failure, but the reader should try it and judge for himself.

The rest of our choices will be analogous.

We also note the following, based on a direct examination of these fonts:

- Based on the above grouping, small caps will be available in both weights for \tf and for \it, but not for \bf (sigh) or \bi. Oddly, semibold has both a small caps variation and a small caps italic variation. According to Lehman (page 63), semibold is the actual default bold weight; maybe he's right.[6]
- For a given optical size (as tentatively defined above), old style figures are available in both the expert font and in the old style figures font;
- For some reason, Minion Bold Oldstyle Figures as well as Minion Bold Italic Oldstyle Figures have no small caps; each are identical to Minion Bold and Minion Bold Italic respectively, except for the numerals. The other styles use small caps in their old style figures versions;
- It is our intention to make old style numerals the default for our entire typeface collection; this makes sense in the humanities, in my view;[7]
- For all five primary variations (\tf, \it, \bf, \bi, and \sb) and their derivatives, we will try
  - using the expert fonts for both old figures and expert ligatures;
  - using the old style figures fonts for small caps only (\tf and \it);

  Although we could just default to old style figures for \bf and \bi, for purposes of consistency we will, for the time being, treat all four typefaces equally in this regard. You can always change this...
- The most difficult task to accomplish the above is dealing with the expert fonts in this collection. They share a non-standard encoding vector. We need to make our typeface collection default to the expert ligatures and to the old style numerals.

Preparing the raw fonts for installation involves making a fontinst file makeminion.tex like the following:

```
\input fontinst.sty

\installfamily{T1}{pmn}{}
\installfonts

% minionr
\installfont{minionr10}   {pmnr8a,pmnr8x,latin}
            {T1j}{T1}{minion}{m}{n}{}
\installfont{minionr17}   {pmnrd8a,pmnrd8x,latin}
            {T1j}{T1}{minion}{m}{n}{}

% minioni
\installfont{minioni10}   {pmnri8a,pmnri8x,latin}
            {T1j}{T1}{minion}{m}{it}{}
\installfont{minioni17}   {pmnrdi8a,pmnrdi8x,latin}
            {T1j}{T1}{minion}{m}{it}{}

% minionb
\installfont{minionb10}   {pmnb8a,pmnb8x,latin}
            {T1j}{T1}{minion}{b}{n}{}
\installfont{minionbl10}  {pmnc8a,pmnc8x,latin}
            {T1j}{T1}{minion}{b}{n}{}

% minionbi
\installfont{minionbi10}  {pmnbi8a,pmnbi8x,latin}
            {T1j}{T1}{minion}{m}{bi}{}

% minionsc
\installfont{minionsc10}  {pmnrc8a,latin}
            {T1j}{T1}{minion}{m}{sc}{}
\installfont{minionsc17}  {pmnrdc8a,latin}
            {T1j}{T1}{minion}{m}{sc}{}

% minionisci
\installfont{minionsci10} {pmnric8a,latin}
            {T1j}{T1}{minion}{m}{sc}{}
\installfont{minionsci17} {pmnrdic8a,latin}
            {T1j}{T1}{minion}{m}{sc}{}

% minionsb
\installfont{minionsb10}  {pmns8a,pmns8x,latin}
            {T1j}{T1}{minion}{sb}{n}{}

% minionsbi
\installfont{minionsbi10}  {pmnsi8a,pmnsi8x,latin}
            {T1j}{T1}{minion}{sb}{it}{}

% minionsbsc
\installfont{minionsc8}   {pmnsc8a,latin}
            {T1j}{T1}{minion}{sb}{sc}{}
\installfont{minionsci8}  {pmnsic8a,latin}
            {T1j}{T1}{minion}{sb}{sc}{}

% minionisw
%\installfont{minionswi10}  {pmnriw8a,latin}
%            {T1j}{T1}{minion}{m}{it}{}
%\installfont{minionswi17}  {pmnrdiw8a,latin}
%            {T1j}{T1}{minion}{m}{it}{}
```

[6] On the other hand, Minion Pro Opticals has small caps for bold, and the official documentation seems to indicate that the default bold is, indeed, Minion Bold.

[7] In *The Elements of Typographic Style*, Bringhurst enjoins:

> *Use titling [upright] figures with full caps, and text [old style] figures in all other circumstances.*

```
%\installfont{minionsbswi10}{pmnsiw8a,latin}
%            {T1j}{T1}{minion}{sb}{it}{}

% miniono
%\installfont{miniono10}   {pmnrp8a,latin}
%            {T1j}{T1}{minion}{m}{n}{}
\endinstallfonts
\bye
```

Let us look briefly at the first `\installfont` line (see Hoenig or Lehman for details):

- `\installfont {minionr10}`
  The name of our virtual font is `minionr10`;
- `{pmnr8a,pmnr8x,latin}`
  Our standard font is `pmnr8a`, expert font is `pmnr8x`, and `latin.mtx` is the default fontinst metric file that defines at least 401 glyphs found in Latin alphabets (see Hoenig, page 180);
- `{T1j}{T1}{minion}{m}{n}{}`
  The *encoding file* is `t1j.etx` (Cork with oldstyle numerals), *general encoding* is T1 (cork), *family* is minion, *series* is medium, *shape* is normal, and size is left empty. This is all NFSS terminology.

We have intentionally organized `makeminion.tex` to be analogous to our future typescript file. Also, we have commented out the swash and ornament lines. This is because I personally prefer to deal with the preparation, installation, and configuration of each of these two in its own directory, separate from the main fonts. So in the `/swash` subdirectory the file `makeminionsw.tex` will contain only the swash lines, and `/ornaments` will contain only the ornament line. Looking ahead, we will have three separate typescript classes: main, swash, and ornament. Because writing advanced typescripts requires a lot of careful accounting, it is better to keep these classes separate. If you don't believe me, try doing everything that follows in the configuration stage in a single typescript. You'll see!

In the LaTeX version, the final fonts are given names like `pmnr9e` instead of `minionr10`. Since we don't have to deal with NFSS and old encodings, we can happily dispense with that here.

In retrospect, I prefer to avoid fontinst. There is a very limited number of pre-made `*.etx` files, though you can make your own. But if you have a set of 256-character-slot Type 1 fonts, the next method will make our life a bit easier later, as you'll see. On the other hand, if you really need dvips, then you may need to go the fontinst route (dvipdfmx works fine with the next method).

2.2 FontForge

While I was working with fontinst, the thought occurred to me: is there some way we can merge the expert and standard fonts into a single font file, so we can just use TeXfont (you will soon see why this makes things easier)? I tried FontLab: no such feature. Fontographer? Foiled again. Then I looked at the open source FontForge (formerly PfaEdit).[8] For Windows users, there is a version for Cygwin. It's definitely worth installing a minimal Cygwin to have FontForge; instructions are on the FontForge site.

In FontForge, open `pmnr8a.pfb`. Then go to the menu `ELEMENT => MERGE` to choose the corresponding expert font, `pmnr8x.pfb`. FontForge will add every character with a different name to the original glyph palette. Then save this new font to `minionr10.pfb`. You must repeat this for all standard fonts that have an expert companion. You can use `makeminion.tex` in the above subsection on fontinst to identify the correspondences and correct names. For those fonts that have no expert companion, just copy and rename them to our scheme.

A nice thing about FontForge is that it is scriptable. So those who are familiar with that can write a script so that FontForge can do all of this in batch. That skill is a bit beyond me, so I just did it the point-and-click way. Look up "scripting" in the FontForge documentation.

2.3 Pre-prepared fonts

If you have OpenType or TrueType versions of the fonts you are set. If you need to use Aleph (א, which cannot use `*.ttf/*.otf files`), or need dvipdfmx, then you need to convert each font to Type 1.[9] Don't worry about the 256-character-slot limit for Type 1 fonts; it won't affect things for us. To follow along easily, save copies of your OpenType Minion Pro fonts to the names we are using here.

3 Installing the fonts

First, we must have the `afm` files for all the raw fonts. You can generate them with any decent font-editing software. There is an afm-generation utility, `getafm`, that comes with TeX Live, but it does not procure the proper kerning info. A package of metrics for the *Adobe Type Classics for Learn-*

[8] Available here: `http://fontforge.sourceforge.net/` .

[9] One may use FontForge for this. There is also a tool `cfftot1` provided by LCDF (`http://www.lcdf.org/type/`) but it can not, as far as I can tell, generate an `*.afm` file. But see Appendix 1.

ing suite (including Minion Pro) is available from `http://www.lcdf.org/type/`.

3.1 fontinst

A long batch file can handle most of the work. (Available from the author or in the ConTEXt My-Way version of this article.) Once you have generated the files and directories, you can install them in your local tree or in `/texmf-fonts`, which ConTEXt uses. You will also need a proper map file, which is where I made my big mistake with fontinst. For dvips I used lines like this:

```
pmnr8a pmnr8a <pmnr8a
```

But I discovered that I needed lines like this (line break is editorial):

```
pmnr8a Minion-Regular
 "TeXBase1Encoding ReEncodeFont " <8r.enc <pmnr8a.pfb
```

Walter Schmidt has provided a complete LATEX package for Minion: `http://mirror.ctan.org/fonts/psfonts/w-a-schmidt/pmn.zip`. For details see this package. Between this and Tutorial VI of Lehman's *Guide* you will learn all you need to know about installing Minion Pro, as well as a lot about fontinst. In any case, I much prefer using TEXFONT. Our installation in TEXFONT will involve multiple encodings. To do this in fontinst you may have to write your own `*.etx` files, etc, endure a lot of debugging, and so forth. Make your life easy and get FontForge.

3.2 TEXfont: Type 1, TrueType or OpenType

Let us begin by making three temporary directories:

- `/main`: Put all Minion `pfb`'s and `afm`'s here;
- `/swash`: Put the swash fonts `minionswi10.pfb`, `minionsbswi10.pfb`, and `minionswi17.pfb` here;
- `/ornament`: Put the ornament font `miniono10.pfb` and metric file here.

Here we set our encoding vectors for Minion Pro. We will use the texnansi encoding as our base, though you can easily choose another (like ec) if you like. Actually, we will use the file `texnansi-lm.enc`, in `/texmf-local/fonts/enc/dvips/lm`, as our base file, because it is easier to edit than `texnansi.enc`. Just make sure to remove all `*.dup` extensions. For example, change `/OE.dup` to `/OE`.

Now we create a few encoding files (all go into `/main` except the last two). From careful study these examples, you can easily make your own special encodings at will. NOTE: each encoding file must have precisely 256 character lines, not counting the beginning line and the ending line:

- `texnansi-axo.enc`
  The prefix `texnansi` is important; it tells us that texnansi encoding is our foundation. The string '`ax`' stands for 'Adobe Expert' and the '`o`' stands for 'old style numerals'. This encoding file will be used to create and install a virtual font that defaults to old style numerals. Simply replace the lines

  ```
  /zero
  /one
  ...
  /nine
  ```

  with

  ```
  /zerooldstyle       %/zero etc.
  /oneoldstyle
  ...
  /nineoldstyle
  ```

  The comments just remind us of the original characters we are replacing. Change the beginning line to `/enctexnansiaxo[`. There are a couple of minor quirks to keep in mind. The batch file mentioned earlier has the full `texnansi-axo.enc`; changes from the original `texnansi.enc` are noted. For example, there is no dotless 'j' in either the Adobe standard or expert encodings, at least not with Minion Pro.[10]
- `texnansi-axu.enc`
  This encoding file will be used to create and install a virtual font that defaults to upright numerals. Use the default numeral characters from `texnansi.enc`.
- `texnansi-axs.enc`
  This encoding file will be used to create and install a virtual font that defaults to superior numerals. Use `/zerosuperior`, etc.
- `texnansi-axi.enc`
  This encoding file will be used to create and install a virtual font that defaults to inferior numerals. Use `/zeroinferior`, etc.
- `texnansi-axuc.enc`
  This encoding file will be used to create and install a virtual font that defaults to upright numerals and small caps. The small caps

[10] There is a free tool, `t1dotlessj`, that creates a dotless-'j' Type 1 font from an existing standard font: `http://www.lcdf.org/type/t1dotlessj.1.html` . You may then use FontForge to merge this with your main font (preferable), or go through fontinst.

fonts that come with Minion Pro all default to old style numerals, and these numerals are encoded with the upright character names. Take `texnansi-axu.enc` and replace

```
/a
/b
...
```

with

```
/Asmall
/Bsmall
...
```

and so forth. Using this particular encoding is only good for those standard fonts with small caps in the corresponding expert font. For example, Minion Bold Expert has no small caps (although Semibold Expert does). Basically you will be replacing all of the original small caps fonts with non-small caps big fonts encoded with small caps glyphs. We will say more about this below, in the section on typescripting.

- `texnansi-ao.enc` and `texnansi-aw.enc`
  We make encodings for the swashes and ornaments. The ornaments take up 23 slots corresponding to A through W; the swashes take up A through Z. The `/space` slot is the only other one kept in place; fill up the rest with `/.notdef`'s, e.g.,

```
/.notdef
/.notdef
/.notdef
/ornament1      %/A,
/ornament2      %/B
/ornament3      %/C, etc
```

for ornaments, and

```
/.notdef
/.notdef
/.notdef
/A
/B
/C
/D              % etc.
```

for swashes.

There are lots of other possibilities, like an encoding that uses text-fractions and so forth. You are now in control!

It is now time to install. TEXFONT will do most of the work, but you have to install the encoding files by hand. It would be nice if TEXFONT could do this for us as well. In the meantime, copy the encoding files to `/texmf-fonts/fonts/enc/dvips/minion`. *Do not* forget to install the encoding files!

Now we are ready to install our main fonts with TEXFONT. The directory `/main` should have the `*.pfb` files, the `*.afm` files, and the encoding files (or you can pre-install the encoding files and run `texhash`). From each of the three respective directories, issue the corresponding commands from the following:

```
texfont --ma --in --en=texnansi-axo --ve=adobe \
        --co=minion --show
texfont --ma --in --en=texnansi-axu --ve=adobe \
        --co=minion --show
texfont --ma --in --en=texnansi-axs --ve=adobe \
        --co=minion --show
texfont --ma --in --en=texnansi-axi --ve=adobe \
        --co=minion --show

% texfont --ma --in --en=texnansi-axuc --ve=adobe \
%         --co=minion --show
% uncomment for small caps with upright numerals.
```

Do `texfont --help` to see the meaning of each of the above switches. The above commands use abbreviated versions (first two letters) of these options.[11]

Now you will find four `pdf` files in `/main`:

- `texnansi-axo-adobe-minion.pdf`,
- `texnansi-axu-adobe-minion.pdf`,
- `texnansi-axs-adobe-minion.pdf`, and
- `texnansi-axi-adobe-minion.pdf`.

Take a look at these; they include beautiful font charts of your encodings. Also visit the map files in `/texmf-fonts/map/pdftex/context`. Peruse especially the way the virtual fonts and `tfm` files fonts are named. It's verbose but very easy to read and systematic.

Did you remember to install the encoding files?

4 Configuration

4.1 The texnansi-axo typeface collection

Now we need a set of typescripts that can handle our main Minion font collection: let's call them `type-mino.tex`, `type-minu.tex`, `type-mins.tex`, and `type-mini.tex`. The four are almost identical so we will choose one of them to analyze in detail, namely `type-mino`. Each typescript will have

[11] Adam Lindsay pointed out to me that you may also use the `--variant` option (e.g., `--va=texnansi-axo` instead of `--en`. Then there may be no need to install the encoding files: just use `[encoding=texnansi]` in the typescripts.

five main parts: *font mapping*, *general names*, *font sizes*, *map loading*, and *final typefaces*. Let us deal with each of these in turn.

4.1.1 Font mapping

We map the raw fonts to easy-to-understand names. We are not mapping directly to the `pfb`'s, but rather to the virtual fonts.

```
% We need a few switches: I don't guarantee
% they don't conflict with other commands;-)
\definestyle [italicsmallcaps,smallcapsitalic]
             [\si] []
\definestyle [black] [\bk] []
\definestyle [semiboldroman,semibold] [\sb] []
\definestyle [semibolditalic] [\st] []
\definestyle [semiboldsmallcaps] [\sp] []
\definestyle [semiboldsmallcapsitalic] [\stp] []

% Regular serifs, > 8pt, < 17.3pt
\starttypescript[serif] [miniono] [texnansi-axo]

\definefontsynonym [Minion10]
                   [texnansi-axo-minionr10]
\definefontsynonym [Minion17]
                   [texnansi-axo-minionr17]

\definefontsynonym [MinionItalic10]
                   [texnansi-axo-minioni10]
\definefontsynonym [MinionItalic17]
                   [texnansi-axo-minioni17]

\definefontsynonym [MinionBold10]
                   [texnansi-axo-minionb10]
\definefontsynonym [MinionBlack]
                   [texnansi-axo-minionbl10]

\definefontsynonym [MinionBoldItalic]
                   [texnansi-axo-minionbi10]

\definefontsynonym [MinionCaps10]
                   [texnansi-axu-minionsc10]
\definefontsynonym [MinionCaps17]
                   [texnansi-axu-minionsc17]

\definefontsynonym [MinionItalicCaps10]
                   [texnansi-axu-minionsci10]
\definefontsynonym [MinionItalicCaps17]
                   [texnansi-axu-minionsci17]

\definefontsynonym [MinionSemiBold]
                   [texnansi-axo-minionsb10]

\definefontsynonym [MinionSemiBoldItalic]
                   [texnansi-axo-minionsbi10]

\definefontsynonym [MinionSemiBoldCaps]
                   [texnansi-axu-minionsbsc10]

\definefontsynonym [MinionSemiBoldItalicCaps]
                   [texnansi-axu-minionsbsci10]
\stoptypescript
```

Note that we map to the upright-encoded fonts for the six fonts with small caps. This is because the small caps fonts each default to old style numerals, but those numerals are encoded in the font with upright names. Furthermore, the small caps fonts do not have corresponding experts. So the small caps virtual fonts in `texnansi-axo` encoding have no numerals at all.

On the other hand, the `texnansi-axu` encoded small caps virtual fonts will display old style numerals because those numerals are encoded in the font with upright names. The rest will display upright numerals. This inconsistency is wholly due to the manufacturer of the original raw fonts.

Similarly, the typescript for `texnansi-axu` encoded fonts will need to map small caps to the `texnansi-axuc` encoded fonts, if full consistency is desired. The `texnansi-axuc` encoded fonts do not need their own typescript, since they are meant to supplement texnansi-axu.[12]

Note the option `[miniono]`. For `type-minu.tex` it should be `[minionu]` (with a 'u') and so forth.

4.1.2 General names

This part may seem redundant right now, but it will make sense when we add the Myriad Pro collection. That is a sans serif, while Minion is a serif, so this helps keep things clear and organized.

```
\starttypescript[serif] [miniono] [name]

\definefontsynonym [Serif]  [Minion10]
\definefontsynonym [Serif17] [Minion17]

\definefontsynonym [SerifItalic10] [MinionItalic10]
\definefontsynonym [SerifItalic17] [MinionItalic17]

\definefontsynonym [SerifBold10] [MinionBold10]
\definefontsynonym [SerifBlack] [MinionBlack]

\definefontsynonym [SerifBoldItalic]
                   [MinionBoldItalic]
```

[12] In `type-minu.tex`, replace the raw font names in these lines:

```
\definefontsynonym [MinionCaps10]
                   [texnansi-axu-minionsc10]
...
\definefontsynonym [MinionSemiBoldItalicCaps]
                   [texnansi-axu-minionsbsci10]
```

with the corresponding names from the `texnansi-axuc`:

```
\definefontsynonym [MinionCaps10]
                   [texnansi-axuc-minionr10]
...
\definefontsynonym [MinionSemiBoldItalicCaps]
                   [texnansi-axuc-minionsbi10]
```

```
\definefontsynonym [SerifCaps10] [MinionCaps10]
\definefontsynonym [SerifCaps17] [MinionCaps17]

\definefontsynonym [SerifItalicCaps10]
                   [MinionItalicCaps10]
\definefontsynonym [SerifItalicCaps17]
                   [MinionItalicCaps17]

\definefontsynonym [SerifSemiBold] [MinionSemiBold]

\definefontsynonym [SerifSemiBoldItalic]
                   [MinionSemiBoldItalic]

\definefontsynonym [SerifSemiBoldCaps]
                   [MinionSemiBoldCaps]

\definefontsynonym [SerifSemiBoldItalicCaps]
                   [MinionSemiBoldItalicCaps]
\stoptypescript
```

4.1.3 Font sizes

This is where we implement optical scaling (what little there is, anyway). If you have Minion Pro Opticals, you will have more choices. The following typescript will give you the needed insight to implement your own scheme for optical scaling.

Note that in the first line of this section of our typescript, we have mapped `Minion10`, not to `Serif10`, but to just `Serif`. ConTEXt treats the `Serif` font as the default or empty font; if it is not defined, in a few cases ConTEXt will fall back to a typeface where it is defined (generally Latin Modern).[13]

```
\starttypescript [serif] [miniono] [size]

\definebodyfont [9pt,10pt,11pt,12pt,14.4pt]
  [rm]
  [tf=Serif sa 1,
   sc=SerifCaps10 sa 1,
   it=SerifItalic10 sa 1,
   si=SerifItalicCaps10 sa 1]

\definebodyfont [4pt,5pt,6pt,7pt,8pt]
  [rm]
  [tf=Serif sa 1,
   sc=SerifCaps10 sa 1,
   it=SerifItalic10 sa 1,
   si=SerifItalicCaps10 sa 1,
   bf=SerifBlack sa 1]

\definebodyfont [17.3pt,20.7pt,24.9pt]
  [rm]
  [tf=Serif17 sa 1,
   sc=SerifCaps17 sa 1,
   it=SerifItalic17 sa 1,
   si=SerifItalicCaps17 sa 1]

\definebodyfont [9pt,10pt,11pt,12pt,14.4pt,
                 17.3pt,20.7pt,24.9pt]
  [rm]
  [bf=SerifBold10 sa 1]

\definebodyfont
  [24.9pt,20.7pt,17.3pt,14.4pt,12pt,11pt,10pt,
   9pt,8pt,7pt,6pt,5pt,4pt]
  [rm]
  [bi=SerifBoldItalic sa 1,
   sb=SerifSemiBold sa 1,
   st=SerifSemiBoldItalic sa 1,
   sp=SerifSemiBoldCaps sa 1,
   stp=SerifSemiBoldItalicCaps sa 1]
\stoptypescript
```

Some of these switches are newly defined (like `\sp`), and the ConTEXt mechanism for enlarging and reducing the size of a given style variation will not work. We need to define them for completeness. See pages 129–131 of *ConTEXt: the Manual* for details. It's really quite straightforward, just a bit tedious and verbose, so we leave it as an exercise for the reader.[14] See also the typescript in Appendix 1.

Choosing optical sizes is an area that needs experimentation to get right. For example, does the black font really work at small bold sizes?

4.1.4 Map loading

Here we load our map files, created during installation.

```
\starttypescript[map] [miniono] [texnansi-axo]
\loadmapfile[texnansi-axo-adobe-minion.map]
\loadmapfile[texnansi-axu-adobe-minion.map]
\stoptypescript
```

We also use the `texnansi-axu` map for small caps as discussed above. The `type-minu.tex` file will also need to load the `texnansi-axuc` map file, if you have installed and desire to have small caps with upright numerals.[15]

[13] My thanks to Adam Lindsay for pointing this out.

[14] Here is one example to get you started:

```
\definebodyfont
 [24.9pt,20.7pt,17.3pt,14.4pt,12pt,
  11pt,10pt,9pt,8pt,7pt,6pt,5pt,4pt]
 [rm]
 [sp=SerifSemiBoldCaps sa 1,
  spa=SerifSemiBoldCaps scaled \magstep1, % or sa a
  spb=SerifSemiBoldCaps scaled \magstep2, % or sa b
  spc=SerifSemiBoldCaps scaled \magstep3, % or sa c
  spd=SerifSemiBoldCaps scaled \magstep4] % or sa d
  ...
```

[15] That is, in `type-minu.tex` you will need to declare something like this:

```
\starttypescript[map] [minionu] [texnansi-axu]
\loadmapfile[texnansi-axu-adobe-minion.map]
\loadmapfile[texnansi-axuc-adobe-minion.map]
\stoptypescript
```

4.1.5 Final typefaces

This is where we put it all together, our Minion typeface collection. We also define those fonts that do not come with Minion, Myriad, or Poetica, such as math fonts (we use Euler) and monospaced (we use Latin Modern). Note the 'o' suffix in what follows. Such identifying suffixes will be needed in the other typescript files as well as well.

While very powerful and transparent, typescripts are quite sensitive to these kinds of seemingly minor accounting issues, so be careful.

```
\starttypescript[ADOBEMiniono]

\definebodyfontenvironment
  [adobeminiono]
  [default]
  [interlinespace=2.6ex]

\definetypeface [adobeminiono]
[rm] [serif] [miniono] [miniono]
[encoding=texnansi-axo]

% Configure Myriad and Poetica later, then uncomment
%\definetypeface [adobeminiono]
%[ss] [sans] [myriado] [myriado]
%[encoding=texnansi]

%\definetypeface [adobeminiono]
%[cg] [calligraphy] [poetica] [poetica]
%[encoding=texnansi]

\definetypeface [adobeminiono]
[mm] [math] [euler] [default]
[encoding=texnansi,rscale=0.89]

\definetypeface [adobeminiono]
[tt] [mono] [modern] [default]
[encoding=texnansi,rscale=0.99]

\stoptypescript
```

We found that the non-Minion fonts used in our typeface collection, such as Euler math fonts and Latin Modern monospaced, need to be scaled. That is what the `rscale=<scale factor>` option does for us. We also note that Minion needs a smaller interline space factor than the usual 2.8`ex`. We may need to do some more testing in this regard, though 2.6`ex` seems to work well for the Minion design.

Be aware that ConTEXt sets up `\em` with the slanted (`\sl`) style variation by default. But Minion Pro does not come with a slanted font. So `\em` will not work unless you map one of your fonts — see the previous section on size definitions — to `\sl`. Declare

```
\setupbodyfontenvironment[default][em=italic]
```

either in your typescript or in your style or environment file. It may not be such a good idea to define it in the typescript, because you could get odd results depending on the order your typescripts are scanned during compilation (assuming you've set up `\em` differently somewhere else).

Now write the above set of typescripts to a file, `type-mino.tex`. We can now test our typescript so far. Here is a test file:

```
% output=pdf interface=en

\usetypescriptfile[type-mino]
\usetypescript[ADOBEMiniono]
\switchtotypeface[adobeminiono]%

\starttext
This is a test of Minion in \CONTEXTT. 1234\par
\bf This is a test of Minion in \CONTEXTT. 1234\par
\it This is a test of Minion in \CONTEXTT. 1234\par
\bi This is a test of Minion in \CONTEXTT. 1234\par
\sc This is a test of Minion in \CONTEXTT. 1234\par
\si This is a test of Minion in \CONTEXTT. 1234\par
\sb This is a test of Minion in \CONTEXTT. 1234\par
\stp This is a test of Minion in \CONTEXTT. 1234\par

\switchtotypeface[adobeminionor]
{\tf ABCDEFGHIJKLMNOPQRSTUVW \par}\blank

\switchtotypeface[adobeminionsw]
{\sw ABCDEFGHIJKLMNOPQRSTUVW \par}\blank
\stoptext
```

5 Post-dvi processing

Unfortunately, inconsistencies between pdfTEX, dvips, and dvipdfmx mean we have to do more work if we need post-`dvi` processing for any reason (this is the case with ℵ, for example).

5.1 dvipdfmx

You will need to write at least one map file for your collection. Look at, e.g., `texnansi-axo-adobeminion.map`. Change the syntax from (line break is editorial)

```
texnansi-axo-raw-minionr10 Minion-Regular 4
  <minionr10.pfb texnansi-axo.enc
```

to

```
texnansi-axo-raw-minionr10 texnansi-axo minionr10
```

Now, to make your map file available to dvipdfmx, you can add a line like

```
f minion-dvipdf.map
```

to the file `/texmf-local/fonts/dvipdfm/config/config`, or you can invoke your map file from the command line:

```
dvipdfmx -f minion-dvipdf.map
```

5.2 dvips

If you need to use dvips, you may have to go the

fontinst route. See Tutorial VII of Lehman's *Guide* for a very thorough discussion of preparing map files for dvips with fontinst.

Best wishes for painless font installation in ConTEXt!

6 Acknowledgements

I would like to especially thank Hans Hagen, Adam Lindsay, Thomas A. Schmitz, Ralf Stubner, and others from the ConTEXt mailing list for their help and assistance during the struggle to prepare this article.

Appendix 1 Minion Pro Opticals

As I was finishing this article I received the complete Minion Pro Opticals set, in OpenType format. This set is more internally coherent than the older version we used for this tutorial. It contains six style variations: medium or regular, semibold, bold, italic, semibold italic, and bold italic. The black style variation is apparently gone. Each style variation comes in four optical sizes: normal, caption, subhead, and display. Each font has a standard 256-character encoding, plus a set of old style numerals, superiors, inferiors, a set of small caps (we have bold small caps now!), ornaments and hundreds of other alternates. There is no dedicated small caps or ornaments font. Each italic font has a large palette of swashes, many more than the original swash fonts. There are also Greek, Cyrillic, and lots of alternate or fancy ligatures. This is really much better than the original set.

Our encodings prepared earlier will suffice with a few changes (and you can always make your own):

- `texnansi-axo.enc`, `texnansi-axu.enc`, `texnansi-axs.enc`, and `texnansi-axi.enc` will stay the same;
- For small caps we need `texnansi-axoc.enc` for small caps with old style numerals. Just modify `texnansi-axuc.enc` and replace the default numerals with the old style ones;
- `texnansi-aw.enc` will have to change `/Aswash` to `/A.swash`, etc. The italics font also offer lots of swash capitals with accents. Some of these swashes do not have a corresponding entry in the standard encoding: for example, there is no `/Ebreve` in the standard encoding to match `/Ebreve.swash`. So if you want the esoteric swashes you will have to pick and choose how you want to encode this within a 256-character context.

  One idea about swashes: since they are now part of the full italic fonts, treat them like small caps, and encode them in the same `/a--/z` band of the encoding. This was much harder to accomplish in the old fonts.
- The ornaments' names in `texnansi-ao.enc` have to be changed: `/ornament1` becomes `/orn.001`, etc., up to `/orn.023`. An identical set of ornaments is present in every font, so you can also encode ornaments like a small caps font if you like.

Installation is just as before. Convert fonts to `*.pfb` (with `*.afm`), place in a separate directory with your encodings, then run TEXFONT for as many encodings as you like.[16]

The typescript files are mostly as before: the only really interesting difference is the much better optical scaling. According to the Minion Pro Opticals documentation, the intended optical scaling spectrum is as follows:

- Caption: 6–8.4 point
- Normal (Regular): 8.5–13 point
- Subhead: 13.1–19.9 point
- Display: 20+ point

For small caps, one may choose to write a separate typescript file and typeface collection, in which case one has to switch fonts to use small caps. Or one can integrate, e.g., the small caps fonts into the upright numerals typescript file. Experiment to get the combination that works best for you.

Enjoy!

◇ Idris Samawi Hamid
Colorado State University
`ishamid (at) colostate dot edu`

[16] For an alternative approach, see Adam Lindsay's "OpenType installation basics for ConTEXt" in *The PracTEX Journal* 2005-02: `http://tug.org/pracjourn/`. It makes use of the `cfftot1` utility mentioned in footnote 9.

Software & Tools

Hacking DVI files: Birth of DVIasm

Jin-Hwan Cho

Abstract

This paper is devoted to the first step of developing a new DVI editing utility, called DVIasm. Editing DVI files consists of three parts: disassembling, editing, and assembling. DVIasm disassembles a DVI file to a human-readable text format (more flexible than DTL), and assembles the output back to a DVI file.

DVIasm is useful for people who have a DVI file without TEX source, but need to modify the document. It enables us to put a preprint number, a watermark, or an emblem on the document without touching the TEX source. DVIasm is attractive to even a TEX expert who wants to modify a few words in his document more than a hundred pages long.

We discuss in the paper how DVIasm supplements TEX. The current version supports only the standard DVI file format as in DVItype and DTL. The next versions will support 16-bit TEX extensions including Omega, pTEX, and XETEX.

1 Introduction

Have you ever heard of DVI, not the Digital Visual Interface[1] but the DeVice-Independent file format? In past years, every TEX user knew what DVI is and used DVI utilities to view and print TEX results. However, in recent times, TEX users have paid attention to DVI less and less because pdfTEX outputs directly to the PDF[2] file format. It is true without doubt that PDF is more powerful than DVI in almost all aspects. Then, do we have to obsolete DVI as PostScript is gradually replaced by PDF?

The DVI file format was designed by David R. Fuchs in 1979, in contrast to the release of PDF version 1.0 in 1993. It is intended to be both compact and easily interpreted by a machine [4, §14]. The most powerful aspect of DVI compared to PDF is nothing but *simplicity*. Imagine the speed of three previewers of DVI, PostScript, and PDF, and compare also the file size of the three different file formats. Furthermore, simplicity enables us to control DVI files in various ways. One of these is to edit DVI files directly — the main object of this paper.

There are many applications of editing DVI files. The most critical situation is when we have a DVI file without TEX source, but we want to modify or to add something to the document. A technical editor may want to put a preprint number on each paper without touching the TEX source. He may also want to put a watermark or an emblem on every paper.

Editing a DVI file is much faster for a TEX novice than learning TEX, when all he wants to do is to add some decorations to his document, and is not familiar with TEX codes. It may even be attractive to a TEX expert who wants to modify a few words in a long document.

Since a DVI file consists of binary data, it must be converted to a human readable format to inspect and edit its contents. The original DVI utility is DVItype [4], written by Donald E. Knuth in 1982. It has two chief purposes: to validate DVI files, and to give an example for developers of DVI utilities [4, §1]. DVItype is a nice utility to inspect the contents of a DVI file because of its human readable text output. However, it lacks any procedure for converting the output back to a DVI file.

A true DVI editing utility is the Device-independent Text Language (DTL) package [5] developed by Geoffrey Tobin. It includes two utilities `dv2dt` and `dt2dv` for converting from DVI to DTL and vice versa. It is notable that there is a one-to-one correspondence between DTL and DVI, and that DTL does not require TFM (font metric) files, in contrast to DVItype. However, DTL is not flexible for ordinary TEX users. For example, users must choose the correct command from `r1` to `r4` according to the width of the move to the right. Moreover, the latest version of DTL was released in 1995, and so it does not support extended DVI formats generated by Omega[3] or Japanese pTEX.[4]

The development plan for a new DVI editing utility, called DVIasm, consists of three phases. This paper is devoted to the first step, where DVIasm is introduced with several examples. The current version of DVIasm supports only the standard DVI file format, like DVItype and DTL, but is more flexible than DTL.

Editor's note: Reprinted from *The PracTEX Journal* 2007-1 (`http://tug.org/pracjourn`), by permission.

[1] A video interface standard designed to maximize the visual quality of digital display devices such as flat panel LCD computer displays and digital projectors [Wikipedia, `http://en.wikipedia.org/wiki/DVI`].

[2] PDF (Portable Document Format) is an open file format created and controlled by Adobe Systems, for representing two-dimensional documents in a device independent and resolution independent fixed-layout document format [Wikipedia, `http://en.wikipedia.org/wiki/PDF`].

[3] An extension of TEX by John Plaice and Yannis Haralambous, `http://omega.enstb.org`.

[4] ASCII Nihongo TEX by ASCII Corporation, `http://www.ascii.co.jp/pb/ptex/index.html`.

In the second phase we will focus on 16-bit characters, for instance, Chinese, Japanese, Korean, and Unicode, to support Omega, pTEX, and the subfont scheme[5] which enables us to use 16-bit characters in TEX and pdfTEX. In the final phase, DVIasm will communicate with the Kpathsea library, so that it will read font metric information from TFM, OFM, JFM, TrueType, and OpenType font files. DVIasm will also support XETEX[6] which reads font metric information directly from the font file itself.

2 Prerequisite

2.1 Download and installation

The current version of DVIasm is written in the Python programming language.[7] Why Python not C? The main reason is that Python does not require compiling and linking to get an executable file. Thus, DVIasm consists of a single Python program `dviasm.py` in a human-readable text format and it can run on any platform in which Python is installed. If speed-up is required later, some parts of DVIasm will be translated into C.

The development of DVIasm is controlled by Subversion, a popular version control system, and all revisions of DVIasm can be downloaded at [2]. From now on we assume that `dviasm.py` is in the working directory. The basic usage of DVIasm will be output if the option `--help` is specified as follows:

```
python dviasm.py --help
```

2.2 Creating a DVI file without TEX

For our first example, let's suppose we have saved following three lines as `hello.dump`. (The number at the beginning of each line is just the line number for reference and should not be typed.)

```
1  [page 1 0 0 0 0 0 0 0 0 0]
2  fnt: cmr10 at 50pt
3  set: 'Hello, World!'
```

Then run the following command:

```
python dviasm.py hello.dump -o hello.dump.dvi
```

to get a new DVI file, `hello.dump.dvi`. Its contents are shown in Figure 1(a).

(a) `hello.dump.dvi`

(b) `hello.dvi`

Figure 1: DVI result generated by (a) DVIasm and (b) TEX.

All DVI files in this paper are converted to PDF with DVIPDFMx[8] version 20061211. The DVI result can also be converted to PostScript with Dvips,[9] or viewed on the screen with the DVI previewers, xdvi,[10] dviout,[11] or yap.[12]

In the input, each page begins with the opening square bracket followed by the string '`page`' (without a colon), ten numbers, and the closing square bracket. Among the numbers the first one usually stands for the page number. In the second line the DVI command '`fnt:`' selects the Computer Modern font, `cmr10` scaled at 50 pt. In the last line the text '`Hello, World!`' is typeset by the command '`set:`'.

2.3 Disassembling a DVI file

We now try to disassemble a DVI file. First, make a TEX file `hello.tex` consisting of the following:

```
\nopagenumbers \font\fnt=cmr10 at 50pt
\noindent\fnt Hello, World! \bye
```

and run TEX (not LATEX) to get `hello.dvi`. The result is shown in Figure 1(b).

[5] The subfont scheme is a way of splitting a set of 16-bit characters into groups of 256 characters or less, the number of characters that TFM format can accommodate [3].

[6] A typesetting system based on a merger of TEX with Unicode and Mac OS X font technologies, by Jonathan Kew, `http://scripts.sil.org/xetex`.

[7] Python is a dynamic object-oriented programming language that runs on almost all operating systems. Just type '`python`' and hit the return key in the terminal to check whether Python is already installed or not. If not installed, visit the official website `http://www.python.org`.

[8] A DVI to PDF converting utility by Shunsaku Hirata and Jin-Hwan Cho, `http://project.ktug.or.kr/dvipdfmx/`. It is an extension of DVIPDFM written by Mark A. Wicks, `http://gaspra.kettering.edu/dvipdfm/`.

[9] A DVI to PostScript converter by Tom Rokicki, `http://www.radicaleye.com/dvips.html`.

[10] A DVI previewer in X Window system by Paul Vojta, `http://math.berkeley.edu/~vojta/xdvi.html`.

[11] The most popular DVI previewer in Japan that supports pTEX, `http://akagi.ms.u-tokyo.ac.jp/dviout-ftp.html`.

[12] The DVI previewer in the MiKTEX system by Christian Schenk, `http://www.miktex.org`.

```
[preamble]
id: 2
numerator: 25400000
denominator: 473628672
magnification: 1000
comment: ' TeX output 2007.01.24:1740'

[postamble]
maxv: 667.202545pt
maxh: 469.754990pt
maxs: 2
pages: 1

[font definitions]
fntdef: cmr10 (10.0pt) at 50.0pt

[page 1 0 0 0 0 0 0 0 0 0]
push:
  down: -14.0pt
pop:
down: 643.202545pt
push:
  down: -608.480316pt
  push:
    fnt: cmr10 (10.0pt) at 50.0pt
    set: 'Hello,'
    right: 16.666687pt
    set: 'W'
    right: -4.166702pt
    set: 'orld!'
  pop:
pop:
down: 24.0pt
```

Code 1: Output of disassembling `hello.dvi` with DVIasm.

One may easily find two points of difference between (a) and (b) in Figure 1. The first is the location of the text,[13] and the other one is the bar for the Polish letters ł and Ł[14] in (a) instead of the blank space in (b).

Looking at the figures closely, one more difference can be found: there is no kerning between the two characters 'W' and 'o' in (a). The kerning information is stored in TFM files; the implication is that DVIasm would need to communicate with the Kpathsea library to fetch the information. Thus, DVIasm no longer works if the whole TEX system is not installed. This is the reason why DTL and the current version of DVIasm do not require TFM files.

[13] The upper left corner of the paper has the coordinate (−1 in,−1 in), since the default x- and y-offsets are both one inch as usual. So the reference point of 'H' is the origin (0,0) in Figure 1(a). However, it is common to place the upper left corner of 'H' at the origin as in Figure 1(b).

[14] The ASCII code of the blank space is 32, and glyph at position 32 in `cmr10` is the bar for Polish ł and Ł.

```
[page 1 0 0 0 0 0 0 0 0 0]
putrule: 1cm 0.5pt
putrule: 0.5pt 1cm
push:
  down: -14.0pt
pop:
... (skip) ...
```

Code 2: Put a mark at the origin (0,0).

To see the exact differences, let us disassemble `hello.dvi` with DVIasm by running

```
python dviasm.py hello.dvi
```

to get the output[15] shown in Code 1. One can see four new commands, '`push:`', '`pop:`', '`right:`', and '`down:`'. An amount to move follows '`right:`' and '`down:`' as an argument. The meaning of these two commands seems clear.

However, there are two things to keep in mind. First, the coordinate system of DVI is different from the standard Cartesian coordinate system[16] used in PostScript and PDF. In DVI the x-coordinate increases from left to right, like Cartesian coordinates, but the y-coordinate increases from top to bottom, the opposite of Cartesian coordinates. Second, all positions in DVI are specified relatively, not absolutely. It is not possible in DVI to give a command like "go to the coordinate (100 pt, 100 pt)." Only '`right:`' and '`down:`' are allowed in DVI.

Then how do we move to a specific position in DVI? We can use the two commands '`push:`' and '`pop:`'. The command '`push:`' stores the current position in the stack, and '`pop:`' restores the position saved in the stack to the current position.

3 DVI commands

Let's now assume that the lines in Code 1 from the 17th line to the end are saved as `hello.dump`. The first example is to put some mark at the origin (0,0).

[15] DVIasm always outputs to standard output if the `-o` option is not specified.

[16] The Cartesian coordinate system is used to determine each point uniquely in a plane through a pair of numbers (x, y), usually called the x-coordinate and the y-coordinate of the point [Wikipedia, `http://en.wikipedia.org/wiki/Cartesian_coordinate_system`].

command	argument		description
`set:`	string		draw [string] and move to the right by the total width of the string
`put:`	string		draw [string] without moving to the right
`setrule:`	length1	length2	draw a box with width [length2] and height [length1] and then move to the right by [length2]
`putrule:`	length1	length2	draw a box with width [length2] and height [length1] without moving to the right
`push:`			save the current position to the stack
`pop:`			restore the position in the stack to the current position
`right:`	length		move to the right by [length] move to the left if [length] is negative
`down:`	length		move down by [length] move up if [length] is negative
`fnt:`	name `at` length		select the font [name] scaled at [length] [name] does not allow spaces
`xxx:`	string		DVI special command to be processed by DVI utilities; see the next section

Figure 2: DVIasm commands.

command	argument	description
`w:`	length	the same as `right:`, but [length] is stored in the 'w' variable
`x:`	length	the same as `right:`, but [length] is stored in the 'x' variable
`y:`	length	the same as `down:`, but [length] is stored in the 'y' variable
`z:`	length	the same as `down:`, but [length] is stored in the 'z' variable
`w0:`		move to the right by the length in the 'w' variable
`x0:`		move to the right by the length in the 'x' variable
`y0:`		move down by the length in the 'y' variable
`z0:`		move down by the length in the 'z' variable

Figure 3: DVIasm move commands.

This is achieved by inserting two lines after the first line, as in Code 2.

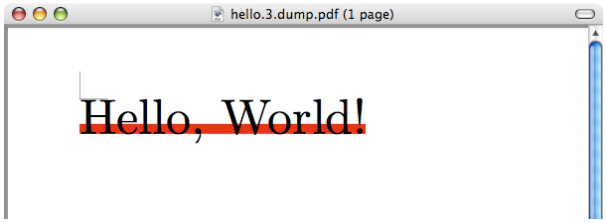
DVI has only two commands for drawing graphics, '`putrule:`' and '`setrule:`'. Both commands draw a box filled with black. The first and the second arguments indicate the size of the height and the width of the box, respectively. (Do not confuse the order of height and width!) The command '`setrule:`' is the same as '`putrule:`' except for moving to the right by the amount of the width after drawing the box.

The next example is to put a box filled with red *under* the text. Since DVI has no color command, Code 3 uses the special command '`xxx:`' that will be explained in the next section.

Exercise. Put the red box *over* the string to hide the overlapped part of the text.

Figures 2 and 3 give the input commands supported by DVIasm. There are two types of arguments, string and length. The string type consists of a text string surrounded by either apostrophes (') or double quotation marks ("). It has the same format as the Python string type.[17] The length type is either an integer or a floating point number followed

```
8   ... (skip) ...
9   down: -608.480316pt
10  xxx: 'color push rgb 1 0 0'
11  putrule: 10pt 4in
12  xxx: 'color pop'
13  push:
14  ... (skip) ...
```



Code 3: Put a box filled with red under the text.

[17] We can input any 8-bit character with hexadecimal value *hh* by '\x*hh*'. Thus, '\\' must be used to type the escape character '\(backslash)'.

```
1  [page 1 0 0 0 0 0 0 0 0 0]
2  xxx: 'papersize=6in,3in'
3  putrule: 1cm 0.5pt
4  putrule: 0.5pt 1cm
5  push:
6    down: -14.0pt
7  pop:
8  ... (skip) ...
```

Code 4: Resize the page of Code 3.

by unit (e.g., `sp`, `pt`, `bp`, `mm`, `cm`, `in`).[18] If no unit is specified, the number is in units of `sp` by default. The argument of '`fnt:`' is exceptional. The name of the font is given without apostrophes.

4 DVI specials

We saw all the DVI commands in the previous section, and we may note that there are no commands for color, graphics, or transformations in DVI. But we already know that they are possible in TEX. How do they work?

The answer is the DVI special command '`xxx:`'. It is the only way for TEX to communicate with DVI utilities. However, each DVI utility supports its own DVI specials. For example, neither DVIPDFM nor DVIPDFM*x* support a PostScript literal special containing PostScript code. On the other hand, almost none of the PDF specials work with Dvips.

In this section we introduce common DVI specials and show some examples using DVIasm. The material in this section is based on the author's talk at the TUG 2005 conference [1].

4.1 Page specials

There are two kinds of page specials. Code 4 is an example of the first, specifying a page size; it resizes the previous example (Code 3).

`papersize=[width],[height]` changes the size of whole pages. But it has no effect on the paper size that can be changed by the command line option

[18] 1 in = 2.54 cm = 25.4 mm = 72 bp = 72.27 pt, and 1 pt = 2^{16} sp = 65,536 sp

```
1  [page 1 0 0 0 0 0 0 0 0 0]
2  xxx: 'landscape'
3  putrule: 1cm 0.5pt
4  putrule: 0.5pt 1cm
5  push:
6    down: -14.0pt
7  pop:
8  ... (skip) ...
```

Code 5: Landscape orientation.

or by the configuration file (supported by Dvips*,[19] DVIPDFM, and DVIPDFM*x*).

`pdf:pagesize width [length] height [length]` changes the size of the page containing this special (supported by DVIPDFM*(?) and DVIPDFM*x*).

Code 5 shows an example of the second kind of page special: landscape paper orientation, rather than portrait.

`landscape` swaps the width and the height of the paper size (supported by Dvips*, DVIPDFM, and DVIPDFM*x*).

4.2 Color specials

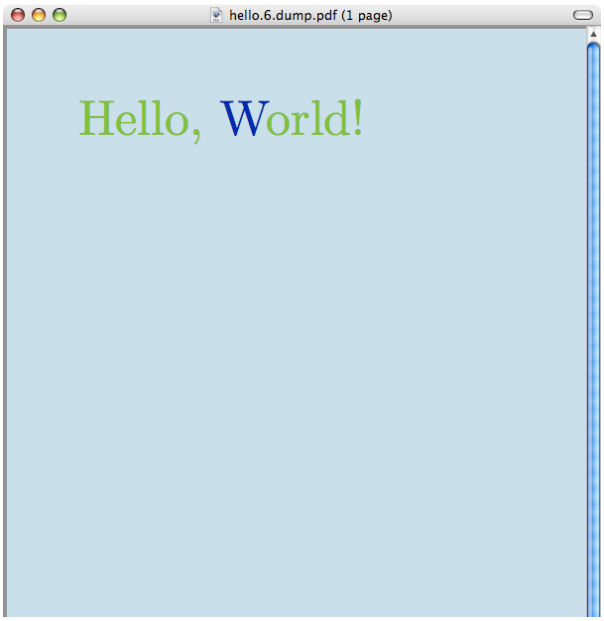
All of the common color specials originated with Dvips. In the specials below, color values can be specified in various ways (Code 6):

- `cmyk [c] [m] [y] [k]`
- `rgb [r] [g] [b]`
- `hsb [h] [s] [b]`
- `gray [g]`
- or a predefined color name.

The value of each color component is a number between 0.0 and 1.0. We refer to [6, pp. 12–13] and [1, p. 11] for PDF color specials, which are easier to understand than PostScript color specials.

[19] * denotes the original source of the feature, and (?) means that the behavior looks mysterious or buggy.

```
1  [page 1 0 0 0 0 0 0 0 0 0]
2  xxx: 'background cmyk .183 .054 0 0'
3  down: 643.202545pt
4  push:
5    down: -608.480316pt
6    xxx: 'color push LimeGreen'
7    push:
8      fnt: cmr10 (10.0pt) at 50.0pt
9      set: 'Hello,'
10     right: 16.666687pt
11     xxx: 'color push rgb 0 0 .625'
12     set: 'W'
13     xxx: 'color pop'
14     right: -4.166702pt
15     set: 'orld!'
16   pop:
17   xxx: 'color pop'
18 pop:
```



Code 6: Example of coloring background and text.

`background [PScolor]` sets a fill color for the background (supported by Dvips*, DVIPDFM, and DVIPDFMx).

`color push [PScolor]` saves the current color on the color stack and sets the current color to the given one (supported by Dvips*, DVIPDFM, and DVIPDFMx).

`color pop` pops a color from the color stack and sets the current color to be that color (supported by Dvips*, DVIPDFM, and DVIPDFMx).

`color [PScolor]` clears the color stack, and saves and sets the given color (supported by Dvips*, DVIPDFM(?), DVIPDFMx).

```
[page 1 0 0 0 0 0 0 0 0 0]
down: 150bp
xxx: 'psfile=tiger.eps rhi=1500
      llx=17 lly=171 urx=617 ury=771 clip'
right: 150bp
xxx: 'psfile=tiger.eps rhi=750
      llx=17 lly=171 urx=617 ury=771
      angle=45 clip'
right: 75bp
xxx: 'psfile=tiger.eps rwi=1500 rhi=750
      llx=17 lly=171 urx=617 ury=771 clip'
right: 150bp
xxx: 'psfile=tiger.eps rwi=750 rhi=1500
      llx=17 lly=171 urx=617 ury=771 clip'
```

Code 7: Manipulating an image. (Line breaks are editorial.)

4.3 Image specials

The special '`psfile`' is used for including an EPS (PostScript) graphics file. EPS files contain bounding box information. For example, the bounding box of the EPS file in the following example[20] is

```
%%BoundingBox: 17 171 567 739
```

Four options `llx`, `lly`, `urx`, and `ury` specify the clipping area of the EPS file, and two options `rwi` and `rhi` (0.1 bp unit) are used to resize the clipped area.

```
psfile=[name] hsize=[num] vsize=[num]
hoffset=[num] voffset=[num]
hscale=[num] vscale=[num] angle=[num]
llx=[num] lly=[num] urx=[num] ury=[num]
rwi=[num] rhi=[num] [clip]
```

Although Dvips*, DVIPDFM, and DVIPDFMx all recognize `psfile`, neither DVIPDFM nor DVIPDFMx have internal PostScript interpretation support, so they cannot process EPS files without Ghostscript or another PostScript "distiller" available.

However, both DVI utilities support JPEG and PDF image files, which are not processed by Dvips. The PDF image special for JPEG and PDF images

[20] Namely `tiger.eps`, which can be found in the `examples` directory of Ghostscript, the most popular free software interpreter (available under the GPL) for PostScript and PDF. See `http://www.ghostscript.com` for more information.

has reader-friendly syntax. We refer to [6, p. 13] and [1, pp. 12–14] for examples.

```
pdf:image width [length] height [length]
depth [length] rotate [num]
scale [num] xscale [num] yscale [num]
bbox [ulx] [uly] [lrx] [lry]
matrix [a] [b] [c] [d] [x] [y] ([name])
```

(Supported by DVIPDFM*(?) and DVIPDFMx).

4.4 Transformation specials

It is possible in LaTeX to rotate and scale text and figure. But Dvips has no transformation special for this purpose. Instead, it enables us to insert literal PostScript code.

`" [PScode]` inserts literal PostScript code surrounded by a `gsave` and `grestore` pair, so that it will have no effect on the rest of the document (supported by Dvips* only).

`ps:[PScode]` inserts literal PostScript code without `gsave` and `grestore` (supported by Dvips* only).

The code for the example above follows (line breaks in the long specials are editorial):

```
[page 1 0 0 0 0 0 0 0 0 0]
xxx: 'papersize 2in,2in'
xxx: '" Goldenrod newpath 0 0 moveto 50 0 lineto
       0 0 50 0 90 arc closepath fill'
xxx: '" Dandelion newpath 0 0 moveto 0 50 lineto
       0 0 50 90 180 arc closepath fill'
xxx: '" Apricot newpath 0 0 moveto -50 0 lineto
       0 0 50 180 270 arc closepath fill'
xxx: '" Peach newpath 0 0 moveto 0 -50 lineto
       0 0 50 270 0 arc closepath fill'
xxx: 'color gray 1'
fnt: ptmr8r at 50pt
xxx: 'ps:gsave'
put: 'A'
xxx: 'ps:currentpoint currentpoint translate
         90 rotate neg exch neg exch translate'
put: 'A'
xxx: 'ps:currentpoint currentpoint translate
      90 rotate neg exch neg exch translate'
put: 'A'
xxx: 'ps:currentpoint currentpoint translate
      90 rotate neg exch neg exch translate'
put: 'A'
xxx: 'ps:grestore'
```

On the other hand, DVIPDFM and DVIPDFMx have a PDF transformation special for rotation and scaling, etc. Note that literal PDF code is used in the following example.

`pdf:btrans [`*same options as* `pdf:image]` applies the specified transformation to all subsequent text (supported by DVIPDFM* and DVIPDFMx).

`pdf:etrans` concludes the action of the immediately preceding `pdf:btrans` special (supported by DVIPDFM* and DVIPDFMx).

`pdf:content [PDFcode]` inserts literal PDF code surrounded by a `q` and `Q` pair, so it will have no effect on the rest of the document (supported by DVIPDFM* and DVIPDFMx).

`pdf:literal [PDFcode]` inserts literal PDF code without the `q` and `Q` pair (supported by DVIPDFMx* only).

Here is the PDF implementation of the figure above (again, line breaks are editorial):

```
[page 1 0 0 0 0 0 0 0 0 0]
xxx: 'papersize 2in,2in'
xxx: 'color Goldenrod'
xxx: 'pdf:content 0 0 m 50 0 l
      50 25 25 50 0 50 c f'
xxx: 'color Dandelion'
xxx: 'pdf:content 0 0 m 0 50 l
      -25 50 -50 25 -50 0 c f'
xxx: 'color Apricot'
xxx: 'pdf:content 0 0 m -50 0 l
      -50 -25 -25 -50 0 -50 c f'
xxx: 'color Peach'
xxx: 'pdf:content 0 0 m 0 -50 l
      25 -50 50 -25 50 0 c f'
xxx: 'color gray 1'
fnt: ptmr8r at 50pt
put: 'A'
xxx: 'pdf:btrans rotate 90 scale .5'
put: 'A'
xxx: 'pdf:btrans rotate 90 scale 2'
put: 'A'
xxx: 'pdf:btrans rotate 90 scale 2'
put: 'A'
xxx: 'pdf:etrans'
xxx: 'pdf:etrans'
xxx: 'pdf:etrans'
```

(This figure is not quite circular, compared to the previous PostScript one, since that would require much longer code.)

To this point, we have discussed common DVI specials, mostly originated by Dvips. There are also many PDF specials not yet mentioned. DVIPDFM originates almost all PDF specials, and its manual [6]

is a good source. Moreover, the present author discussed at TUG 2005 [1] how differently the three DVI utilities, Dvips, DVIPDFM, and DVIPDFM*x* behave on the same special command.

5 Conclusion

Imagine that one has a DVI file without TEX source, but wishes to modify or to add something to the document. For example, a technical editor may want to put a preprint number on each paper, which was not fixed at the time of writing. He may also want to put a watermark or an emblem on every paper.

We also imagine a TEX novice who wants to include some decorations in his document, but has trouble writing TEX code. Is it the best advice for him to learn TEX? It might be — if he has enough time. If not, DVIasm is an alternative. In fact, he may learn DVI commands more quickly than TEX commands. DVIasm may even be attractive to a TEX expert who wants to modify a few words in a long document.

DVIasm is written for these purposes, as a supplement to TEX and its extended versions. Of course, DVIasm is not an alternative to TEX! Neither line breaking nor page breaking is (or ever will be) supported.

As mentioned at the beginning of the paper, DVIasm development is in its first phase. The next paper will discuss how to support 16-bit characters in DVIasm. Any comments are welcome, and will be helpful to improve the program.

References

[1] Jin-Hwan Cho, *Practical Use of Special Commands in DVIPDFMx*, TUG 2005, International Typesetting Conference. Wuhan, China. `http://project.ktug.or.kr/dvipdfmx/doc/tug2005.pdf`

[2] Jin-Hwan Cho, *The DVIasm Python script.* `http://svn.ktug.or.kr/viewvc/dviasm/?root=ChoF`

[3] Jin-Hwan Cho and Haruhiko Okumura, *Typesetting CJK languages with Omega.* TEX XML, and Digital Typography, Lecture Notes in Computer Science **3130** (2004), 139–148.

[4] Donald E. Knuth, *The DVItype processor* (Version 3.6, December 1995). `http://ctan.org/tex-archive/systems/knuth/texware/dvitype.web`.

[5] Geoffrey Tobin, *The DTL Package* (Version 0.6.1, March 1995). `http://ctan.org/tex-archive/dviware/dtl/`.

[6] Mark A. Wicks, *DVIPDFM User's Manual* (Version 0.12.4, September 1999). `http://gaspra.kettering.edu/dvipdfm/dvipdfm-0.12.4.pdf`.

◇ Jin-Hwan Cho
Department of Mathematics
The University of Suwon
Republic of Korea
`chofchof (at) ktug dot or dot kr`

Graphics

A complex drawing in descriptive geometry

Denis Roegel

Abstract

This article describes the reproduction of a complex drawing in descriptive geometry. The original plate was published by Théodore Olivier in 1842 and represents the meshing of two gears on skew axes. The drawing was analyzed and redone in METAPOST, and the construction illustrates a number of typical features of that programming environment.

1 Introduction

The French geometer Gaspard Monge (1746–1818) developed "descriptive geometry", and this discipline flourished during the 19th century, at the same time as the industrial revolution, and the construction of machines, buildings, and other masterpieces of architecture.

Geometry, and in particular descriptive geometry, is a wonderful application for METAPOST enthusiasts (Goossens, Mittelbach, Rahtz, Roegel, and Voß, 2007). METAPOST makes it possible to draw lines and curves in a very exact way, and at the same time relate different parts of a drawing to each other, as they should be in descriptive geometry.

In this article, I go into the details of the construction of a complex drawing, taken from the work of Théodore Olivier (1793–1853), one of Monge's best followers. Olivier was a former student of the *École Polytechnique* and went on to do groundbreaking work in the geometrical theory of gears. In 1829 he was one of the founders of the *École Centrale des Arts et Manufactures*. One of his most important books is his *Théorie géométrique des engrenages* (Olivier, 1842), where exact drawings for gearings are produced in an almost purely geometrical way. Application of this theory led to the many models found in the *Musée des arts et métiers* in Paris.

One of the chapters in his book is devoted to the meshing of wheels with non-intersecting axes, and my purpose is to show how Olivier's corresponding drawing (figure 1) can be produced with a tool such as METAPOST. But this article is neither meant as an introduction to the theory of gears, nor to the rules of descriptive geometry. I will focus only on geometrical relations, without always stating why things are so or not. I am taking the vantage point of an engineer who has some drawing to produce, of which he/she knows the geometrical relationships, but the grounds for these relationships will not be essential in our analysis. The reader interested in more details may consult Olivier's book or other books on the theory of gears.

Figure 2 shows the final figure produced with our code.

2 Olivier's plate

Olivier writes (Olivier, 1842, p. 118) that the plate represents the original-scale working-drawing which was used to manufacture the gearing-model to transmit rotation motion between two axes not in the same plane and having an angle of 30° between them.

LT (figure 2) represents the "ground line" (*ligne de terre*). The axis A of the wheel — carrying twenty-four cylindrical teeth with circle involute shapes — is vertical; its projections are A^h and A^v.

The axis A_1 of the wheel — carrying eighteen helical teeth — is in the vertical projection plane; its projections are A_1 and A_1^h or LT.

The middle lines M and M_1 are, on the vertical projection plane, the vertical traces of the two planes dividing the rings to cut in equal parts.

These lines M and M_1 intersect at X^v which is the vertical projection of the line X, intersection of the two planes dividing the rings in equal parts.

The wheel attached to the inclined axis A_1 was turned around the line X, in order to bring this wheel into a horizontal position.

It is in this horizontal position that the wheel carrying eighteen teeth is represented by the working drawing.

T is a vertical plane tangent to the cylinders H and H_1. The vertical line Y^v is in the plane T.

For each wheel, the outer circle shows the base circle of the involutes. The inner circle shows how far the teeth from the other wheel penetrate one wheel.

Olivier's plate is also reproduced in von Seherr-Thoss's book on the development of gearing technology (von Seherr-Thoss, 1965, p. 120), but the drawing was redone and some errors were not corrected (figure 3).

3 Involute cylindrical and helical teeth

3.1 Involute teeth

Figure 4 shows the construction of the involute curve of a circle. It is straightforward to obtain its cartesian equations which are:

$$x_M = r(\cos\theta + \theta\sin\theta)$$
$$y_M = r(\sin\theta - \theta\cos\theta)$$

with θ being expressed in radians.

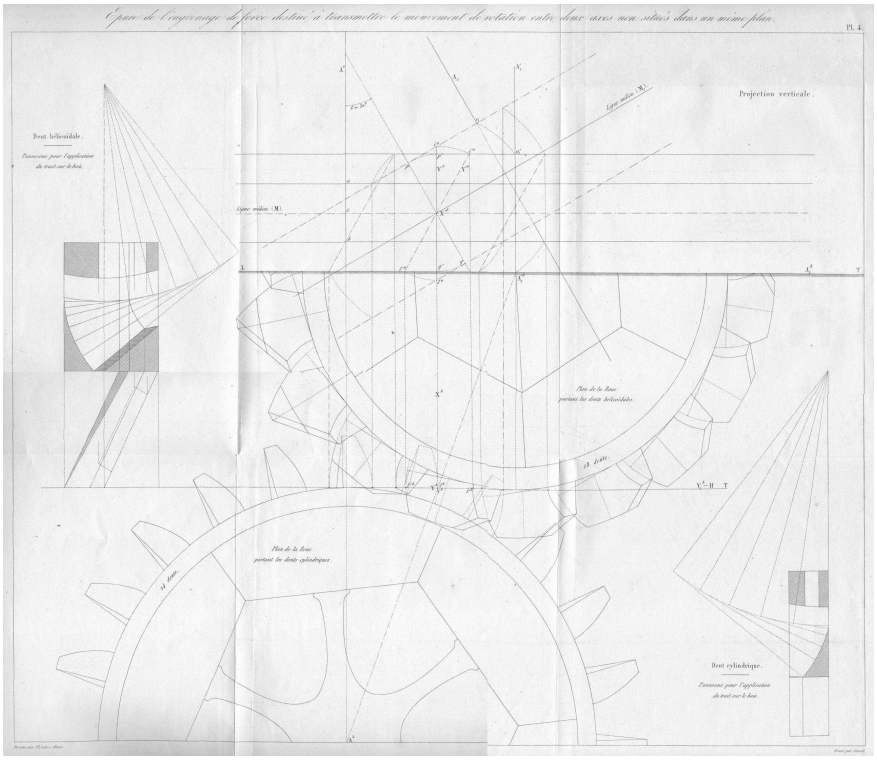


Figure 1: Olivier's plate, with a frame measuring 35cm × 42.5cm (Olivier, 1842). Note that a tooth is missing on the lower wheel.

The following METAPOST code returns a path for an involute tooth. The three parameters of the macro `involute_tooth` are the radius r of the base circle, an angle a for truncating the involute, and the angular step s for θ. It turns out that $s = 20$ gives an excellent approximation of the involute curve.

```
RAD=3.14159/180; % conversion degrees -> radians
vardef involute_tooth(expr r,a,s)=
  save p,t;
  path p;
  p=(r,0)
  for i=1 step s until 90:
    ..(r*(cosd(i) + RAD*i*sind(i)),
       r*(sind(i) - RAD*i*cosd(i)))
  endfor;
  t=xpart(p intersectiontimes
             (origin--(3r*dir(a))));
  (subpath(0,t) of p)
enddef;
```

Two more macros are defined for the linear parts of the cylindrical teeth of the lower wheel:

```
vardef cyl_full_tooth(expr r,a,b)=
  save p,L;path p;pair L;
  p=involute_tooth(r,a,20);
  L=point length(p) of p;
  (p--(L rotated b)--(r*dir(a+b)))
enddef;
vardef cyl_full_tooth_x(expr r,a,b,c)=
  save p,L;
  path p;pair L;
  p=involute_tooth(r,a,20);
  L=point length(p) of p;
  ((L rotated b)--(r*dir(a+b+c)))
enddef;
```

3.2 Tooth contact

Teeth can be put in contact easily, provided the shapes of the curves to put in contact and the point of contact are known. We then only needed to find out how much the standard teeth curves need to be rotated.

In the example below, two involutes p_1 and p_2 are defined, and the angles a and b (from the centers of the respective circles) between the horizontal and the contact of the involutes with circles corresponding to the final contact are determined. Then the two involutes are merely rotated according to these two angles (see figure 5):

```
p1=involute_tooth(v,25,5);p2=p1 shifted z2;
a=angle(p1 intersectionpoint
           circle(z1,.5arclength(z1--z2)));
b=angle((p2 intersectionpoint
           circle(z2,.5arclength(z1--z2)))-z2);
draw p1 rotated -a;
draw p2 rotatedaround(z2,180-b);
```

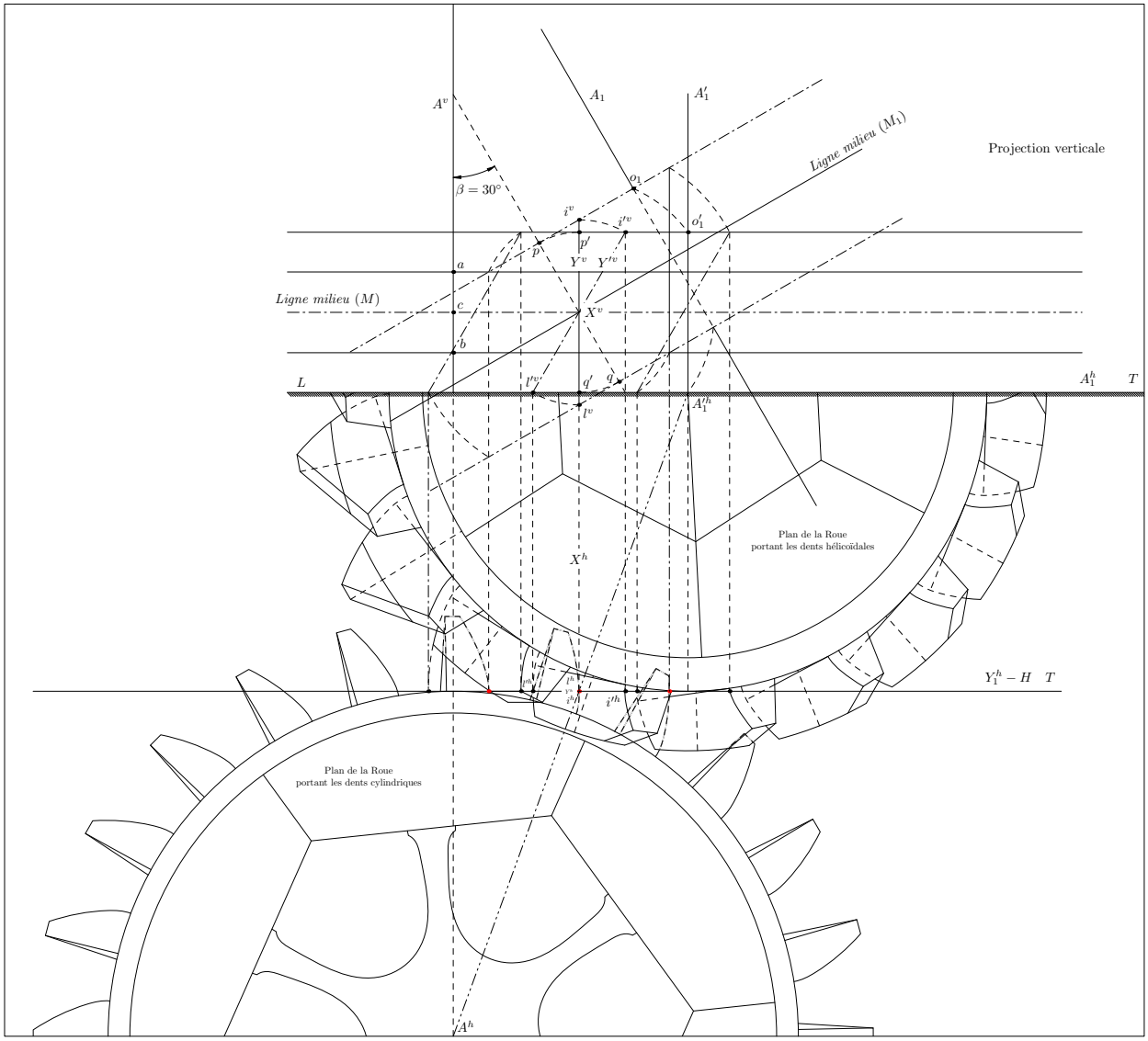


Figure 2: The entire figure (except the ancillary drawings within the frame) reproduced from Olivier's plate.

3.3 Helical teeth with involute profiles

The teeth in Olivier's upper (helical) wheel are obtained from the envelope of the tangents to a helix (see figure 6). The intersections of the envelope with planes orthogonal to the cylinder axis are circle involutes. This can be proved easily: let the helix equation be

$$M = \begin{pmatrix} r\cos\theta \\ r\sin\theta \\ a\theta \end{pmatrix}$$

A tangent vector to the helix in M is

$$\frac{dM}{d\theta} = \begin{pmatrix} -r\sin\theta \\ r\cos\theta \\ a \end{pmatrix}$$

A point $T(\alpha)$ of the tangent has the coordinates

$$T(\alpha) = \begin{pmatrix} r\cos\theta - \alpha r\sin\theta \\ r\sin\theta + \alpha r\cos\theta \\ a\theta + \alpha a \end{pmatrix}$$

Taking $a\theta + \alpha a = 0$, we obtain $\alpha = -\theta$ and therefore the intersection with the plane $z = 0$ is

$$I(\theta) = \begin{pmatrix} r(\cos\theta + \theta\sin\theta) \\ r(\sin\theta - \theta\cos\theta) \\ 0 \end{pmatrix}$$

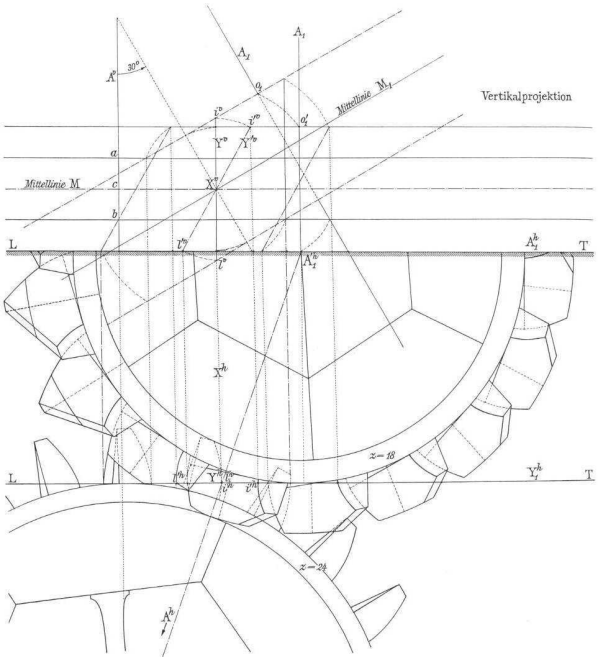


Figure 3: Seherr-Thoss's reproduction of Olivier's drawing (von Seherr-Thoss, 1965, p. 120). The most obvious changes are the German labels, but actually, as a close examination shows, the whole drawing was redone.

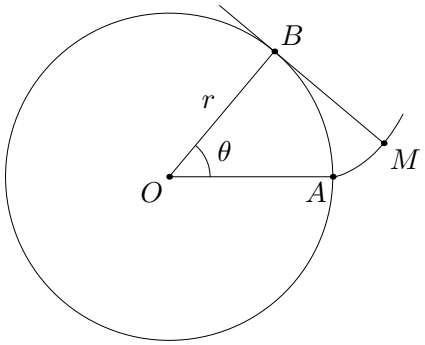
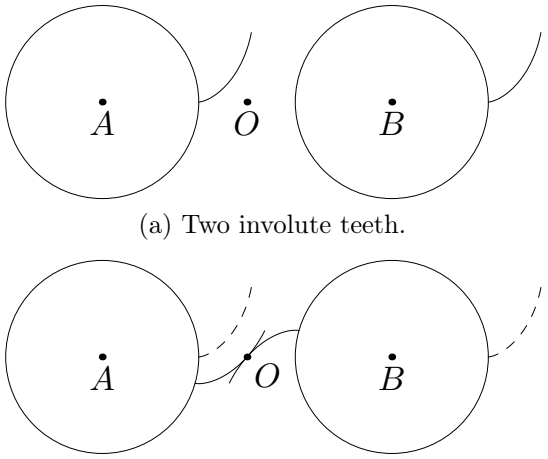


Figure 4: The construction of an involute.



(a) Two involute teeth.

(b) The two rotated teeth meeting in O.

Figure 5: The contact of two involute teeth.

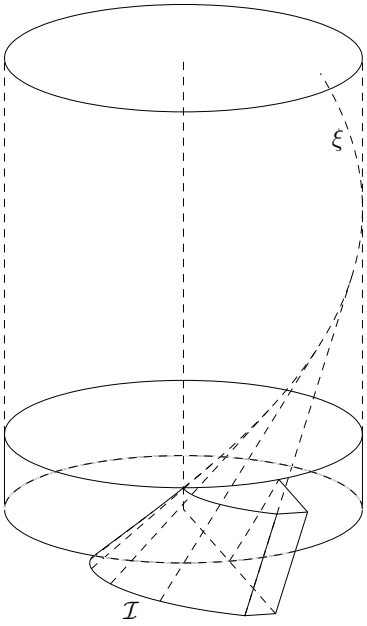


Figure 6: Helical tooth with involute profiles. The tooth is generated by an involute $\mathcal{I}$ moving along a helix ξ. (In this drawing, the four non-working faces of the tooth have been made planar.)

which is exactly the equation of the involute described above. The intersection with any other plane $z = h$ also gives an involute, of course.

4 The construction

Olivier's drawing is reconstructed as follows:

4.1 Units

All the dimensions are expressed as multiples of a conventional unit, so that it becomes easy to scale the figure afterwards. In this case, the unit u was 1 cm on the original drawing. Changing the unit still affects the drawing, in that text and line widths are usually not changed when the unit is changed (but they could be taken into account).

```
numeric u;
u=5mm;
```

4.2 Useful macros

A few useful macros are defined, in particular a shortcut for a circle of center c and radius r:

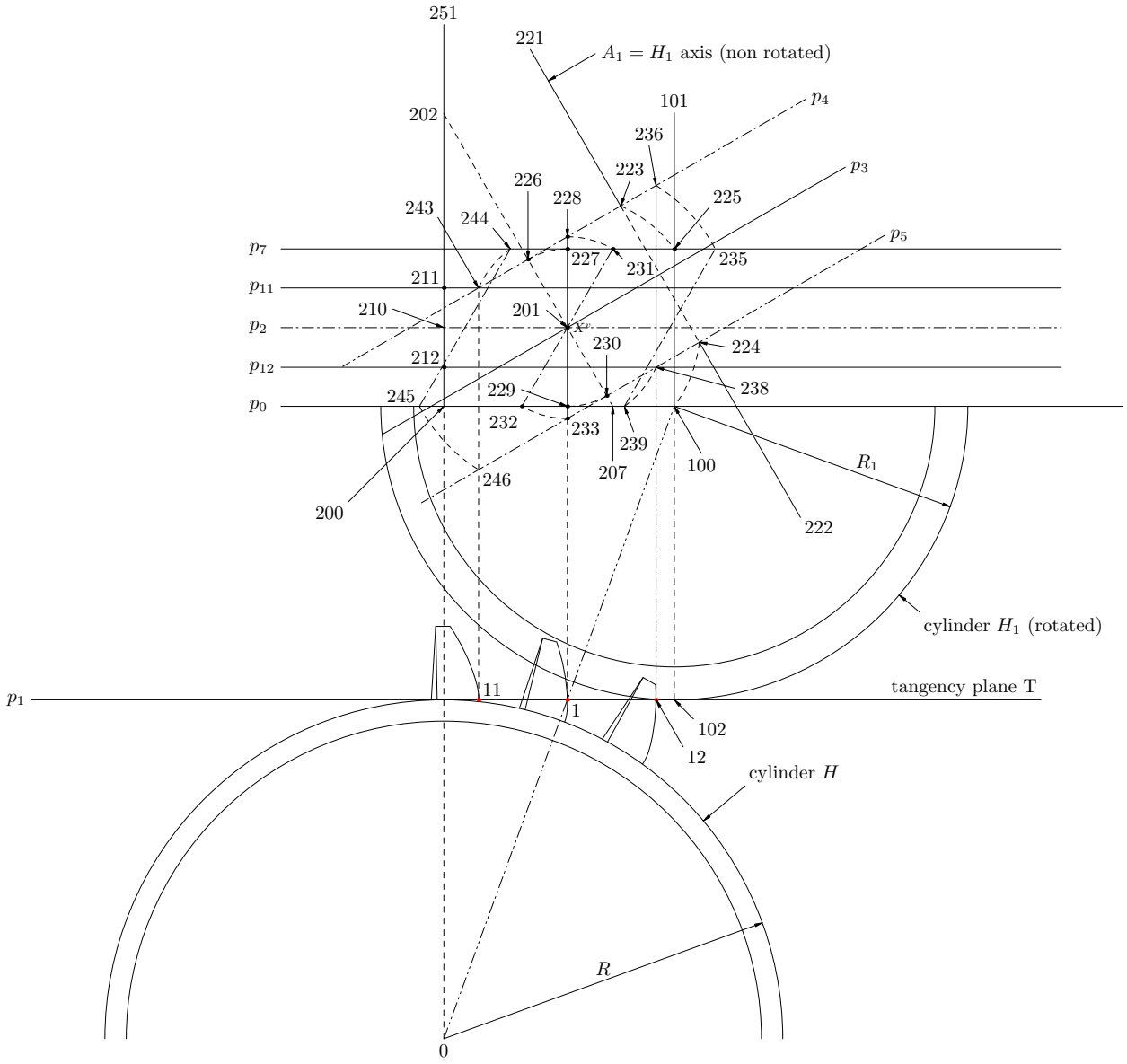


Figure 7: Step 1. The distance $D = z_0 - z_{200}$ is the shortest distance between the axes A and A_1.

```
def circle(expr c,r)=
  (fullcircle scaled 2r shifted c)
enddef;

% pathpoint(p) is some point on
%   (point 0 of p, point 1 of p)
def pathpoint(text p)=
  whatever[point 0 of p,point 1 of p]
enddef;
```

4.3 General layout

We start with the general layout of the figure, and its various parameters. The number of teeth of the lower (cylindrical) wheel is $n_a = 24$, whereas the number of teeth of the upper (helical) wheel is $n_b = 18$. The angle between the two wheel axes is $\beta = 30°$. The (shortest) distance between the two axes is d_0.

On figure 7, the "ground line" is p_0; this separates the horizontal projection (bottom) from the vertical projection (top). Below the p_0 line, we see two wheels, one with a vertical axis A (located at z_0), and another with an inclined axis A_1. However, on the horizontal projection, the wheel was first turned, so that its axis in fact is also vertical. The base cylinders of the two wheels therefore appear as circles on the horizontal projection.

p_0 is also the (horizontal) projection of the A_1

axis and point z_{200} is the projection of the point on A_1 which is at the closest distance from A.

```
na=24;     % number of teeth of wheel 0
nb=18;     %                          1
beta=30;   % angle between the wheel axes
z0=origin; % center of wheel 0
d0=23.3u;  % distance between the axes of the wheels
z200-z0=d0*up;
```

The two cylinders H (lower wheel) and H_1 (upper wheel) are provided with teeth and these teeth are positioned on two bands. The (vertical) projections of the teeth for H are located between p_0 and p_7, whereas those for H_1 are located between p_4 and p_5. The position of these bands determines the contact between the two wheels and is a parameter of the drawing. The intersection of the median planes of these bands is a line X whose vertical projection is $z_{201} = X^v$. We use d_1 and d_2 as parameters locating X^v. The line p_0 can now be defined.

```
d1=2.9u;  % distance from X to the line LT
d2=4.55u; % distance from X to the vertical A^v
z201-z200=(d2,d1);z210-z200=d1*up;
z203=z200+6u*left;
z204=z200+25u*right;p0=z203--z204; % LT line
```

The radii r_1 and r_2 of the two cylinders depend both on the number of teeth and on the angle between the two axes. These values will serve as base radii for the teeth's involute curves, but teeth will be undercut below these base radii for deeper meshing. The undercut radii are r_{11} and r_{21}. Finally, z_1 is the intersection between X and the tangency plane common to the two cylinders H and H_1. The (horizontal) projection of this tangency plane T is p_1.

```
r1=d0/(1+(nb/na)/cosd(beta)); % radius of wheel 0
r2=d0-r1;                     % radius of wheel 1
r11=11.7u; % inner radius of wheel 0
r21=9.6u;  % inner radius of wheel 1

z1=(x201,r1);
z2=z0+r1*up+15.2u*left;
z3=z0+r1*up+22u*right;
p1=z2--z3; % tangent between the wheels
```

The upper wheel is rotated around X by an angle β and the horizontal projection of its center is z_{100}. The intersection of p_3 and A_1 is z_{220}. The (vertical) projection of the point on A_1 closest to A is z_{219}. Finally, we compute z_{202} and z_{207}, a segment parallel to A_1 and going through X^v.

```
z100=whatever[z0,z1]
    =whatever[z200,z200+right];
z220=z201+(x100-x201)*dir(beta);
z219=whatever[z200,z210]
    =whatever[z220,z220+dir(90+beta)];
z202=whatever[z201,z201+z219-z220]
    =whatever[z210,z219];
z207=whatever[z202,z201]
    =whatever[z200,z100];
```

Two contours p_{101} and p_{102} are now determined for the purpose of framing or clipping parts of the drawing under construction: p_{101} frames the whole drawing, and p_{102} frames the upper part. These contours will be used to hide parts of the teeth, as in Olivier's original plate.

```
% contours:
z301=z0+1.3r1*left;
z302=(xpart(point 1 of p0),y0);
z303=(x302,y100+1.3r2);
z304=(x301,y303);
% frame for the whole drawing:
p101=z301--z302--z303--z304--cycle;
z251=(x0,y303); % vertical edge
% contour for clipping the upper teeth:
z305=(x303,y100);
z306=(x304,y100);
p102=z301--z302--z305--z306--cycle;
```

The lines M, M_1 (dividing the teeth bands) and their parallels are now easily defined:

```
p2=(xpart(point 0 of p0),y210)
        --(z201+4(z201-z210)); % M
p3=(z220+2(z220-z201))--(z201-2(z220-z201)); % M1
p4=p3 shifted (d1*dir(90+beta));  % parallel to M1
p5=p3 shifted (d1*dir(-90+beta)); % parallel to M1
p7=p2 shifted (d1*up);
```

Next we define p_6 (the A_1 axis) using z_{221} and z_{222}. This axis being partly drawn dashed, we introduce two intermediate points z_{223} and z_{223}.

```
z221=.4[z219,z220];z222=(z220+.65(z220-z219));
p6=z221--z222; % A1 axis
z223=p6 intersectionpoint p4;
z224=p6 intersectionpoint p5;
```

The projection of the rotated axis A_1 is p_8:

```
z101-z100=z100-z102=r2*up;
p8=z101--z102;
```

The contacts between the helical teeth of the upper wheel and the cylindrical teeth of the lower wheel are vertical segments which are parallel to A, and one of them is the segment (z_{228}, z_{233}). Since the upper wheel is rotated for the drawing, the segments need to be rotated too. A number of arcs on figure 7 show the points and their position after rotation. For instance, z_{223} (intersection of axis A_1 and p_4) becomes z_{225}. For the main contact between the wheels, we define points z_{226} (p), z_{227} (p'), z_{228} (i^v), z_{229} (q'), z_{230} (q), z_{231} (i'^v), z_{232} (l'^v) and z_{233} (l^v). The segment $p_9 = l'^v - i'^v$ is the vertical projection of the main rotated contact.

The vertical line through X^v is tangent to one of the cylindrical teeth. The next vertical tangent to the right is at a distance which is the circumference of the lower wheel divided by n_a. The intersection with LT is z_{234}. The contact segment on the upper band is (z_{236}, z_{238}). The horizontal projection of that contact is point z_{12} on the cylinder tangency line.

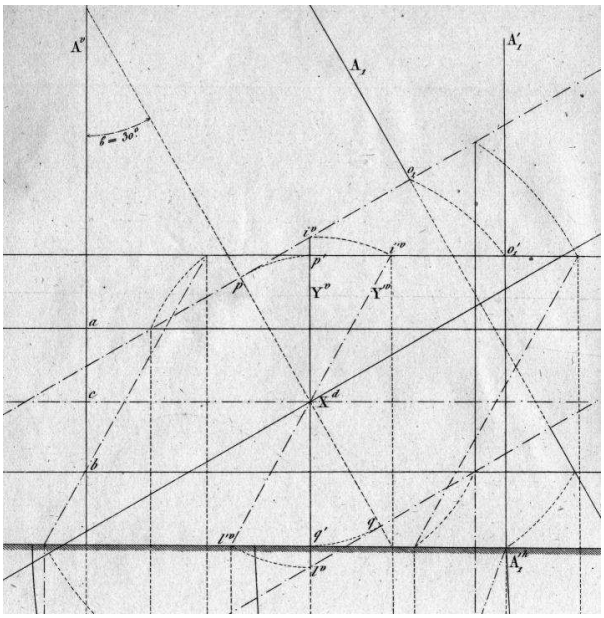


Figure 8: Excerpt of Olivier's plate: contact segments. The arc at the lower right corner seems incorrect.

```
z225=p8 intersectionpoint p7;

% rotation arcs
z226=z201+d1*dir(90+beta); % p
z227=z201+d1*up; % p'
z228=whatever[z201,z201+up]
    =whatever[z226,z226+dir(beta)]; % i^v
z229=z201+d1*down; % q'
z230=z201+d1*dir(-90+beta); % q
z231=z228 rotatedaround(z201,-beta); % i'v
z232-z201=z201-z231; % 232=l'v
z233-z201=z201-z228; % 233=l^v
p9=z232--z231; % l'v - i'v
```

At a given time, several teeth are in contact. In our case, three teeth are in contact. We call these teeth 1, 2 and 3. The segment p_9 is the contact for the second lower tooth.

```
z234=z229+(2*3.14159*r1/na)*right;
% intersection with p4 (=upper line parallel to M1)
z236=pathpoint(p4)=z234+whatever*up;
% intersection with p5 (=lower line parallel to M1)
z238=(z236--z234) intersectionpoint p5;
z12=(x236,y0+r1); % contact with lower tooth
```

p_{10} is the rotated contact segment for the third tooth. It is obtained by translating the second contact segment.

```
z235=z236 rotatedaround(z201,-beta); % end of arc
% parallel to l'v-i'v:
p10=p9 shifted (z235-z231);
z239=point 0 of p10; % p10=z239--z235
```

z_{238} is used to define the two lines p_{11} and p_{12}.

```
z241=(z236--z12) intersectionpoint p2;
```

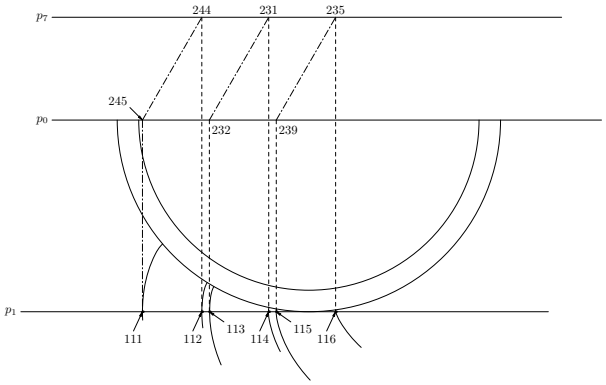


Figure 9: Step 2. Involutes tangent to the vertical lines going through z_{111}, ..., z_{116}.

```
% parallel to M going through 'a':
p11=p2 shifted (z241-z238);
% parallel to M going through 'b':
p12=p2 shifted (z238-z241);
```

z_{211} and z_{212} are the intersections of p_{11} and p_{12} with the vertical projection of A:

```
z211=p11 intersectionpoint (z0--z219); % a
z212=p12 intersectionpoint (z0--z219); % b
```

The vertical tangent to the first tooth, going through z_{242}, is obtained from the two other tangents (going through z_{229} and z_{234}). z_{11} is the contact with the first tooth on the tangency line T.

```
z242=z229-(z234-z229);
% intersection with p11:
z243=(x242,y241+(y241-y238));
z11=(x242,y0+r1);
p13=z11--z243; % vertical tangent to lower tooth 1
```

Next we rotate the contact segment and obtain p_{14}.

```
z244=z243 rotatedaround(z201,-beta);
% parallel to l'v-i'v:
p14=p9 shifted (z244-z231);
z245=point 0 of p14; % p14=z245--z244
z246=z245 rotatedaround(z201,beta);
```

4.4 Tangents to the lower teeth

The three contact segments have the horizontal projections z_{11}, z_1 and z_{12}. The intersections of these segments with the planes whose (vertical) projections are p_4 and p_5 are located on involutes, as shown in figure 6. These involutes are actually tangent to a line orthogonal to the (horizontal) projection of the contact segment, as one can easily see. The six involutes under consideration go through the points z_{111}, ..., z_{116} (figure 9).

```
% tangent 1 with upper tooth 1:
z111=(x245,y0+r1);p15=z245--z111;
z112=(x244,y0+r1);p16=z244--z112; % tang. 2/tooth 1
z113=(x232,y0+r1);p17=z232--z113; % tang. 1/tooth 2
z114=(x231,y0+r1);p18=z231--z114; % tang. 2/tooth 2
z115=(x239,y0+r1);p19=z239--z115; % tang. 1/tooth 3
z116=(x235,y0+r1);p20=z235--z116; % tang. 2/tooth 3
```

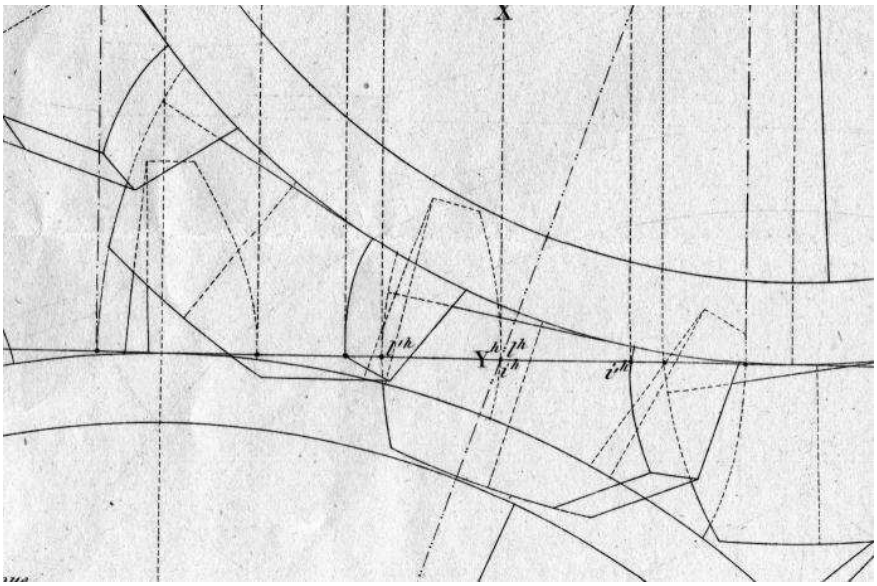


Figure 10: Excerpt of Olivier's plate: upper teeth construction.

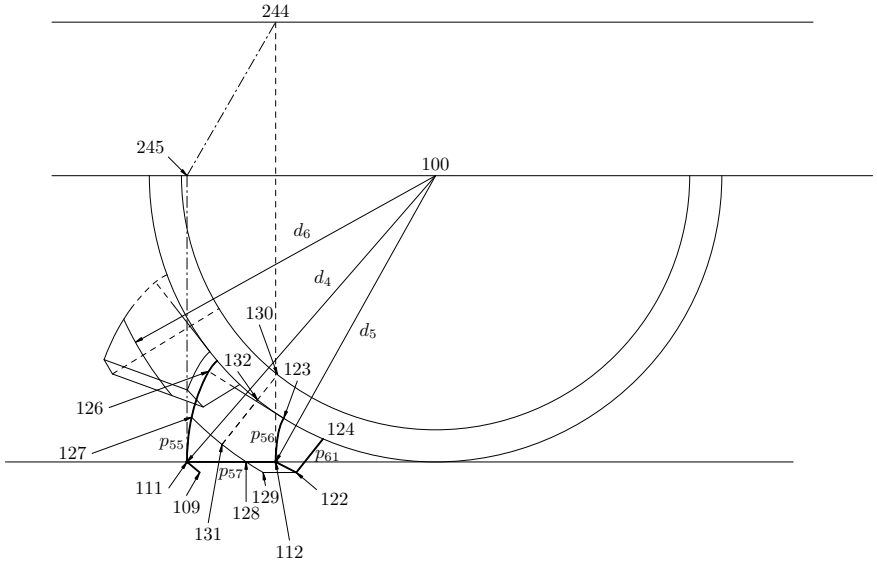


Figure 11: Step 3. Construction of the upper teeth: the lines (123,126) and (112,111) are two of the generating lines of the surface making up the helical teeth.

4.5 Construction of the upper wheel teeth

Figure 10 shows a detail of Olivier's plate. The construction of the upper teeth is shown in figure 11. On this figure, the main helical tooth is bounded by two involute curves, p_{55} and p_{56}. These two curves are obtained as follows.

We know that p_{55} goes through z_{111} and that p_{56} goes through z_{112}. We start with an involute curve p_{51} on the upper wheel. We obtain a point z_{105} on p_{51}, at exactly the distance d_4 between z_{111} and z_{100}. This point z_{105} is then used to rotate p_{51} in such a way that it goes through z_{111}, the new path being p_{52}:

```
p51=involute_tooth(r2,10,20) shifted z100;
d4=arclength(z100--z111);
z105=p51 intersectionpoint circle(z100,d4);
p52=p51 rotatedaround
        (z100,angle(z111-z100)-angle(z105-z100));
```

Similarly, we obtain p_{54} going through z_{112}:

```
p53=involute_tooth(r2,5,20) shifted z100;
d5=arclength(z100--z112);
```

```
z106=p53 intersectionpoint circle(z100,d5);
p54=p53 rotatedaround
        (z100,angle(z112-z100)-angle(z106-z100));
```

The two involutes p_{52} and p_{54} extend below the tangency line, and we cut them so that they end at that line. We now have three edges of the projected tooth: p_{55}, p_{56} and p_{57}.

```
p55=p52 cutafter circle(z100,d4);
p56=p54 cutafter circle(z100,d5);
p57=z111--z112;
```

The other faces of the tooth are non-working, and they can be determined in different ways, taking into account the way they are manufactured and the material used. The tooth thickness is a parameter of the drawing and we determine z_{109} by rotating z_{111} of a certain angle.

```
z109-z100=(z111-z100) rotated 2.5; % empirical
```

z_{122} is obtained as an intersection of a circle going through z_{112} and a line parallel to p_{57} and going through z_{109}. However, although this makes it easy to draw the figure, it produces a face which is not planar. Since Olivier did so in his plate, we reproduced it. In figure 6, however, we made sure the corresponding face is planar.

```
p58=z111--z109;
z110-z112=z109-z111;
z121-z110=z110-z109;
z122=(z109--z121) intersectionpoint circle(z100,d5);
p59=z109--z122;
p60=z112--z122;
```

z_{124} is obtained as the intersection of the base circle with a parallel to (z_{109},z_{100}) going through z_{122}. As a consequence, the face which is opposite the working-face can be made planar.

```
z123=point 0 of p56;
z124=(z122--(z122+(z100-z109)))
     intersectionpoint circle(z100,r2);
p61=z122--z124;
```

And when the wheels turn, the contact segment shifts from the outside to the inside of the tooth, or in the opposite direction. When the contact occurs at the base radius, the contact segment which is involved is the segment (z_{123}, z_{126}), and z_{126} is computed as follows:

```
z126=p55 intersectionpoint
         (z123--(z123+(z100-z123) rotated 90));
p62=z123--z126;
```

We now cut the tips of the helical teeth, in order to limit their projection inside the cylindrical wheel. We compute a radius d_6 based on the value of r_{11} measured on Olivier's plate. p_{63} is the circle for the maximum extent of the helical teeth.

```
r10=arclength(z0--z100);
d6=.99(r10-r11);
p63=circle(z100,d6);
```

Using the previous circle, we cut a part of the helical tooth. For that, we determine three points:

```
z127=p55 intersectionpoint p63;
z128=p57 intersectionpoint p63;
z129=p59 intersectionpoint p63;
```

Then, we define several paths. p_{64} is the arc going from z_{127} to z_{129}. p_{65} is the part of p_{55} that is left once we cut what goes beyond p_{63}. p_{66} and p_{67} are two more edges produced by this cut.

```
p64=the_arc(z127,z100,
           angle(z129-z100)-angle(z127-z100));
p65=p55 cutafter p63;
p66=z128--z112;
p67=z129--z122;
```

The macro `the_arc` is defined as follows:

```
vardef the_arc(expr s,c,a)=
  save p,t;
  path p;
  p=if a<0:reverse fi
    fullcircle rotated (angle(s-c))
      scaled (2arclength(s--c)) shifted c;
  t=xpart(p intersectiontimes
            (c--(c+2(s-c) rotated a)));
  (subpath(0,t) of p)
enddef;
```

Several points are defined for the purpose of drawing the line from z_{100} to the tip z_{109}:

```
z130=(z109--z100) intersectionpoint
                   circle(z100,r21);
% dashed line to the tip of the upper tooth (1):
p68=z109--z130;
z131=(z109--z100) intersectionpoint p63;
z132=(z109--z100) intersectionpoint circle(z100,r2);
% dashed line to the tip of the upper tooth (2):
p69=z131--z132;
```

Finally, we construct a contour for the whole upper wheel (figure 12). This contour is made of the three paths p_{71}, p_{64}, p_{70}, and these paths are repeated to form the contour p_{73}:

```
% construction of a contour for drawing
% the dashed lines of the lower teeth:
p70=p67 cutafter (p55 rotatedaround(z100,(360/nb)));
p71=p65 cutbefore
          (p67 rotatedaround(z100,-(360/nb)));
p72=p64--p70--(p71 rotatedaround(z100,(360/nb)));

% contour to hide the lower teeth:
p73=p72 for i=1 upto nb-1:
  --(p72 rotatedaround(z100,i*(360/nb)))
  endfor--cycle;
```

Four additional paths are defined for the variant teeth shown for the upper wheel in Olivier's plate:

```
p74=p65 cutafter
          (p67 rotatedaround(z100,-(360/nb)));
p75=p55 cutbefore
          (p67 rotatedaround(z100,-(360/nb)));
p76=p62 cutbefore
          (p61 rotatedaround(z100,-(360/nb)));
p77=p62 cutafter
          (p61 rotatedaround(z100,-(360/nb)));
```

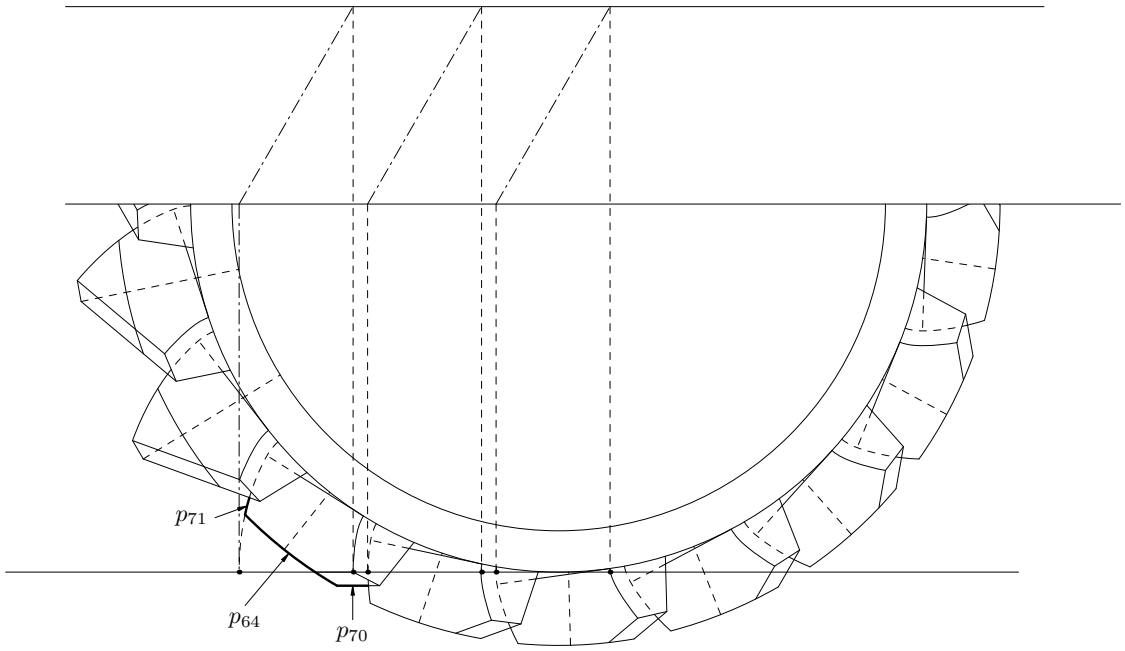


Figure 12: Step 4. Construction of the upper wheel contour.

4.6 Construction of the lower wheel teeth

The lower teeth are easier to draw, since they are purely cylindrical (figure 13). They are positioned in the same way as the involutes for the upper teeth. In other words, the involutes are rotated in such a way that the contact occurs on z_1. The teeth shapes are made of two paths, p_{22} and p_{24}, and the same technique is applied for both. Only the macro used is not the same.

```
p21=cyl_full_tooth(r1,5,2);
d3=arclength(z0--z1);
z14=p21 intersectionpoint circle(origin,d3);
p22=p21 rotated (angle(z1)-angle(z14));
% second part of the tooth:
p23=cyl_full_tooth_x(r1,5,2,1);
p24=p23 rotated (angle(z1)-angle(z14));
```

Finally, the lower teeth are partly hidden by drawing them all and then using the `hiddenpath` macro.

```
def draw_lower_teeth=
  for i=-5 upto 7: % Olivier forgot the case i=7
    draw p22 rotatedaround(origin,i*(360/na));
    draw p24 rotatedaround(origin,i*(360/na));
    % we remove what lies inside p73:
    hiddenpath(p22 rotatedaround(origin,i*(360/na)),
               p73,dashtype(1));
    hiddenpath(p24 rotatedaround(origin,i*(360/na)),
               p73,dashtype(1));
  endfor;
enddef;
```

The `hiddenpath` macro is defined as follows. `hiddenpath(`*under*`,`*over*`)(`*dt*`)` draws that part of the path "*under*" which is within path "*over*" with dashes of type "*dt*".

```
vardef hiddenpath(expr under,over)(text dt)=
  save p,q;
  picture p,q;
  p=image(draw under);clip p to over;
  undraw p;
  q=image(draw under dt);
  clip q to over;
  draw q;
enddef;
```

and the various dashes are obtained with:

```
def dashtype(expr n)=
  if n=0: dashed withdots
  elseif n=1: dashed evenly
  elseif n=2: dashed
    dashpattern(
      on 6bp off 2bp on 1bp off 2bp on 1bp off 2bp)
  elseif n=3: dashed
    dashpattern(on 6bp off 2bp on 1bp off 2bp)
  fi
enddef;
```

5 Drawing the figure

Given all the previous definitions, drawing the figure is pretty straightforward, and we won't describe it in detail. We will merely give an insight into the hatched line LT and the clipping of parts of the drawing.

The hatched line LT is drawn with the macro:

```
vardef hatch(expr p,n,l)=
  save A,B;
  pair A,B;
  A=point 0 of p;B=point 1 of p;
```

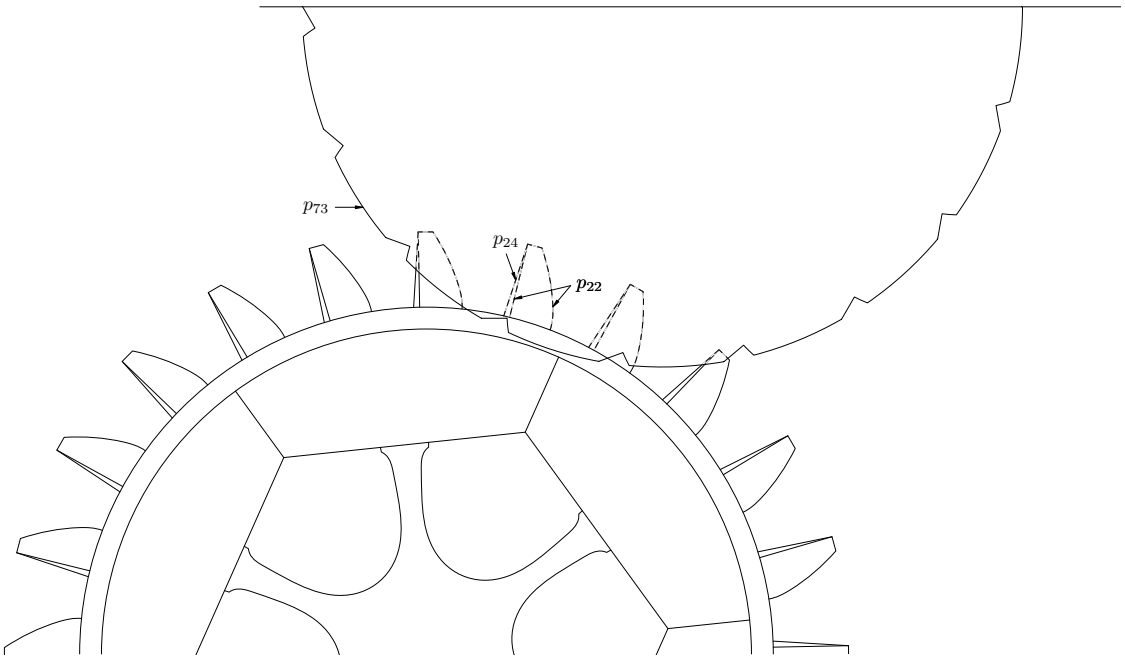


Figure 13: Step 5. Lower wheel teeth construction, and dashing.

```
  for i=0 upto n:
    draw (i/n)[A,B]--((i/n)[A,B]+1*dir(-45));
  endfor;
enddef;
```

and 400 marks are produced with a call to

```
hatch(p0,400,.2u);
```

Clipping the upper wheel may be seen as tricky, especially if parts of the lower wheel have been previously drawn, and we want to retain them. The solution is to save the current picture in a `picture` variable, then to reset the current picture, to draw the upper wheel, then to clip it, and finally to redraw the saved picture. This is summarized in the following macro:

```
vardef draw_upper_wheel=
  save oldpic;
  picture oldpic;
  oldpic=currentpicture;
  currentpicture:=nullpicture;
  draw_upper_teeth;
  draw_upper_wheel_structure;
  clip currentpicture to p102; % we cut beyond p102
  draw oldpic;
enddef;
```

6 Conclusion

We were able to produce a very faithful copy of Olivier's original plate. We think that we identified two errors in Olivier's drawing (or rather in the etching which was made from his drawing). First, the wheel with cylindrical teeth is obviously missing one tooth in Olivier's drawing. Second, the arc between points z_{224} and z_{100} is incorrectly represented.

Other than that, reproducing Olivier's figure has provided an understanding that would be difficult to reach by merely gazing at the drawing. By reproducing it, one is led to find the relationships between the parts, and, in particular, to try to minimize the number of parameters, such that the figure becomes as general as possible.

References

Goossens, Michel, F. Mittelbach, S. Rahtz, D. Roegel, and H. Voß. *The LaTeX Graphics Companion, Second Edition.* Addison-Wesley, 2007.

Olivier, Théodore. *Théorie géométrique des engrenages destinés à transmettre le mouvement de rotation entre deux axes situés ou non situés dans un même plan.* Paris: Bachelier, 1842.

von Seherr-Thoss, Hans Christoph. *Die Entwicklung der Zahnrad-Technik: Zahnformen und Tragfähigkeitsberechnung.* Berlin: Springer-Verlag, 1965.

◇ Denis Roegel
LORIA, BP 239
54506 Vandœuvre-lès-Nancy
FRANCE
roegel (at) loria dot fr
http://www.loria.fr/~roegel

Glisterings

Peter Wilson

> Whose waves do glister by the Queen's
> bright beams.
> Which makes them murmure as they passe
> away.
>
> *Poems and Fancies*, Margaret Cavendish

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.

Corrections, suggestions, and contributions will always be welcome.

In recent times there seems to have been a spate of questions on the `comp.text.tex` newsgroup about controlling paragraphs.

> for life's not a paragraph
> and death I think is no parenthesis.
>
> *since feeling is first*, e e cummings

1 Paragraphs regular

The typical paragraph in a LaTeX document is typeset like:

> The typical paragraph in a LaTeX document looks like this, with the first line indented otherwise lines are set left and right justified except for the last line which is set ragged right.

The indent at the start of a paragraph is given by the length `\parindent`. To temporarily achieve a paragraph with no indentation of the first line put `\noindent` at the start of the paragraph.

The other regular available paragraph shapes are ragged left (flush right), ragged right (flush left) and centered (each line is centered).

On occasions it is useful to be able to set a 'hanging paragraph' where more than one line is indented. TeX provides two commands for specifying such a paragraph (the hanging package [7] provides these in a more LaTeX-like manner).

`\hangindent`⟨*length*⟩ specifies a 'hanging indentation' and `\hangafter`⟨*num*⟩ specifies the number of hung lines. If ⟨*num*⟩ is positive hanging indentation is applied to lines ⟨*num*⟩+1, ⟨*num*⟩+2, ..., while if negative hanging indentation is applied to the first ⟨*num*⟩ lines of the paragraph. When ⟨*length*⟩ is positive the indentation applies to the lefthand end of the lines and when it is negative the righthand ends are indented.

```
\hangindent=3pc\hangafter=-2
```
> Following the above incantation the first two lines of this paragraph are indented at the left by the given amount. You can use these commands in a LaTeX document.

Note that you have to repeat the hanging specification for each hung paragraph.

LaTeX has an internal command `\@hangfrom` that it uses for several purposes, such as the internal code for section titles. An author-friendly version of this is:

```
\makeatletter % unless in a .cls or .sty file
\newcommand*{\hangfrom}[1]{%
  \setbox\@tempboxa\hbox{{#1}}%
  \hangindent \wd\@tempboxa
  \noindent\box\@tempboxa}
\makeatother  % unless in a .cls or .sty file
```

Using `\hangfrom{`⟨*text*⟩`}` at the start of a paragraph produces a paragraph where the second and further lines are indented with respect to the first by the width of ⟨*text*⟩. For instance, the code below produces the result following.

```
\hangfrom{$\Longrightarrow$\space\space}Here
we get a paragraph that is hung in
relation to its first element, which can
sometimes be useful.
```

> ⟹ Here we get a paragraph that is hung in relation to its first element, which can sometimes be useful.

Much more exotically you can get very odd looking paragraphs.

> For more symbolic paragraph shapes like this one, TeX provides the `\parshape` command. For LaTeX users this is provided in a more friendly fashion via Donald Arseneau's shapepar package [1]. Use this kind of paragraph very, very rarely, and only then if you really cannot avoid it as it is a typesetting curiosity. This one has been made using the shapepar package and the `\nutshape` specification.

The shapepar package provides several shape specifications, and there are programs which will automatically generate shape specifications.

Table 1: LaTeX's paragraph settings

	regular	raggedleft	raggedright	centered
\leftskip	\Zeroskip	\Flushglue	\Zeroskip	\Flushglue
\rightskip	\Zeroskip	\Zeroskip	\Flushglue	\Flushglue
\parfillskip	\Flushglue	\Zeroskip	\Zeroskip	\Zeroskip

> For precept must be upon precept; precept upon precept; line upon line; line upon line; here a little, and there a little.
>
> *Isiah*, ch. 28, v 10

2 Paragraphs particular

Besides \hangafter, \hangindent and \parshape, TeX provides 4 parameters[1] for controlling the shape of regular paragraphs. The length \parindent sets the initial indentation of the first line. \leftskip and \rightskip are inserted at the start and end of each line, and \parfillskip is put at the end of the last line; these last three macros are akin to LaTeX's concept of rubber lengths. By changing these you can obtain some particular kinds of paragraph shapes. Assuming that:

```
\Zeroskip = 0pt plus 0pt minus 0pt
\Flushglue = 0pt plus 1fil
```

then LaTeX's settings for its regular paragraph styles are given in table 1.

By adjusting the parameters you can arrange that the middle lines of a paragaph have a particular shape, while the first and/or last lines can have different forms. For instance:

```
\newcommand*{\justlastragged}{%
  \leftskip=0pt plus 1fil
  \rightskip=-\leftskip
  \parfillskip=\leftskip
  \parindent=0pt}
```

Following a \justlastragged declaration paragraph(s) will look like the example below.

> The shape of this paragraph is not too strange. It is flush left and right, except for the last line which is ragged left.

Another particular paragraph shape is one with the lines flush left and right, except for the last which is to be centered, as in:

> The lines in this paragraph, with the normal indentation of the first line, should be flush left and right except for the last line which should be centered.

[1] There are really 5 parameters, the fifth being \everypar which is inserted between the indent and the start of the first line. Change this only if you really know what you are doing.

This can be achieved by using the declaration \centerlastline before the paragraphs(s) in question:

```
\newcommand{\centerlastline}{%
  \leftskip=0pt plus 1fil
  \rightskip=0pt plus -1fil
  \parfillskip=0pt plus 2fil}
```

For no initial indentation put \noindent at the start of the paragraph's text.

```
\newcommand*{\raggedrightthenleft}{%
  \leftskip=0pt plus 1fill
  \rightskip=0pt plus 1fil
  \parfillskip=0pt
  \everypar{\hskip 0pt plus -1fill}%
  \parindent=0pt}
```

After a \raggedrightthenleft declaration the first line of any paragraph will be ragged right and all the other lines will be set ragged left. This looks odd to me.

> This is a strangely shaped paragraph. The paragraph's first line is ragged right and all the remaining lines are ragged left.

One way or another the paragraph layouts depend on the amount of what Knuth terms 'glue' (I think, though, that 'spring' would be a more evocative term) at the start and end of each line (the values of fil). For fuller explanations of glue see the *TeXbook* [5, ch. 12] or, in my view, more accessibly by Victor Eijkhout [3, ch. 8, 16–18].

In the definition of the last macro the \leftskip has a glue (spring) of strength 1fill, which is infinitely stronger than the \rightskip with strength 1fil, so normally a line will get pushed to the right. The last line has a \parfillskip of 0pt, which will not affect the end of the line. The first line of the paragraph has a springiness of -1fill from the \everypar and a springiness of 1fill from the \leftskip, which cancel each other out, leaving the spring of 1fil at the right of the line, and consequently the line gets pushed to the left.

Nikos Platis [6] wanted to ensure that some words at the end of a paragraph were right justified — if there was not enough space on the current line for them then they should be moved to the next

line while leaving the current line ragged right. That is either:

A short line. Text at right

or

A much longer line than the first one.
Text at right

Several solutions were given but the one initially proposed by Knuth in *The TEXbook* [5, p.106] and submitted by Dirk Schlimm turned out to be the most robust in Nikos' tests. In LaTeX terms:

```
\newcommand*{\atright}[1]{{%
  \unskip\nobreak\hfil\penalty50
  \hskip2em\hbox{}\nobreak\hfil#1
  \parfillskip=0pt\finalhyphendemerits=0\par}}
```

and putting `\atright{`⟨*text*⟩`}` at the end of a paragraph ensures that ⟨*text*⟩ is flush right.

Another often occuring request is how to ensure that the last line of a paragraph is 'not too short'.

Following the declaration `\nottooshort`, which I have defined as

```
\newdimen\parabout
\newdimen\about
\about=2em
\newcommand*{\nottooshort}{%
  \parabout=\hsize
  \advance\parabout -\about
  \leftskip=0pt plus 0pt minus 0pt
  \rightskip=\leftskip
  \parfillskip=\parabout minus \parabout
  \parindent=2em}
```

then the last lines of paragraphs will be at least approximately `\about` long.

The last line in this paragraph should not be too short, for a suitable definition of short. 1 2 3 4

The last line in this paragraph should not be too short, for a suitable definition of short. 1 2 3 4 5

With short paragraphs, like the examples, the overall effect might not look as good as you might expect. The situation improves with more lines.

Peter Flynn [4] answered Mark's question posed below by providing code for what he termed a 'spring margin', noting that very few systems provided it.

Hi, I'd like left- and right-justified text on the same line. e.g., some text here... ... and some text over here. I've tried `tabularx` and `TabularC`, but they are not precise enough to line up with the margins. Any suggestions? Thanks, Mark

The above was produced, with appropriate replacements for the ..., by

```
\spring{0.3}{0.6}%
       {Hi, I'd like left- ...}%
       {\dots and some text ...}
```

where `\spring` is a slight extension of Peter's code. The first two arguments are the fractions of the overall line allocated to the left and right texts; the sum of these must be less than 1. The second pair of arguments are the left and right texts.

```
\newcommand{\spring}[4]{%
 \par\noindent\hbox to\columnwidth{\vtop{%
 \hsize=#1\columnwidth\flushleft#3\par}\hss
 \vtop{\hsize=#2\columnwidth\flushright#4\par}}}
```

I have seen legal documents where each line must be filled at the right so that no extra words can be added later.

This last example was created based on the following code.

```
\let\origpar\par
\newcommand*{\parrule}{%
  \hrule height 2.2pt depth -1.8pt\relax}
\newcommand*{\lastlinerule}{%
  \unskip\nobreak\space
  \leaders\parrule\hskip\Flushglue
  \vadjust{}{\parfillskip=0pt\origpar}}
```

If you have many paragraphs of this kind then following a

```
\let\par\lastlinerule
```

all paragraphs will potentially have the last line filled with a rule. Be aware that LaTeX considers many things to be paragraphs so you could be in for some surprises. To revert back to the regular paragraphs specify:

```
\let\par\origpar
```

Alternatively, do something along these lines:

```
\begingroup
\let\par\lastlinerule
A ruled paragraph ...

Another one...

Even more...

\endgroup
```

The end of a paragraph is normally signalled by either a blank line or the `\par` command. For an isolated ruled paragraph, just end the text with `\lastlinerule` instead of `\par` or a blank line.

3 Paragraphs Russian

A while after I had completed this column I was going through old papers, trying to winnow those

that were no longer useful. Doing this I came across an old issue of *Baskerville — The Annals of the UK TEX Users' Group* which included an article about Russian-style paragraphs [2]. Apparently in the Russian typographic tradition the last line of a multiline paragraph must be either at least as long as the `\parindent` and have at least `\parindent` space at the end, or it must be flush left and flush right.

This requirement can't be fulfilled by any simple adjustment of the paragraph setting parameters.

The article ended with two solutions. The first, shown below, was by Peter Schmitt. The basic technique is to end each paragraph by (`glue + hbox + glue`), where the empty `hbox` spans `\parindent`, the (`glue+hbox`) ranges from `\parindent` to (`\hsize - \parindent`) and the (`hbox+glue`) covers (`\hsize-\parindent`) to `\hsize`, where `\hsize` is the line length. According to TEX rules, a linebreak may occur either before the `glue+hbox` or just after the `hbox`, in either case giving the paragraph a final blank line, which has to be backed up over.

```
\def\Srussianpar{\ifhmode \unskip
  \hskip-2\parindent minus -2\parindent
  \hskip\hsize minus\hsize
  \hbox{\hskip\parindent}%
  \hskip0pt \hbox{\strut}%
  \hskip-\parindent
  \hskip\hsize plus\parindent
  \vadjust{\nobreak\vskip-\baselineskip}%
  \parfillskip0pt
  \origpar
  \fi}
```

How this looks in practice is shown below, with `\parindent` set to 2em, together with the definition `\let\par\Srussianpar`.

The last line in this paragraph should conform to the Russian typesetting tradition. 1 2 3 4

The last line in this paragraph should conform to the Russian typesetting tradition. 1 2 3 4 5

The last line in this paragraph should conform to the Russian typesetting tradition. 1 2 3 4 5 6

The last line in this paragraph should conform to the Russian typesetting tradition. 1 2 3 4 5 6 7

The last line in this paragraph should conform to the Russian typesetting tradition. 1 2 3 4 5 6 7 8

The last line in this paragraph should conform to the Russian typesetting tradition. 1 2 3 4 5 6 7 8 9

The last line in this paragraph should conform to the Russian typesetting tradition. 1 2 3 4 5 6 7 8 9 0

The second solution was by Donald Arseneau:

```
\def\Arussianpar{\ifhmode \unskip
  \strut\vadjust{}\nobreak
  \discretionary{}%
    {\hbox{\hskip2\parindent
           \vrule depth 273sp
           width 0sp height \ht\strutbox}}%
    {\hbox{\hskip\parindent}}%
  \hskip-2\parindent minus 2\parindent
  \hskip\hsize minus\hsize
  \kern0pt\parfillskip0pt
  \origpar
  \ifdim\prevdepth=273sp
    \nobreak
    \vskip-2\baselineskip
    \hbox{\strut}%
  \fi
  \fi}
```

This works in approximately the same manner as Peter Schmitt's but it does not always produce an unwanted extra blank line. A rule with a unique depth small enough to be invisible on the page is inserted with the glue items. If the break is such that this is left on the last line, which will be otherwise empty, it can be detected from the `\prevdepth` value and the line backed up.

References

[1] Donald Arseneau. shapepar.sty, 2002. (Available on CTAN in `latex/macros/generic/shapepar`).

[2] David Carlisle and Peter Schmitt. Russian paragraph shapes. *Baskerville*, 6(1):13–15, February 1996.

[3] Victor Eijkhout. *TEX by Topic, A TEXnician's Reference*. Addison-Wesley, 1991. ISBN 0–201–56882–9. (Available at `http://www.eijkhout.net/tbt/`).

[4] Peter Flynn. Re: simultaneous justification in latex. Post to `comp.text.tex` newsgroup, 20 September 2006.

[5] Donald E. Knuth. *The TEXbook*. Addison Wesley, 1984.

[6] Nikos Platis. Justify at right margin or in next line. Post to `comp.text.tex` newsgroup, 21 August 2006.

[7] Peter Wilson. The hanging package, April 2004. (Available on CTAN in `latex/macros/contrib/hanging`).

◇ Peter Wilson
18912 8th Ave. SW
Normandy Park, WA 98166
USA
herries dot press (at) earthlink dot net

The Treasure Chest

This is a list of selected new packages posted to CTAN (`http://ctan.org`) from January–June 30, 2007, with descriptions based on the announcements and edited for brevity.

Entries are listed alphabetically within CTAN directories. A few entries which the editors subjectively believed to be of especially wide interest or otherwise notable are starred; of course, this is not intended to slight the other contributions!

Hopefully this column and its companions will help to make CTAN a more accessible resource to the TEX community. Comments are welcome, as always.

fonts

`*GFS` in `fonts`
: Support for Greek Font Society fonts: Artemisia, Bodoni, Complutum, Didot, Epigrafica, NeoHellenic, Porson, and Solomos.

`kpfonts` in `fonts`
: Full set of new fonts for typesetting text and math, based on URW Palladio.

`linuxlibertine` in `fonts`
: The Linux-Libertine project supports several alphabets (including Latin, Cyrillic, Greek and Hebrew) in a variety of shapes.

`tex-gyre` in `fonts`
: The ongoing TEX Gyre project extends many free fonts from URW and other sources.

graphics

`pdftex.def` in `graphics/pdftex`
: This single-file package provides support for LATEX graphics and color in pdfTEX. It has received many recent updates.

`pst-fractal` in `graphics/pstricks/contrib`
: A PSTricks package for drawing Julia sets, Mandelbrot sets, the Sierpinski triangle.

`pst-qtree` in `graphics/pstricks/contrib`
: A wrapper around PSTricks providing easy syntax for trees.

`sketch` in `graphics`
: Sketch translates 3d scenes expressed in a fairly powerful PSTricks-like language to both PSTricks and Ti*k*Z/PGF.

info

`Math_into_LaTeX-4` in `info/examples`
: A short course to help you get started quickly with LATEX, including detailed instructions on how to install LATEX under Windows or Mac OS X.

`pdf-forms-tuturial` in `info`
: Tutorial on creating PDF forms using `pdflatex`, `hyperref` and `insdljs`.

`sommer` in `info/templates`
: Document templates by Jörg Sommer.

language

`japanese` in `language/japanese`
: Adds functionality of a Japanese option in Babel.

`mnhyphn` in `language/hyphenation`
: Mongolian hyphenation, with `babel` support.

macros/generic

`fenixpar` in `macros/generic`
: Handle `\everypar` and, in general, token registers that need to restore themselves.

macros/latex/contrib

`aeb_tilebg` in `macros/latex/contrib`
: Tile a rectangular graphic as page background.

`animate` in `macros/latex/contrib`
: Creating PDF animations.

`bookest` in `macros/latex/contrib`
: An extension of the `book` class.

`*cleveref` in `macros/latex/contrib`
: Automatic cross-reference formatting, according to label type.

`cmdstring` in `macros/latex/contrib`
: Reliable `\string` on commands.

`colorwav` in `macros/latex/contrib`
: Get an RGB set for a wavelength of visible light.

`dlfltxb` in `macros/latex/contrib`
: Various macros either used for creating *Introduktion til LATEX* or presented in the book as code tips.

`doi` in `macros/latex/contrib`
: Ensure correct hyperlinks to `dx.doi.org`.

`ecv` in `macros/latex/contrib`
: A fancy Curriculum Vitae class.

`*elatex` in `macros/latex/contrib`
: Toolbox of programming facilities geared primarily towards LATEX class and package authors. (Its name is not meant to imply that it patches the LATEX kernel; it does not.)

`fancytooltips` in `macros/latex/contrib`
: Define tooltips with arbitrary TEX or graphics material.

gcard in macros/latex/contrib
Arrange text on a sheet to fold into a greeting card.

glossaries in macros/latex/contrib
Create glossaries and acronym lists.

gmeometric in macros/latex/contrib
Support \geometry of the geometry package inside {document}.

inversepath in macros/latex/contrib
Calculate relative inverse (../) file paths.

jj_game in macros/latex/contrib
A LaTeX class to construct Jeopardy-like games.

leading in macros/latex/contrib
Define leading via a length (cf. \linespread).

memexsupp in macros/latex/contrib
Experimental memoir support.

mlist in macros/latex/contrib
Exploring separation of form and content for math constructs.

oberdiek in macros/latex/contrib
This bundle from Heiko Oberdiek includes many new packages and updates.

philosophersimprint in macros/latex/contrib
Typesetting articles for the online journal *Philosophers' Imprint*.

qstest in macros/latex/contrib
Bundle for unit tests and pattern matching.

sdrt in macros/latex/contrib
Produce the "Box notation" of SDRT and DRT.

simplecv in macros/latex/contrib
Simple class for a curriculum vitae.

songs in macros/latex/contrib
Create beautiful song books for your church or fellowship.

synproof in macros/latex/contrib
Easy drawing of syntactic proofs (a.k.a. derivations) in modern logic.

texilikecover in macros/latex/contrib
Create covers similar to those of GNU Texinfo.

trivfloat in macros/latex/contrib
A simple way to define new float types in LaTeX.

umthesis in macros/latex/contrib
Dissertation class for the University of Michigan.

unroman in macros/latex/contrib
Convert from roman numerals to Arabic numbers.

verbatimcopy in macros/latex/contrib
Small package to perform a verbatim copy of one text file and save it in another.

xfor in macros/latex/contrib
Allow prematurely terminating a \@for loop.

xnewcommand in macros/latex/contrib
\protected or \global macros via \newcommand.

xoptarg in macros/latex/contrib
Expandable macros that take an optional argument.

---

macros/xetex

arabxetex in macros/xetex/latex
An ArabTeX-like interface for Arabic script typesetting with XeTeX.

bidi in macros/xetex/latex
Typesets bidirectional texts with XeTeX.

xgreek in macros/xetex/latex
Typesets Greek using XeTeX, with all the options of the babel Greek support.

---

support

*mkjobtexmf in support
Records and copies files used in a given run, for archiving or speed.

latexpaper in support
Python script to calculate LaTeX settings for arbitrary font and page sizes.

tab4tex in support
Preprocessor for tabular tables.

LaTeX

LaTeX and the different bibliography styles

Federico Garcia

1 Introduction

Although I chose 'style' for the title of this article, it is perhaps best to specify right away that the article is not devoted to styles of formatting entries in a final bibliography list (i.e., a 'style' in BibTeX terms, as defined by the `.bst` file). Rather, we will be looking at the different types of in-text citation: citation by footnote, by parenthesis labels, or by brackets. The citation style admittedly determines certain aspects of the entry-formatting style, but the two things are pretty much independent, and an article such as this one can focus on only one of them.

The twofold thesis of this article is that there are three main citation styles (the ones mentioned: footnote, brackets, and parentheses), and that LaTeX in 2007 provides virtually complete support for all of them. Today (but not five years ago) it is the case that the choice of citation style is not subject to what the software allows, but is really up to the user (within certain limits at least, since institutions — journals, etc. — influence the decision by enforcing one style or another).

In this circumstance, it seems like a good idea to carry out a survey of the three families of bibliographical citation and their support in LaTeX, and that is my purpose here.

The present article stems from a talk I gave at the 2006 Practical TeX conference.

2 The three main style families

It is curious how proponents of each of the three styles — usually — 'don't like' the other styles. They (we) tend to have strong ideas about why one of the styles (our own, usually) is best, and seldom stop to reflect how it is that whole groups of intelligent people have a directly opposite opinion. One thing is for sure: styles are roughly chosen according to discipline. As a result, in our upbringing we are usually exposed to one of the styles far more than the others. We get used to it, and then the others, when we encounter them, do feel a little odd. We conclude, naturally enough, that we don't like them.

But the truth is that there are good reasons why each of the styles exists and is used. In this section I try to make some of these reasons explicit. I won't hide the fact that I lean strongly toward the footnote-style, but I will try to do the others some kind of 'objective' justice.

3 Bracket styles

LaTeX's native support for bibliographical referencing is directed exclusively toward the family of bibliography styles where the citation is done through brackets: something like '[1]' or '[Cas44]'.

This family of styles, most familiar for LaTeX users, has one immediate advantage: the expression in brackets makes sense *both* as a parenthetical comment *and* within the sentence proper. In other words, one can equally make direct reference to a publication (as in 'see [2]' or '[3] is a good reference for...') or simply add the reference as a clarification (as in 'this has already been proven [2]').

Another advantage is that the brackets can be freely used in conjunction with parentheses, so that the form of the actual reference does not depend on the context. It is equally admissible to say 'I once read a book [2] where...' and 'I saw once (in a book that I read [2]) that...'

This efficiency of the brackets is the main reason why this family of bibliography styles needs only one command name: `\cite`. This, and the fact that it is the natural behavior of LaTeX, means that I need to say little more about this style. (In fact, what I have already said was basically for purposes of comparison.)

4 Author-year styles

4.1 Introduction

When the proof of a theorem makes reference to a previously proven theorem, the author and the publication date of the previous paper are not crucial to the argument. Whoever might be interested in that proof in itself can consult the final list of references, and start the search. But for the purposes of the original argument, things like the author, the title, and year of the references are, generally, of no consequence in this kind of discourse.

On the other hand, if an author is referring to previous essays on — say — ethical perception of environmental issues, *then* information on who wrote those previous essays, when they were written, and even what they are called, can be absolutely crucial to the argument. After all, in this context it is not the same to quote a French postmodern philosopher as to quote a study by the Department of Defense.

It is in these contexts that the bracket citation style is truly insufficient. This kind of discipline — let's call them 'the humanities' — has come to adopt widely an alternative kind of citation, gen-

	Some philosophers of science have claimed that science progresses as life
`\citeaffixed`	(notably Kuhn 1996, Koestler 1959). Kuhn started his conceptual trip with his explo-
`\citeyear`	ration of the Copernican Revolution (1957). At virtually the same time, at the other
`\citeasnoun`	side of the Atlantic, the same trip was documented in Koestler (1959). In musicology,
	Leo Treitler expresses very similar views (1984, 1989, 1999). Kuhn's 'paradigms' were
(`\citename...`)	directly addressed by musicians since 1991 (McClary).
	There was and is, of course, opposition to Kuhn, whose ideas were always shunned
	by mainstream philosophy of science. Most virulent of all was the criticism by
	Imre Lakatos (1970). In musicology, this criticism has no direct offspring, but the other
`\cite`	extreme, the 'anarchy of knowledge' (Feyerabend 1978), finds parallels in the diverse
`\possessivecite`	manifestos of postmodernist musicology, for example Tomlinson's (1984).

Figure 1: harvard sample

erally known as author-year citation. It consists basically of embedding some of the crucial information (the author and the year) into the label of the citation: instead of '[1]', one would have '(Cassirer, 1944)'.

Note that one of the native LaTeX bracket styles, `alpha`, is a compromise between the two things — in 'Cas44', Cas is the first three letters of the last name, and 44 is the year. However, even in this case, `alpha` is oriented more towards the sciences than the humanities: what if one cites Nietzsche, who wrote in the 1800s?

In any case, the most relevant difference between this style and the LaTeX default is that — for unknown reasons, I might add — author-year styles use regular parentheses instead of brackets.

This has a wealth of interesting consequences. Parentheses, unlike brackets, have a meaning other than bibliography, and, alas, the two meanings collide. I can say 'this has already been argued (Cassirer, 1944)'. But things like 'for this issue see (Cassirer, 1944)' or '(Cassirer, 1944) is a good reference for...' are funny. Even funnier results are produced by citations within parentheses: "I saw once (in a book I read (Cassirer, 1944)) that..."

Thus, these styles tend to feature a number of variations to the way sources are actually cited, designed to solve the dilemmas of grammar and aesthetics illustrated above. So:

- This has already been argued (Cassirer, 1944).
- For this issue see Cassirer, 1944.
- Cassirer (1944) is a good reference for...
- I read once a book (Cassirer, 1944) where...
- I saw once (in a book I read [Cassirer, 1944]) that...

The choice of the right kind of citation is probably beyond complete automation. That means that it is the user who has to choose. And, in turn, this means that many different commands have to be available. In fact, LaTeX packages that support this family of styles have an unusually large number of citation commands.

4.2 Samples

I will refer to three particular packages, all very successful, that support author-year citation: harvard, achicago, and natbib.

Appearing at the top of this and the following two pages, we show samples of how they work, arranged so that direct comparison is possible. The notes on the left are the commands that create the relevant citations in each of the packages. The citations are underlined to make them more prominent (i.e., the underlining is *not* an effect of the packages themselves). The reader might want to take a moment to glance at these samples and get the feeling of the differences and the similarities between the three packages.

As can be readily seen, translation between the three is pretty straightforward. But it is interesting to see the different command names that the three authors chose for the several citation variants.

4.2.1 harvard

In harvard (the first, seminal one, by Peter Williams and Thorsten Schnier, final version 1994), the naming follows a 'logical' or 'grammar-oriented' model: citations are qualified by the grammatical function of the label in the sentence. When the citation is a noun, you use `\citenoun`; when something has to be affixed to the parenthesis *before* the citation proper, you type `\citeaffixed` (for 'suffixes', additions *after* the citation, the optional argument of `\cite` is used).

	Some philosophers of science have claimed that science progresses as life
`\citeNP`	(notably Kuhn 1996, Koestler 1959). Kuhn started his conceptual trip with his explo-
`\citeyear`	ration of the Copernican Revolution (1957). At virtually the same time, at the other
`\citeN`	side of the Atlantic, the same trip was documented in Koestler (1959). In musicology,
	Leo Treitler expresses very similar views (1984, 1989, 1999). Kuhn's 'paradigms' were
(`\citeA...`)	directly addressed by musicians since 1991 (McClary).
	There was and is, of course, opposition to Kuhn, whose ideas were always shunned
	by mainstream philosophy of science. Most virulent of all was the criticism by
	Imre Lakatos (1970). In musicology, this criticism has no direct offspring, but the other
`\cite`	extreme, the 'anarchy of knowledge' (Feyerabend 1978), finds parallels in the diverse
Tomlinson's `\citeyear`	manifestos of postmodernist musicology, for example Tomlinson's (1984).

Figure 2: achicago sample

4.2.2 achicago

Matt Swift, who wrote achicago (last version 2001), chose a 'form' criterion: the names of his commands follow what it is that the citation needs (the author? the year?), whether or not parentheses should be added (all commands have a `\...NP` version for 'No-Parenthesis'), and so on. The package does not handle pre-citation notes (like the expression 'notably' in the sample) directly, but using these no-parenthesis commands the user can achieve similar effects.

achicago is a full-fledged package with, quite intriguingly, several extra-bibliography odds and ends. Quotations are no longer typeset `\small`, and `\emph` translates not to `\textit` but to `\textsl`. These things can be a little annoying when one is following uses set by someone else (journals, professors, etc.). On the other hand, the BibTeX that accompanies the package (the file `achicago.bst`) is amazingly comprehensive, providing fields for such notions as translator, original title, etc. In his introduction to the package the author enters the discussion of the pros and cons of each family of styles. More about this later.

4.2.3 natbib

The wonderful natbib package (by Patrick Daly, last version 2006) is the definitive word on author-year bibliography styles with LaTeX. It builds on the harvard experience and offers a most complete set of customization possibilities. Extra features include an easy conversion to bracket labels, a useful system of 'aliases', control over punctuation and capitalization, and continued two-way support with packages like hyperref. The older packages harvard and achicago are very dear to me personally, but for users new to this family of styles I see no reason to recommend any package other than natbib.

The commands in natbib are named somewhat more capriciously than in its predecessors. There is no plain `\cite` (!). Instead of this, `\citep` is intended for parenthetical citations and `\citet` for citations within the text (the ones that would be 'noun' citations). Both commands support two optional arguments, for notes within the parentheses to either side of the citation itself.

4.3 Advantages and disadvantages of author-year

Oren Patashnik (creator of BibTeX, and one who clearly doesn't like author-year labels) has even argued that this citation style "encourages the passive voice and vague writing". With Matt Swift (in his introduction to achicago), I have to say I'm not sure. But there is no denying that the parenthetical labels interrupt the flow of reading. The same reasons that in certain contexts make this style better than bracket labels — i.e., that the author and the year *are* crucial information in some kinds of argument — can be held actually *against* it. In these contexts, the title is also crucial: suppose you quote someone like Foucault; is this an interview, a popularizing essay, or a rigorous book? What the reader is to do with the citation certainly depends on this. And in that case, the reader is forced to put his finger in the book, and thus go search for the entry in the final reference list.

Or what about different editions of books, or reprints of articles? What is one to do with a citation like '(Descartes, 1949)'? If information about the publication is important, citations like '[1]' are insufficient, but '(Adorno, 1976)' is insufficient too, and sometimes even misleading.

One might then ask why it is that these styles

	Some philosophers of science have claimed that science progresses as life
`\citep`	(notably Kuhn 1996, Koestler 1959). Kuhn started his conceptual trip with his explo-
`\citeyearpar`	ration of the Copernican Revolution (1957). At virtually the same time, at the other
`\citet`	side of the Atlantic, the same trip was documented in Koestler (1959). In musicology,
	Leo Treitler expresses very similar views (1984, 1989, 1999). Kuhn's 'paradigms' were
(`\citeauthor...`)	directly addressed by musicians since 1991 (McClary).
	There was and is, of course, opposition to Kuhn, whose ideas were always shunned
	by mainstream philosophy of science. Most virulent of all was the criticism by
	Imre Lakatos (1970). In musicology, this criticism has no direct offspring, but the other
`\citep`	extreme, the 'anarchy of knowledge' (Feyerabend 1978), finds parallels in the diverse
Tomlinson's `\citeyearpar`	manifestos of postmodernist musicology, for example Tomlinson's (1984).

Figure 3: natbib sample

are so widely standardized today. Well, there is a clear reason for their early appeal: unlike numeric references, and unlike footnotes, a late change to a manuscript does not require going over the whole thing to update numbers and cross references. This was extremely relevant in typewriter, WordStar or WordPerfect times (I imagine — I'm just too young to have experienced it myself!). It may even be relevant today, taking on the risks of stereotyping, with Word users, whose vast majority is not aware that this can be automated. In any case, this 'advantage' is of course rendered meaningless by the computers of today, and in particular by TEX.

In fact, it is a little ironic that further development of computerized document preparation is even turning this advantage of author-year styles into a hindrance: more and more, citations are expected to be interactive hyperlinks. This, today, implies an *enormous* difference between typing (say, with the natbib package)

```
...(notably Tomlinson's [1984]).
```

and typing

```
...(notably Tomlinson's
[\citeyear{tomlinson1}]).
```

The first, easy to remember and type, *won't produce a link*. If you want the link, and today you certainly do, you have to use the second — and then the effort of taking care of the punctuation, command sequence, and key, seems a little like... like using a tank to kill a fly.

So, beyond the often unsurmountable institutional pressure — journals, professors, etc. — I really see no reason to use author-year styles. Above all today, that — my main point in this article — software has advanced to a point where all alternatives are equally well supported.

5 Footnote citations

Maybe not for pure mathematics, but in other contexts (certainly including the *history* of mathematics) I would say there is no better option than footnote citation. LaTeX has supported this since 2002, with the appearance of opcit. This package will translate `\cite` into `\footnote` (unless it occurs inside one), and append the information of the reference into the footnote.

The first time a publication is cited, the information will be full: author, title, journal/publisher, address, year, etc. Further citations of the same work, however, will abbreviate the reference into the last name, followed by the traditional '*op. cit.*' (Latin for 'cited work'). Moreover, if the same citation occurs in two successive footnotes, it will simply say '*Idem*' ('same'). The optional argument to `\cite` will be appended after the information (either full or abbreviated), separated by a comma.

opcit provides a starred version `\cite*` that omits the author's name (often redundant in footnotes). On the other hand, if there are several works by the same author, in which case '*op. cit.*' can be ambiguous, a mechanism to assign 'aliases' to the works (the 'hereafter' mechanism) is provided.

6 opcit 2

opcit was written by this author, and its first version dates from 2002. In 2006 I uploaded the second version of the package, with a complete BibTeX style (the first one was very limited). This second version, which owes a lot to comments and suggestions by several users, and in particular those of John Scott, fixes minor problems of the first version, and adds some extra features, notably:

- The ability to omit certain information in the

footnotes but not in the final reference list. This can be used to omit an article's page numbers when a '[p. 12]' optional argument follows, or to omit the second part of the title, information on series, original edition dates, etc. — information that is not really needed in the footnotes.
- '*op. cit.*' expressions and other 'aliases' can be hyperlinks to the footnote where the work was first cited.
- Citations can be reset (for example, at the beginning of chapters) so that a post-citation will again cite the information in full.
- Support for cross referencing between entries through BibTeX's field `crossref`.

6.1 Additions to opcit 2

The second version of opcit has been generally well received and, as far as I can judge, widely used. Some users have already made comments and suggestions, and in two cases they have contributed some pieces of code that fix or improve a couple of opcit's current features. These additions, mentioned in this section, will be included in a third release I'm working on (hopefully for the summer of 2007), but for the moment they are in beta testing.

6.1.1 Hereafter improved

Eric Rauchway, a devoted "fan" of opcit, wrote to me some months ago about getting the "hereafter" of articles *not* italicized. ("Hereafter" is the user-defined reference to a previously cited source, that replaces the default *op. cit.* It is useful when there are citations of several works by the same author. It is desirable that articles' hereafters are not italicized, while those of books are.) He and his friend Kevin Bryant have found a solution to this, and I will include their find in a following release. The solution involves modified versions of both `opcit.sty` and `opcit.bst`. If interested, please write to me (`federook@gmail.com`) to get the modified files.

6.1.2 Name-swapping

The second release of opcit swaps the first and last name of authors for the final reference list (so that the footnote says "Ernest Gellner", but the final list says "Gellner, Ernest"). However, in some cases this it is desirable to keep the regular order: for example for Dante Alighieri. (Also, there is a problem when the author is Aristotle, since opcit doesn't really know what to swap, and inserts a spurious floating comma.) Patrick Gardner contributed the following solution "which might be of use to others who are using opcit for ancient and medieval authors" (like he is himself):

```
"\newBibCommand{\SwapNames[2]}{#1 #2}%
{#2\ifx\@empty#1\else, #1\fi}"
write$ newline$
```

This should replace line 946 (the `begin.bib` function) of `opcit.bst`. With this, opcit will handle "Aristotle" correctly, and putting the full name "Dante Alighieri" between braces in the bib file will then prevent name swapping.

6.2 The future of opcit

The main problem still facing opcit is a very hard-to-understand (for me, anyway) conflict with endnote, the package that collects the notes to be printed at the end of the document/chapter. It really would be nice to be able to turn endnotes on and off without further changes. (The fascinating discussion on footnotes-or-endnotes resembles that of the bibliography styles in that the opposing sides really hate each other; again, both have good arguments to their cause, but 'the truth' probably lies in a context-dependent approach.) I succeeded once in creating a list of endnotes from opcit footnotes, but the solution was far from robust, and did not really throw light on how to address the problem.

On the other hand, there are ideas and work going on regarding other compatibility issues of opcit. With the release of the second version, the package secures LaTeX support for footnote-style bibliography... *in English*. But use with other languages is not directly implemented. This not only requires the modification of the BibTeX style (so that particles like 'in', 'chapter', etc. are translated), but also might bring about problems with babel. For example, José Luis Rivera from Mexico has identified conflicts with the latter's spanish option, and has started working on complementing opcit with a Spanish BibTeX style, which possibly involves some tweaking to opcit itself.

The implementation of opcit in languages other than English will hopefully involve other users as well, and is, as I see it, the most important future extension of the package.

7 Other important things to mention

7.1 Some hybrid approaches

For the sake of completeness, a couple of packages should be mentioned that provide a kind of 'bridge' between the three main families of styles:

alpha was already mentioned to be a compromise between labels like '[1]' and labels like '(Cassirer, 1944)': it gives '[Cas44]'. See page 236. In the same vein, natbib has the option of typesetting labels in either of the two forms (and

also as superscripts).

cite makes bracket labels appear as superscripts, almost as footnote marks (although between [and] and without an actual footnote). The package (which also has other nice features) is extremely sophisticated, but has almost no documentation (it dates from before the doc package for LaTeX documentation). As a result, it has come to be, in effect, obsolete. Even Sebastian Rahtz, when trying to provide support for it in hyperref, had to give up trying to understand it.

footbib goes one step further than cite: the superscripted labels do actually point to a footnote. However, it is not a footnote in the full sense: it follows its own numbering, and in case there are also 'regular' footnotes in the page, the two sets are separated from each other.

7.2 custom-bib

This package is not directly related to the thesis of this article, but it does seem odd to omit it from a general discussion of the possibilities of bibliography in LaTeX.

custom-bib — the latest version at this writing is dated April 27th of 2007 — is another wonderfully ingenious TeX program by Patrick Daly (the author of natbib) that helps the user create a totally customized BibTeX style (i.e., a `.bst` file). Here we are back to the normal meaning of 'style': the set of rules that govern the appearance of the entries in the final reference list — whether the title is italicized, the journal number typeset in boldface, etc.

The package works in a straightforward way. Once there is a `makebst.tex` file in the system — you might have to create it by running TeX on the file `makebst.ins` — the user runs TeX or LaTeX on it: `latex makebst.tex`

Then the program will simply *ask* (!) how you want to format your entries, and from it create a BibTeX style. It is a truly amazing use of TeX's interactive capabilities, which are usually overlooked (since interactivity is not exactly what document preparation is about, after all).

The package is tailored towards the first two style-families described above: brackets (called 'numerical' in custom-bib) and author-year. Use for opcit, I anticipate, would require some extra hacking on the `.djb` file (an intermediate step between `makebst.tex` and the final `.bst` file), but in principle the main difficulties here would arise from the lack of documentation in opcit about custom-bib, and maybe the other way around as well. That is, opcit includes some directions on how to customize the `.bst` file, but these directions assume familiarity with BibTeX's programming language — and this familiarity is precisely what custom-bib is supposed to spare the user.

In any case, I am a newcomer to custom-bib, so not the most qualified to discuss these matters deeply. The package is mentioned here as the wonderful tool it is for deeper-than-surface bibliography handling in LaTeX.

8 Conclusions

I have to finish by pointing out some facts that have come to my attention since I presented a version of this article at PracTeX 2006. For example, José Luis Rivera told me that the Modern Language Association, which is in effect the main legislator (and champion) of author-year styles, has indeed addressed the issue of some funny things like citing '(Aristotle, ca. −340)'. They allow a variation of style that can be called 'author-title': (Aristotle, *Nicomachaean Ethics*). Even for modern authors, as in '(Derrida, Postcards)', this has been accepted and even encouraged. This certainly is a response to the problem that often the title of a work is more crucial than the year.

The discussion can go on and on (is it the title itself, or the fact that the work is so well-known? how does this depend, once again, on context?), but one thing that has to be said is that, as far as I know, LaTeX has not seen any direct efforts in this direction. The MLA, certainly, has adopted this relatively recently (mid-90s is José Luis's recollection).

I cannot claim that the discussion in the previous pages is comprehensive or complete. However, I hope that the points raised above are not obvious or trivial (they weren't to me when I started thinking of all this), and feel that the topic is interesting, if nothing else because it shows how, here too, one little change or decision leads to more and more. In fact, people who have seen drafts of this article tend to respond rather quickly and very 'personally' (as in "personally, I hate footnotes", or "I *totally* agree with this or that..."). The topic, touching on uses that habit has ingrained to the point that they enter the realm of taste, seems to reach everybody and raise very deep opinions in them. So, besides the arguably 'useful' fact that this article might make readers aware of possibilities that were previously unknown to them, I hope it has also provided some enjoyment.

◇ Federico Garcia
http://www.fedegarcia.net
federook (at) gmail dot com

Font selection in LaTeX: The most frequently asked questions

Walter Schmidt

Abstract

This article tries to answer the three most popular questions regarding font selection in LaTeX, primarily providing guidance through the existing documentation.

1 Basic commands

Having read any LaTeX introduction of your choice (for instance, [1]), you should be familiar with the basic commands for font selection. To start with, let's summarize them once again.

These declarations let you choose among three pre-defined font families:

`\rmfamily`	roman (i.e., serifed)
`\sffamily`	sans serif
`\ttfamily`	monospaced ("typewriter")

Within each font family, the following declarations select the "series" (i.e., darkness or stroke width),

`\mdseries`	regular
`\bfseries`	**bold**

and the "shape" (i.e., the form of the letters):

`\upshape`	upright
`\slshape`	*slanted*
`\itshape`	*italic*
`\scshape`	Caps and Small Caps

These commands are "declarations", i.e., they remain in effect until the end of the current group or environment. For each declaration there exists a text-generating command as a counterpart; it typesets only its argument in the desired style; for instance, `\textsf` corresponds to `\sffamily`. See, e.g., [1], chapter 3.1 and appendix C.15.1.

Family, series and shape can be combined, e.g., `\bfseries\itshape` results in ***bold italic*** type. Notice, however, that not every possible combination is required to exist; for instance, many font families are lacking small caps.

This scheme is called NFSS (New Font Selection Scheme), and its official documentation [2] is available in every LaTeX system as a DVI or PDF document `fntguide.dvi` or `.pdf`.

2 How can I change the default fonts for the whole document?

Most likely, you will already recognize the three default font families used by LaTeX:

roman:	Computer Modern Roman
sans serif:	Computer Modern Sans Serif
monospaced:	`Computer Modern Typewriter`

Outside the world of TeX, these font families are far from popular, so the question asked in the title of the present section is perhaps the "top of the FAQs".

The families selected by `\rmfamily`, `\sffamily` and `\ttfamily` are determined by the corresponding macros `\rmdefault`, `\sfdefault` and `\ttdefault`. You can use the well-known `\renewcommand` to alter them—provided that you know the name of the desired font family. For instance, try adding:

```
\renewcommand{\rmdefault}{ptm}
```

to the preamble of a document. `ptm` is the name under which the font family "Times" is installed in your LaTeX system, so all (roman) text in your example document should change from CM Roman to Times. (Most likely, you will now ask the question how to learn the name of a font family: please, be patient, it will be answered in the next section.)

If, however, there is any piece of maths in your example, you will notice that changing `\rmdefault` does *not* affect formulas. In the above case, they will still be typeset using the CM math fonts, which do not blend well with Times.

Changing the math fonts requires more effort than simply redefining a few macros. That's why alternative math fonts are usually accompanied by a *macro package*: loading the package makes all changes needed to replace the default (CM) math fonts; in many cases these packages take care of redefining `\rmdefault` appropriately, too. For instance, to change both text and formulas to Times, you would in fact add the following line to your document preamble, rather than the one given above:

```
\usepackage{mathptmx}
```

There are also macro packages that change only one of the text fonts, but provide additional features such as scaling.

This raises a few questions: You need to know *which* alternative fonts besides Computer Modern are available in your LaTeX system at all, you need to know the "LaTeX names" of the font families you want to use, and you need to know if there are any related macro packages available. These issues lead us to the next section:

3 Which font families are available in my LaTeX system?

There is a minimum set of alternative fonts that must *always* be available besides Computer Mod-

Editor's note: Reprinted from *The PracTeX Journal* 2006-1 (`http://tug.org/pracjourn`), by permission.

ern; the related collection of macro packages is often referred to as the "PSNFSS collection". In particular, it supports the popular typefaces Times, Helvetica, Palatino and Charter (and a few others), and it supports math fonts that suit Times and Palatino. The related documentation [3] is available in every LaTeX distribution as a PDF file named `psnfss2e.pdf`, usually in the directory `doc/latex/psnfss`. Reading this document is *strongly* recommended. It tells the 'family names' of the supported fonts (such as `ptm` above), and explains the usage of the related macro packages. (See [3], tables 1 and 3.)

Everything that goes beyond the PSNFSS collection is, strictly speaking, optional; i.e., only the documentation of your particular TeX distribution can tell you which fonts are shipped with the system, and where the related documentation is installed.

Most contemporary TeX distributions provide (almost) all free text and math fonts that are available from CTAN; a good overview of the most popular ones is given in chapter 7 of the *LaTeX Companion* [4].

Further font families that have been made available for use with LaTeX are summarized in [5].

4 How can I change the fonts to be used for certain parts of the document?

A popular request is to customize the style of certain elements of the document; for example, the font used in section headings and/or captions. The style of these elements, including the font choice, is determined by the document class you are using. Unfortunately, the standard classes (article, report, book) do not by default provide any means for this kind of customization.

One solution is to use extra packages that add the required functionality; the most popular ones are the packages `titlesec` [6] and `sectsty` [7] to change the style of the section headings, and `caption` [8] to control the style of the captions of figures and tables. Use of these packages is described in detail in the related documentation.

Newer, alternative document classes often go a different way. The KOMA-script classes [9] as well as the Memoir class [10] provide various means for customization. As an example, let's take a look at the interface of the KOMA classes to control the style of section headings:

By default, the KOMA classes use the bold series of the sans-serif font family to typeset headings. To change this, the command

```
\setkomafont{sectioning}{...}
```

is provided. Its second argument is to contain all declarations to be applied when the section headings are typeset. A frequent requirement is to use the bold series of the roman font family instead (as in the standard classes), and additionally to apply `\boldmath`, so that mathematical elements in section headings are emboldened, too. Doing this is straightforward with the KOMA classes:

```
\setkomafont{sectioning}
      {\rmfamily\bfseries\boldmath}
```

Analogously, the style of captions can be controlled via the command `\setkomafont{caption}{...}`.

Typically, with these extra packages and classes it becomes a matter of a single line of LaTeX commands to change the formatting of many parts of the document.

5 Conclusion

At first sight, font selection in LaTeX looks like a relatively complex issue, because it differs fundamentally from font handling in "classical" DTP programs. Yet, it isn't hard when you look in the documentation, and in this article we have tried to show some concise pointers.

References

[1] Leslie Lamport: *LaTeX: A Document Preparation System*, 2nd edition. Addison Wesley, 1994.

[2] LaTeX3 Project Team (Ed.): LaTeX 2_ε font selection. Part of the LaTeX online documentation, file `fntguide.pdf`

[3] Walter Schmidt: Using common Postscript fonts with LaTeX. Part of the LaTeX online documentation, file `psnfss2e.pdf`

[4] Frank Mittelbach et al.: *The LaTeX Companion*, 2nd edition. Addison Wesley, 2004.

[5] TeX Users Group: Fonts and TeX. `http://tug.org/fonts`

[6] Javier Bezos: The titlesec and titletoc packages. `http://ctan.org/tex-archive/macros/latex/contrib/titlesec/`

[7] Rowland McDonnell: The sectsty package. `http://ctan.org/tex-archive/macros/latex/contrib/sectsty/`

[8] Axel Sommerfeldt: The caption package. `http://ctan.org/tex-archive/macros/latex/contrib/caption/`

[9] Markus Kohm: The KOMA-script bundle. `http://ctan.org/tex-archive/macros/latex/contrib/koma-script/`

[10] Peter Wilson: The Memoir class. `http://ctan.org/tex-archive/macros/latex/contrib/memoir/`

The memoir class

Peter Wilson

Abstract

The memoir class is essentially the book and report classes with lots of bells and whistles as it includes the functions of over 30 popularly used packages. It can also simulate the appearance of article class documents and provides a basis for producing the typewritten-like manuscripts which some publishers ask for.

1 Introduction

For nearly 20 years I was involved in using LaTeX to produce camera-ready copy of International Standards, in particular ISO 10303 (STEP). As the standard grew — it now consists of thousands of pages spread across many publications — and ISO and various committees kept changing their minds about what they wanted, I grew increasingly frustrated with having to keep modifying the internals of the class and packages that I had developed. Separately I became interested in book design and felt that there was a need for a class that would support layout experiments. I worked on this in a desultory fashion for several years and eventually produced something that I felt might be generally useful.

The memoir class was first released in 2001 and since then has proven to be reasonably popular. The class can be used as a replacement for the book and report classes, by default generating documents virtually indistinguishable from ones produced by those classes. The class includes options to produce documents with other appearances; for example an article class look or one that looks as though the document was produced on a typewriter with a single font, double spacing, no hyphenation, and so on. In the following I use the term 'standard classes' to denote the book and report classes and, when appropriate, the article class as well.

The memoir class includes the functionality of many packages, for instance the tocloft package for controlling the table of contents and methods similar to the fancyhdr package for designing your own headers. The built-in package functions are mainly related to document design and layout; memoir does not touch upon areas like those covered by the babel or hyperref packages or any related to typesetting mathematics. On the other hand it is easy to configure a work produced with memoir to meet a university's thesis layout requirements.

memoir has improved substantially since it was first released — over 50 LaTeXers have provided code or suggestions for improvements. The class is included in the TeX user groups' TeX distributions and the latest version of the class and its supporting documentation is always available from CTAN at `latex/contrib/memoir`.

2 General considerations

The class is a large one consisting of about 10,000 lines of LaTeX code documented in a 400 page report; there is no need for most users to look at this [4]. There is a separate comprehensive User Manual [3] which runs to about 300 pages and from time to time an Addendum [5] is released noting extensions to the class; at the moment this runs to about 70 pages. There is also the memexsupp package by Lars Madsen [2] which provides some extra facilities for the class.

Altogether the documentation for memoir runs to some 800 pages and it is impossible to cover everything in a short, or even a long, article.

The first part of the Manual discusses some aspects of book design and typography in general, something that I haven't come across in the usual LaTeX books and manuals. This is intended to provide a little background for when you design your own printed documents.

memoir provides slightly enhanced facilities for creating title pages but in my view it is better to create your own layout for a title page. To aid in this I have created some 25 examples of title pages that can be used as a starting point for your design [6]. These were produced using regular LaTeX facilities and are not dependent on memoir.

The standard classes provide point options of 10, 11, or 12 points for the main body font. memoir extends this by also providing 9, 14, and 17 point options. The width of the text block is automatically adjusted according to the selected point size to try and keep within generally accepted typographical limits for line lengths; you can override this if you wish. The class also provides easy methods for specifying the page layout parameters such as the margins — both the side margins and those at the top and bottom of the page; the methods are similar to those of the geometry package.

The page layout facilities also include methods, like those provided by the fancyhdr package, for defining your own header and footer styles, and you can have as many different ones as you wish. In fact the class provides seven styles to choose from before you have to resort to creating your own. The styles are all displayed in the Manual.

Sometimes it is useful, or even required, to place trimming marks on each page showing the desired

size of the final page with respect to the sheet of paper that is used in the printer. This is provided by the showtrims option. A variety of trim marks are provided and you can define your own if you need some other kind.

2.1 Sectioning styles

Handles are provided for designing and using your own styles for chapter titles and such. The class comes with over 20 predefined chapter styles ranging from the default look to a style that mimics that used in the *Companion* series of LaTeX books. There is even one which uses words instead of numerals for chapter numbers. The Manual shows examples of at least six of these styles and about 30 are shown in Lars Madsen's collection [1].

For those who like putting quotations near chapter titles the `epigraph` environment can be used.

The options for changing `\section` and lower level titles are more constrained, but generally speaking document design, unless for advertisements or other eye-catching ephemera, should be constrained.

Sometimes, but particularly in novels, a sectional division is indicated by just leaving a blank line or two between a pair of paragraphs, or there might be some decorative item like three or four asterisks. Commands are available for typesetting such anonymous divisions.

In the standard classes, sectioning commands have an optional argument which can be used to put a short version of the section title into the table of contents and the page header. memoir extends this with a second optional argument so you can specify one short version for the contents and an even shorter one for page headers where space is at a premium.

2.2 Captions

memoir incorporates the code from my ccaption package which lets you easily modify the appearance of figure and table captions; bilingual captions are available if required, as are captions placed at the side of a figure or table or continuation captions from, say, one illustration to another. Captioning can also be applied to 'non-floating' illustrations or as legends (i.e., unnumbered captions) to the regular floats. The captioning system also supports subfigures and subtables along the lines of the subfig package, plus letting you define your own new kinds of floats together with the corresponding 'List of . . . '.

3 Tables

Code from the array, dcolumn, delarray and tabularx packages is integrated within the class. To improve the appearance of rules in tabular material the booktabs package is also included.

Multipage tabulations are often set with the longtable or xtab packages, which can of course be used with the class. For simple tabulations that may continue from one page to the next, memoir offers a 'continuous tabular' environment. This doesn't have all the flexibility provided by the packages but can often serve instead of using them.

More interestingly, but more limited, the class provides 'automatic tabulars'. For these you provide a list of simple entries, like a set of names, and a number of columns and the entries are automatically put into the appropriate column. You choose whether the entries should be added row-by-row, like this with the `\autorows` command:

```
\autorows{c}{5}{l}{one, two, three, four,
    five, six, seven, eight, nine, ten,
    eleven, twelve, thirteen }
```

one	two	three	four	five
six	seven	eight	nine	ten
eleven	twelve	thirteen		

Or if you use the `\autocols` command the entries are listed column-by-column, like this:

```
\autocols{c}{5}{l}{one, two, three, four,
    five, six, seven, eight, nine, ten,
    eleven, twelve, thirteen }
```

one	four	seven	ten	thirteen
two	five	eight	eleven	
three	six	nine	twelve	

4 Verse

The standard classes provide a very simple `verse` environment for typesetting poetry. This is greatly extended in memoir. For example in the standard classes the verse stanzas are at a fixed indentation from the left margin whereas memoir lets you control the amount of indentation so that you can make a poem appear optically centered within the textwidth.

Stanzas may be numbered, as can individual lines within a poem. There is a special environment for stanzas where lines are alternately indented. Also you can define an indentation pattern for stanzas when this is not regular as, for example, in a limerick where the 3rd and 4th of the five lines are indented with respect to the other three as shown below.

```
\indentpattern{00110}
\begin{verse}
\begin{patverse}
There was a young man of Quebec \\
```

```
Who was frozen in snow to his neck. \\
When asked: ‘Are you friz?’ \\
He replied: ‘Yes, I is, \\
But we don’t call this cold in Quebec.’
\end{patverse}
\end{verse}
```

> There was a young man of Quebec
> Who was frozen in snow to his neck.
> When asked: ‘Are you friz?’
> He replied: ‘Yes, I is,
> But we don’t call this cold in Quebec.’

It is not always possible to fit a line into the available space and you can specify the particular indentation to be used when a ‘logical’ verse line spills over the available textwidth, thus forming two or more typeset ‘physical’ lines. On other occasions where there are two half lines the poet might want the second half line to start where the first one finished, like this:

```
\begin{verse}
Come away with me. \\
\vinphantom{Come away with me.} Impossible!
\end{verse}
```

> Come away with me.
> Impossible!

5 End matter

Normally appendices come after the main body of a book. The class provides various methods for introducing appendices at the end, or you can place one or more appendices at the end of selected chapters if that suits you better.

memoir also lets you have more than one index and an index can be set in either the normal double column style or as a single column which would be more appropriate, say, for an index of first lines in a book of poetry. The titles of any bibliography or indexes are added to the table of contents, but you can prevent this if you wish.

The class provides a set of tools for making glossaries or lists of symbols, the appearance of which can, of course, be easily altered. The `makeindex` program is used to sort the entries. An example is shown in the current version of the Addendum. A recent addition to the class provides configurable end notes which can be used as well as, or instead of, footnotes.

6 Miscellaneous

As already noted, the Manual for memoir runs to some 300 pages and we cannot cover everything in a short article. Suffice it to say that hooks and macros are provided for most aspects of document layout; for instance, footnotes can be as normal, typeset in two or three columns, or all run into a single paragraph. There is a `\sidenote` macro which is a non-floating `\marginpar` as well as the `\sidebar` macro for typesetting sidebars in the margin, starting at the top of the text block. You can create new verbatim-like environments, read and write information in external files, design your own style of `\maketitle`, convert numbers to words, reserve space at the bottom of a page, and so on and so forth.

7 Packages

Most packages work with the memoir class, the main exception being the hyperref package. This package modifies many of the internals of the standard classes but does not cater for all of the differences between memoir and the standard ones. If you wish to use hyperref with memoir then you must use the memhfixc package[1] after using hyperref. For example:

```
\documentclass[...]{memoir}
...
\usepackage[...]{hyperref}
\usepackage{memhfixc}
...
\begin{document}
```

However, if you have a version of hyperref dated 2006/11/15 or after, hyperref will automatically call in memhfixc so that you don’t have to do anything.

The memoir class includes code either equivalent to, or extensions of, the following packages; that is, the set of commands and environments provided by memoir is at least that of these packages: abstract, appendix, array, booktabs, ccaption, chngcntr, chngpage, dcolumn, delarray, enumerate, epigraph, framed, ifmtarg, ifpdf, index, makeidx, moreverb, needspace, newfile, nextpage, parskip, patchcmd, setspace, shortvrb, showidx, tabularx, titleref, titling, tocbibind, tocloft, verbatim, verse.

The memoir class ignores any `\usepackage` or `\RequirePackage` related to these. However, if you want to specifically use one of these packages rather than the integrated version then you can do so. For argument’s sake, suppose you really want to use the titling package; you can do this:

```
\documentclass[...]{memoir}
\DisemulatePackage{titling}
\usepackage{titling}
```

The memoir class incorporates a version of the setspace package, albeit using different names for the

[1] memhfixc is supplied as part of the memoir distribution.

macros. The package enables documents to be set double spaced but leaves some document elements, like captions for example, single spaced. To do this it has to make some assumptions about how the document class works. I felt that this kind of capability should be part of the class and not depend on assumptions. In the particular case of the setspace package, even with the standard classes, there can be some unexpected spacing around displayed material; this has not occurred with memoir's implementation.

The class also provides functionality similar to those provided by the following packages, although the commands are different: crop, fancyhdr, geometry, sidecap, subfigure, titlesec. You can use these packages if you wish, or just use the capabilities of the memoir class.

Sometimes a class or package may define a command that is also, differently, defined by a succeeding package. As an example, assume that you want to use the memoir class together with the pack package but they have both defined `\amacro`. There are several ways of dealing with this.

1. Discard the class's definition:

   ```
   \documentclass[...]{memoir}
   % kill the class definition
   \let\amacro\undefined% or \relax
   \usepackage{pack}
   ```

   and pack's version of `\amacro` is used from now on.

2. Discard the package's definition:

   ```
   \documentclass[...]{memoir}
   % save the class definition
   \let\memamacro\amacro
   \let\amacro\undefined
   \usepackage{pack}
   % restore the class definition
   \let\amacro\memamacro
   ```

   and memoir's version of `\amacro` is used from now on.

3. Keep both definitions:

   ```
   \documentclass[...]{memoir}
   \let\memamacro\amacro
   \let\amacro\undefined
   \usepackage{pack}
   ```

   and after this use `\memamacro` for memoir's version and `\amacro` for pack's definition. But this solution doesn't always work, as you might not know when the particular versions must be used, or it is impossible to partition the uses.

A last resort is to ask the authors that one or the other macro names be changed; however, for good reasons, neither may be willing to do this.

References

[1] Lars Madsen. Various chapter styles for the memoir class, July 2006. http://mirror.ctan.org/info/latex-samples/MemoirChapStyles/MemoirChapStyles.pdf.

[2] Lars Madsen. The Memoir Experimental Support Package, February 2007. http://mirror.ctan.org/latex/macros/contrib/memexsupp.

[3] Peter Wilson. The Memoir Class for Configurable Typesetting: User Guide, 2004. http://mirror.ctan.org/latex/macros/contrib/memoir/memman.pdf.

[4] Peter Wilson. The LaTeX memoir class for configurable book typesetting: Source code, 2005. http://mirror.ctan.org/latex/macros/contrib/memoir/memoir.pdf.

[5] Peter Wilson. Addendum: The Memoir Class for Configurable Typesetting: User Guide, 2007. http://mirror.ctan.org/latex/macros/contrib/memoir/memmanadd.pdf.

[6] Peter Wilson. Some Examples of Title Pages, 2007. http://mirror.ctan.org/info/latex-samples/titlepages.pdf.

◇ Peter Wilson
18912 8th Ave. SW
Normandy Park, WA 98166
USA
herries dot press (at)
earthlink dot net

Enjoying babel

Enrico Gregorio

1 Introduction

Back in the 1980s, when TEX was making its way in the world, it was an all-American piece of software. LATEX was based on Plain TEX and was even more American in style.

For instance, Knuth chose to set the DVI reference point one inch to the right and one inch from the top of the sheet of paper; maybe this is one of the design errors in the TEX family of programs. However, with a judicious setting of `\hoffset` and `\voffset`, users could correctly print TEX output on A4 paper. He did provide tools for typesetting European languages (with all their strange accents) but it was not possible to hyphenate two languages simultaneously.

Overall, however, the situation was not so nice for us Europeans. As of today, the European Union comprises 27 countries and has 22 official languages (in three different alphabets), not counting Luxemburgish and various languages spoken by minorities: in the UK, besides English, there are Scottish Gaelic, Scots, Scottish English, Welsh, Irish, Cornish and Manx; in Spain, besides *Castellano*, there are *Catalá* (Catalan, in three different varieties), *Galego* (Galician) and *Euskara* (Basque) plus some others. There are many countries where two or more languages have official status, possibly only in some regions: this is the case of Italy, where German and French are official languages in two provinces and Slovenian is "almost official" in one province.

Version 3 of TEX was hailed with enthusiasm, as it provided the possibility of hyphenating in 256 languages simultaneously and its 8 bit design allowed for extended sets of characters which made it possible to get rid of explicit accents, with all the related and well known hyphenation problems: in fact TEX does not hyphenate a word containing an explicit accent (past the accent), which is a big nuisance for languages such as French and German and is intolerable for Slavic languages such as Czech and Polish. By the way: do you know the difference between *slovenčina* and *slovenščina*?[1]

Earlier than the introduction of TEX 3, Johannes Braams developed the babel system that permitted substituting the fixed tags in LATEX like 'Chapter' and 'Table of Contents' with localized tags for some European languages. It also provided a method, based on the package german by Bernd Raichle, for inputting accented characters while allowing for good hyphenation.

Of course, before TEX 3, users were limited to hyphenating one language at a time, and special versions of the LATEX (or Plain or $\mathcal{AMS}$-TEX) format had to be prepared. But, at least, one could typeset a book in Italian where chapters were named 'Capitolo' and the table of contents 'Indice'.

LATEX 2_ε improved the situation. It supported package options and the support for babel was integrated by declaring control sequences that contain the fixed tags: for example, `\chaptername` expands to 'Chapter' by default, but babel can easily change its meaning in every language it supports.

The supported languages are many, 44 in the current version, and not only European. It may be surprising to learn that at least as many European languages are *not* supported. Among the main ones, Maltese, Lithuanian and Latvian still lack support (they are all official in the EU); one of the four official languages of Switzerland, Romansh, is missing. But Latin, Esperanto and Interlingua are present.

I should mention Thomas Esser and his teTEX distribution, which made it easy to enable hyphenation rules and format creation for LATEX. The same idea was used by MiKTEX through a menu. TEX Live, also based on the teTEX scripts, offers this facility as well. Moreover, today's fast computers and large memories make it possible to enable all available rules and then forget about the matter.

I'll talk later briefly about Plain TEX or $\mathcal{AMS}$-TEX users, who are not left in the cold, after all. But the bulk of the paper is devoted to babel and LATEX.

2 Calling babel

The babel package is called as usual:

`\usepackage[`⟨*languages*⟩`]{babel}`

where ⟨*languages*⟩ is a comma separated list of languages, whose names can be found in Table 1. You should name all the languages you plan to use in the document, for example

```
\usepackage[italian,english]{babel}
```

if the document has English as its main language, but some parts of it are written in Italian.

I said that the supported languages are 44, but the table has more items. Some names are just synonyms (Hungarian and Magyar, for example) and some denote *dialects*, that is, languages which share hyphenation patterns with others (for example, Austrian is a dialect of German, Acadian and Canadien

[1] *Slovenčina*, or *slovenský jazyk* is the official language of Slovakia. *Slovenščina*, or *slovenski jezik*, is the official language of Slovenia. They are two different countries of the EU and do not share a border.

Table 1: List of babel languages

acadian	bulgarian	frenchb	lowersorbian	russian
afrikaans	canadian	galician	magyar	samin
albanian	canadien	german	malay	scottish
american	catalan	germanb	meyalu	serbian
australian	croatian	greek	naustrian	slovak
austrian	czech	hebrew	newzealand	slovene
bahasa	danish	hungarian	ngerman	spanish
bahasai	dutch	icelandic	norsk	swedish
bahasam	english	indon	nynorsk	turkish
basque	esperanto	indonesian	polish	ukrainian
brazil	estonian	interlingua	polutonikogreek	uppersorbian
brazilian	finnish	irish	portuges	welsh
breton	francais	italian	portuguese	UKenglish
british	french	latin	romanian	USenglish

are dialects of French).[2] What is a dialect? While it is basically the same language as another, they might differ in minor aspects regarding typesetting rules or fixed tags.

In Portuguese typography, the month name in a date is capitalized, while Brazilians use lowercase. In Austria people speak German, but the name of the first month of the year is *Januar* in Germany and *Jänner* in Austria. These two languages can be called also with the `ngerman` or `naustrian` options, which select the "New Orthography" (*Neue Rechtschreibung*) hyphenation.

Other names are there just for backward compatibility: this is the case of `french` and `frenchb`. It is sufficient to look at the beginning of `babel.sty` to realize what each option does. Let's look at the first lines:

```
\DeclareOption{acadian}{\input{frenchb.ldf}}
\DeclareOption{albanian}{\input{albanian.ldf}}
\DeclareOption{afrikaans}{\input{dutch.ldf}}
```

Every language option loads a *language definition file* with extension `.ldf` (in the following, LDF). We see from these lines that `acadian` loads `frenchb.ldf`, and indeed Acadian is for babel a dialect of French. Similarly, Afrikaans is a dialect of Dutch. Conversely, Albanian is a language by itself, and it had better be, since it does not belong to any of the big European language families; the same is true for Basque.

The name of the LDF for French and its dialects is `frenchb` for historical reasons, which apply also to `germanb`: since there are packages around named french and german, the final 'b' was to remind users that they were using babel in the old days when packages were specified as options to `\documentstyle`.

[2] Of course, Canadien is not a dialect of Canadian.

Note that every option loads the corresponding LDF and it is this file's duty to handle double loadings. We'll see later some examples.

The most important thing to remember about language options is that the *last* language loaded is considered the main language of the document. In case there is only one it can be specified as a global option (i.e., as an option to `\documentclass`); other packages, such as varioref, understanding that option can therefore benefit from it. Notice, though, that varioref does not understand all babel's aliases. If there is more than one language, it can happen that a package does not correctly understand the global options: the solution is to specify them as local for each package.[3]

Don't specify a language as a global option and other languages as options to babel. This is a sure cause for head scratching, trying to figure out what went wrong. Try, for example

```
\documentclass[italian]{article}
\usepackage[greek,italian]{babel}
\begin{document}
XYZ
\end{document}
```

Do you see what happens? The option `italian` is not the last option seen by babel, because global options are scanned first.

3 Tags

In Table 2 is the list of fixed tags with their definitions in English. Not all of these tags are used in the standard classes article, report and book. For example, `\proofname` is used by amsthm as the name used in the `\begin{proof}` environment. In Table 3 you

[3] By the way, while writing this paper I discovered two bugs in varioref, version 1.4p: `\extrasbrazil` and `\extrasportuges` were misspelled as `\extrabrazil` and `\extraportuges`.

Table 2: List of tags in English

`\prefacename`	Preface
`\refname`	References
`\abstractname`	Abstract
`\bibname`	Bibliography
`\chaptername`	Chapter
`\appendixname`	Appendix
`\contentsname`	Contents
`\listfigurename`	List of Figures
`\listtablename`	List of Tables
`\indexname`	Index
`\figurename`	Figure
`\tablename`	Table
`\partname`	Part
`\enclname`	encl
`\ccname`	cc
`\headtoname`	To
`\pagename`	Page
`\seename`	see
`\alsoname`	see also
`\proofname`	Proof
`\glossaryname`	Glossary

Table 3: List of tags in Ukrainian

`\prefacename`	Вступ
`\refname`	Література
`\abstractname`	Анотація
`\bibname`	Бібліоґрафія
`\chaptername`	Розділ
`\appendixname`	Додаток
`\contentsname`	Зміст
`\listfigurename`	Перелік ілюстрацій
`\listtablename`	Перелік таблиць
`\indexname`	Покажчик
`\authorname`	Іменний покажчик
`\figurename`	Рис.
`\tablename`	Табл.
`\partname`	Частина
`\enclname`	вкладка
`\ccname`	копія
`\headtoname`	До
`\pagename`	с.
`\seename`	див.
`\alsoname`	див. також
`\proofname`	Доведення
`\glossaryname`	Словник термінів

find the same tags with their contents in Ukrainian; as you can see, different language traditions require also different tags.

What about changing or improving them? Suppose a document is written part in English and part in Italian. We would like to define a command to refer to sections in an abstract way, with text of the form

```
As we saw in \secref{sec:a} ...
...
Abbiamo visto nella \secref{sec:b} ...
```

in such a way that the command expands to 'Section 2' in English and to 'Sezione 2' in Italian. The definition is straightforward:

```
\newcommand{\secref}[1]{\secname~\ref{#1}}
```

But how to include `\secname` in the babel tags? It's a matter of saying, in the preamble of the document,

```
\newcommand{\secname}{}
\addto\captionsenglish{%
  \renewcommand{\secname}{Section}}
\addto\captionsitalian{%
  \renewcommand{\secname}{Sezione}}
```

We first introduce to LaTeX the command `\secname`; it is babel's job to provide the correct definition when the user chooses the English or the Italian language: babel orders LaTeX to execute `\captions`⟨*lang*⟩ whenever the ⟨*lang*⟩ is selected and the tags need to be changed. The `\addto` trick simply appends the second argument (a token list) to the replacement text of the control sequence given as first argument.

In the same vein, if we need to change a tag, say we want 'Elenco delle illustrazioni' instead of the default for `\listfigurename`, we can say

```
\addto\captionsitalian{%
  \renewcommand{\listfigurename}{%
    Elenco delle illustrazioni}}
```

It is better if these definitions to complement `\captions`⟨*lang*⟩ are given using only 7-bit input, so that they do not depend on the overall encoding of the document. In this way you will be able to simply copy those definitions from one document to another without worrying about the encoding; this is even more important if a personal style file is made.

The package has another facility: for each requested ⟨*lang*⟩, the macro `\extras`⟨*lang*⟩ is defined. It contains commands to be executed every time the ⟨*lang*⟩ is selected. A stupid example could be to typeset every part in Italian in bright red:

```
\addto\extrasitalian{\color{red}}
```

There is a companion macro `\noextras`⟨*lang*⟩ that contains things to be undone when passing from a language to another and this change is not protected by a group or environment. For example, correct hyphenation in Italian requires that the straight quote be considered for hyphenation, i.e., it must have a nonzero `\lccode`. Otherwise, phrases such as `dell'amicizia` would not be hyphenated fully as `del-l'a-mi-ci-zia` but only as `del-l'amicizia`. Therefore `italian.ldf` contains the instructions

```
\addto\extrasitalian{\lccode`\'=`\'}
\addto\noextrasitalian{\lccode`\'=0 }
```

because the \lccode of the straight quote must be reset to zero for other languages. If we were foolish enough to choose to typeset Italian in red, we should undo the choice when returning to other languages, so that we should say

```
\newcommand{\defaultcolor}{\color{black}}
\addto\noextrasitalian{\defaultcolor}
```

At \begin{document}, LaTeX will execute both \extras⟨*lang*⟩ and \captions⟨*lang*⟩, for the default ⟨*lang*⟩, so the modifications stated in the preamble will be active from the beginning.

Other facilities include the setting of dates. For every language there is a macro \date⟨*lang*⟩. When a different language is selected, LaTeX executes this command, which should redefine \today. So, say we want to use abbreviated month names in Italian: we issue in the preamble

```
\renewcommand{\dateitalian}{%
 \renewcommand{\today}{%
  \number\day~\ifcase\month\or gen.\or
   feb.\or ...\or dic.\fi\ \number\year}}
```

(the definition is incomplete to save space). The names of the months are not tags, because the date format can be very different between languages.

4 Language selection

Assume we have made our choice of the languages for the document. How to change from one to another? There are many ways, each solving a particular problem. The main language of the document is selected implicitly, because LaTeX issues a

```
\selectlanguage{⟨main-lang⟩}
```

command, where ⟨*main-lang*⟩ is the last chosen language option, as seen before.

Such a command can be issued everywhere; it changes everything to the new language: tags, typographical choices, shorthands and, of course, hyphenation rules. Therefore, after

```
\selectlanguage{portuges}
```

every following chapter will be tagged as '*Capítulo*'; after

```
\selectlanguage{french}
```

the typographical rules for French will be active. For example,

```
\selectlanguage{french}
Il dit: \og Qu'est-ce que tu veux?\fg
```

will be typeset as

Il dit : « Qu'est-ce que tu veux ? »

The correct spaces before the colon and the question mark will be automatically inserted, as required by the French tradition.

The language selection can act on various aspects regarding typesetting:

1. tags and dates,
2. typesetting conventions,
3. input conventions,
4. hyphenation.

The command \selectlanguage acts on all four aspects. The same holds for its environment form \begin{otherlanguage}. Input such as

```
\begin{otherlanguage}{turkish}
...
\end{otherlanguage}
```

is equivalent to \selectlanguage{turkish}, but confines the changes to the duration of the environment, in the usual way. The *-form environment \begin{otherlanguage*} acts only on typesetting and input conventions and hyphenation. It has a command form, for setting a small piece of text:

```
\foreignlanguage{⟨lang⟩}{⟨text⟩}
```

is largely equivalent to

```
\begin{otherlanguage*}{⟨lang⟩}
⟨text⟩
\end{otherlanguage*}
```

but the environment form allows many paragraphs.

The last environment is \begin{hyphenrules}; it acts only on the hyphenation rules. Usually, among the loaded hyphenation rules there is a set with no rule at all, commonly called nohyphenation. So, if we have text in an unsupported language, we can use this empty set of rules.

5 Other commands

The macro \languagename expands to the name of the current language. The command \iflanguage takes as arguments

1. a language name,
2. a token list to be executed if the current language is the same as the first argument,
3. a token list to be executed otherwise.

6 Input conventions

Before LaTeX supported 8-bit input via the inputenc package, people had a hard time with all the encodings used by different operating systems. Only 7-bit-clean input was guaranteed to be interpreted in the same way on all platforms. With TeX 2 it was even impossible to directly input characters in the upper half of an 8-bit code page.

During this time, the package german introduced a convention for inputting accented characters by preceding them with a double quote:

```
sch"oner G"otterfunken Stra"se
ba"cken Schi"ffart
```

could be used instead of the more awkward

```
sch\"oner G\"otterfunken Stra{\ss}e
ba{\ck}en Schi{\ff}ahrt
```

after having defined

```
\def\ck{\discretionary{k-}{k}{ck}}
\def\ff{ff\discretionary{-}{f}{}}
```

and similar commands for other sequences.

Braams developed this scheme further, making it easy to define similar shorthands for all languages. Nowadays, with the development of encodings such as UTF-8, these conventions are less important. However, UTF-8 is not yet widespread and is intrinsically foreign to standard TEX, so occasionally they can still be useful.

Suppose I have to use a Latin 1 keyboard, but need to type text in Czech: most of the diacritics used by Czech are not directly accessible with Latin 1. Fortunately, it is fairly easy to set up suitable "double quote" conventions.

The only letters that can take different diacritics are the 'e' (háček and acute accent) and the 'u' (ring and acute accent). Since Latin 1 keyboards have vowels with the acute accent, except for 'y', we don't need anything special for the other five.

Let's analyze the Czech alphabet. It uses four kinds of diacritics: the *háček* (as in 'č'), the acute accent (as in 'ý'), the ring (as in 'ů') and the apostrophe. The háček is produced with `\v`; the Czech support by babel provides `\q` for the apostrophe and `\w` for the the ring.

Let's decide to use the double quote for inputting most diacritics. In a `.sty` file to be loaded after babel we can write the following code.

```
\initiate@active@char{"}
\addto\extrasczech{%
  \languageshorthands{czech}}
\addto\extrasczech{\bbl@activate{"}}
\addto\noextrasczech{\bbl@deactivate{"}}
\begingroup \catcode`\"12
\def\x{\endgroup
  \def\dq{"}}\x
```

Here `\initiate@active@char`, `\bbl@activate` and `\bbl@deactivate` are standard babel functions; we add `\languageshorthands` to `\extrasczech` in order to declare the use of the defined shorthands. The last three lines are a trick to define `\dq` as a double quote with category code 12.

Then we can write (incomplete for brevity):

Table 4: Improved input for Czech or „Česko“

A	A	"E	Ě	"L	Ľ	"R	Ř	X	X
Á	Á	F	F	M	M	S	S	Y	Y
B	B	G	G	N	N	"S	Š	"Y	Ý
C	C	H	H	"N	Ň	T	T	Z	Z
"C	Č	I	I	O	O	"T	Ť	"Z	Ž
D	D	Í	Í	Ő	Ó	U	U		
"D	Ď	J	J	P	P	Ú	Ú		
E	E	K	K	Q	Q	"U	Ů	"‘	„
É	É	L	L	R	R	V	V	"’	“
a	a	"e	ě	"l	ľ	"r	ř	x	x
á	á	f	f	m	m	s	s	y	y
b	b	g	g	n	n	"s	š	"y	ý
c	c	h	h	"n	ň	t	t	z	z
"c	č	i	i	o	o	"t	ť	"z	ž
d	d	í	í	ó	ó	u	u		
"d	ď	j	j	p	p	ú	ú		
e	e	k	k	q	q	"u	ů	"<	«
é	é	l	l	r	r	v	v	">	»

```
\declare@shorthand{czech}{"c}
  {\textormath{\v{c}}{\ddot c}}
\declare@shorthand{czech}{"C}
  {\textormath{\v{C}}{\ddot C}}
\declare@shorthand{czech}{"d}
  {\textormath{\q{d}}{\ddot d}}
\declare@shorthand{czech}{"D}
  {\textormath{\q{D}}{\ddot D}}
...
\declare@shorthand{czech}{"y}
  {\textormath{\'{y}}{\ddot y}}
\declare@shorthand{czech}{"Y}
  {\textormath{\'{Y}}{\ddot Y}}
\declare@shorthand{czech}{"z}
  {\textormath{\v{z}}{\ddot z}}
\declare@shorthand{czech}{"Z}
  {\textormath{\v{Z}}{\ddot Z}}
\declare@shorthand{czech}{"‘}
  {\textormath{\quotedblbase}
              {\mbox{\quotedblbase}}}
\declare@shorthand{czech}{"’}
  {\textormath{\textquotedblleft}
              {\mbox{\textquotedblleft}}}
\declare@shorthand{czech}{"<}{\flqq}
\declare@shorthand{czech}{">}{\frqq}
```

The `\declare@shorthand` command used here takes three arguments:

1. a language name,
2. a one or two character sequence, the first of which must have been declared as we did before,
3. what we want to substitute when TEX sees that character sequence.

Table 5: Warning message for missing hyphenation patterns

```
Package babel Warning: No hyphenation patterns were loaded for
(babel)                the language 'Albanian'
(babel)                I will use the patterns loaded for \language=0 instead.
```

The macro `\textormath` typesets its first argument in text mode, the second one in math mode. It is customary to make a double quote combination equivalent to the math accent `\ddot` in order to avoid strange error messages.

We take the occasion to also introduce abbreviations for the inverted double quotes in German style, common in Czech, and also the guillemets (`\flqq` and `\frqq` are babel jargon for them). The non-obvious choices are `"y` and `"u` for the 'y with acute accent' and 'u with ring' (and the corresponding uppercase letters). In Table 4 we find a list of characters along with their input.

Users can define new shortcuts on the fly using similar commands. The internal commands are more efficient and can be restricted to one language:

- `\useshorthands{`⟨*chars*⟩`}`, for introducing new shorthand characters.
- `\defineshorthand`, which behaves like the internal command `\declare@shorthand`, but it doesn't take a language as an argument, only the shorthand and its definition.
- `\aliasshorthand{`⟨*char*$_1$⟩`}{`⟨*char*$_2$⟩`}` for making ⟨*char*$_2$⟩ a shorthand equivalent to ⟨*char*$_1$⟩.

Continuing our example, we could add

```
\aliasshorthand{"}{|}
```

to `\extrasczech` and then input `"C` as `|C`. Of course ⟨*char*$_1$⟩ must have already been defined as a shorthand.

Sometimes it is necessary to disable a shorthand character, because of bad interactions with other packages, notably XY-pic. This package does its job by parsing the source looking for special characters. An activated character is likely to disturb this parsing, so users can say

```
\shorthandoff{⟨chars⟩}
\shorthandon{⟨chars⟩}
```

where ⟨*chars*⟩ is the list of characters to disable or enable. For example:

```
\shorthandoff{"^}
```

7 Attributes

Some languages can have *attributes*, which modify their behavior. Currently only `greek` and `latin` use this facility. So "Polytoniko Greek" can be chosen either with

```
\usepackage[greek]{babel}
\languageattribute{polutoniko}
```

or simply by specifying `polutonikogreek` as the language.

Latin uses the `withprosodicmarks` attribute, which makes `^` and `=` shorthands to typeset accents in Latin poetry and specifying the vowel quantities in order to emphasize the meter. It has also the attribute `medievallatin` for making the lowercase 'u' equivalent to the uppercase 'V' and using traditional ligatures.

8 Problems

The choice of hyphenation rules is done at format creation, based on the file `language.dat`. This file (excluding comments) has the following appearance:

```
english         hyphen.tex
=usenglish
=USenglish
=american
usenglishmax    ushyphmax.tex
dumylang        dumyhyph.tex
nohyphenation   zerohyph.tex
basque          xu-bahyph.tex
bulgarian       xu-bghyphen.tex
...
```

Its format is due to Sebastian Rahtz. Basically it lists on each line a language name along with the hyphenation patterns file; a line can consist also of an equal sign followed by a language name, meaning that this name is an alias for the preceding two-item line. At this level, language names are actually arbitrary strings. It is babel's job to associate each of its supported languages with one of these strings.

A problem comes immediately to our attention: Albanian is supported by babel, but no hyphenation patterns for it are available. A user saying

```
\usepackage[albanian]{babel}
```

will be saluted with a message from TEX which you find in Table 5.

A warning of the same type would appear for any language whose hyphenation patterns were not enabled at format creation time by the system administrator. Some distributions like MiKTEX are pretty conservative in this regard and enable only a few languages; many questions in the discussion forums are about this.

In my opinion this is a design error: in fact Albanian (or the other non-enabled language) would be hyphenated using US English rules which are completely different from those of Albanian. I believe that no hyphenation is better than *wrong* hyphenation. Splitting an Italian word like *cestino* (small basket) as *ces-tino* is a bad grammatical error.[4]

In Albanian the digraph 'rr' is considered a single letter, and must never be divided. And this is only one of the problems which can arise by allowing hyphenation with English rules.

All language definition files begin with something like

```
\ifx\l@italian\@undefined
    \@nopatterns{Italian}%
    \adddialect\l@italian0\fi
```

This could be changed into

```
\ifx\l@italian\@undefined
    \@nopatterns{Italian}%
  \ifx\l@nohyphenation\@undefined
    \adddialect\l@italian\@cclv
  \else
    \adddialect\l@italian\l@nohyphenation
  \fi
\fi
```

Thus we would associate a non-enabled language either to the one with no hyphenation patterns by definition or to language number 255, which is very likely undefined. The `\@nopatterns` error message can be changed by saying that TEX won't use any hyphenation.

Users who can't enable a language, either because they are not the system administrator or there is no hyphenation patterns file, can correct this behavior themselves by finding in the log file the warning similar to that of Table 5; immediately after it they'll find a line such as

```
\l@albanian = a dialect from \language0
```

The first control sequence is the key to the solution. It is now sufficient to write, just after loading babel,

```
\makeatletter
\ifx\l@nohyphenation\@undefined
\chardef\l@albanian=255
\else
\let\l@albanian=\l@nohyphenation
\fi
\makeatother
```

Since commands like `\selectlanguage{albanian}` execute the command

```
\language\l@albanian
```

the trick is done.

[4] The kind of error which made our teachers in primary school shriek in horror.

It is important to note that shorthand characters in a language remain active also in the languages where they are not used in that way; in those cases they expand to the character they denote. This is why they can have unwanted side effects with other packages.

The reason to keep them active is clear: a language changing command can appear in risky places, for example in the `.aux` file, as Braams points out in the babel documentation. The example he makes is the following: a user could use a shorthand in the optional argument of a `\bibitem` command.

However even this doesn't work. Suppose someone writes a document with German as the main language and parts in English. Assume that in the bibliography, written in German, we find

```
\bibitem["UB99]{ub99}
A. "User und E. Benutzer, ``Ein Titel''.
```

A reference like `\cite{ub99}` will come out correctly in a German context, but not in an English context: they will print, respectively, [ÜB99] and ["UB99] (or ["UB99], if OT1 encoding is used). The reference will consistently be resolved correctly only if the author writes

```
\bibitem[\german{"UB99}]{ub99}
```

(where I've used `\german` simply as a shortcut for `\foreignlanguage{german}`).

9 Double loading

The LDF for Hungarian starts as follows:

```
\@namedef{captions\CurrentOption}{%
  \def\prefacename{El\H osz\'o}%
  \def\refname{Hivatkoz\'asok}%
...
\@namedef{date\CurrentOption}{%
  \def\today{%
    \number\year.\nobreakspace
    \ifcase\month\or
...
```

after the check for the existence of the hyphenation patterns in the format. What does this mean?

This LDF is called if Hungarian is requested with either the option `hungarian` or `magyar`. In the first case, the fundamental macros will be defined as

```
\captionshungarian
\datehungarian
\extrashungarian
\noextrashungarian
```

and with the suffix `magyar` in the second case. Quite recently, Péter Szabó has exploited this possibility with a new implementation of the Hungarian LDF (`http://ctan.org/tex-archive/language/hungarian/babel`).

10 Plain TEX users

In principle, it should be possible to use babel with Plain TEX or formats built upon it like $\mathcal{AMS}$-TEX. However, this is pretty much undocumented, apart from the instructions to build a format by running iniTEX on `bplain.tex` and `\dump`. The user interface is not specified.

Recently I wrote a very primitive Plain TEX package supporting multiple languages. It's a substantially simplified version of babel's machinery and I will use it to try and illustrate how it works.

First we run iniTEX on a file called `hyplain.tex`:

```
\catcode`\{=1
\catcode`\}=2
\catcode`\@=11
\let\orig@input\input
\def\input hyphen {%
  \let\input\orig@input \input hyrules }
\orig@input plain
```

D. E. Knuth has decreed that `plain.tex` cannot be modified except for preloaded fonts. But we can always use some TEX trick; since the file *is* immutable, it will contain the line '`\input hyphen`'; at that point we restore the original meaning of `\input` and input `hyrules.tex` instead of `hyphen.tex`.

The file `hyrules.tex` defines the interface commands. The most important is `\selectlanguage` which, unlike that of babel, requires two arguments: a two letter ISO language code and a two letter country code. For example,

```
\selectlanguage{en}{US}
\selectlanguage{it}{IT}
\selectlanguage{de}{AT}
```

would switch, respectively, to American English, to the Italian of Italy[5] and to the German of Austria.

Users can also define personal language selection commands: define command `\italiano` with

```
\addalias{\italiano}{it}{IT}
```

to make it equivalent to `\selectlanguage{it}{IT}`. Internally, this command calls `\it_IT` and similarly for other combinations. If a language combination is not defined in the user modifiable file `hylang.tex`, a fallback language `\zz_ZZ`, without hyphenation patterns, is selected and a warning message is issued.

Two token lists are associated to each language, similar to the `\extras`⟨*lang*⟩ and `\noextras`⟨*lang*⟩ of babel and an 'undo' token list is maintained. Each time a language is selected, the following happens:

1. what is in the 'undo' token list is executed and the list is cleared (locally),
2. the parameter `\language` is given the appropriate value,
3. the *extras* for the chosen language are executed,
4. the *noextras* are put in the 'undo' token list.

Since assignments to the 'undo' token list are local, this list will always be loaded with the correct commands.

The 'extras' list for a language ought to set the left and right hyphenation minima; this setting has no counterpart in the 'noextras' list. Other things, instead, must be set in both places: for example, the `\lccode` setting of the apostrophe for Italian.

Users can modify `hylang.tex`, adding or deleting languages. It is recommended not to change the first one, so that we are sure that `\language` zero refers always to American English, as in Plain TEX.

The commands are

```
\definelanguage{xx}{YY}{xxhyph}
\refinelanguage{xx}{YY}
  {<something>}{<something>}

\definedialect{yy}{YY}{xx}{XX}
\refinedialect{yy}{YY}
  {<something>}{<something>}
```

where `xx` is the two letter code of a language (I suggest '`nde`' for 'New Orthography German') and `XX` is the two letter country code. Actually these codes could be arbitrary strings, but I believe that we (and babel) would benefit from standardization.

For example, one could write

```
\definelanguage{fr}{FR}{frhyph}
\definedialect{fr}{CA}{fr}{FR}
```

to set up for Canadian French. One could use the nonexistent country code 'ZZ' for an unspecified country: this would be the case for Esperanto.

The macro `\refinelanguage` refers to the language by its codes; then in the third argument one puts the 'extras' and in the fourth the 'noextras'. This command can be given as many times as one desires, since new lists are appended to the existing ones. The macro `\refinedialect` is the same as `\refinelanguage`. Dialects do not inherit extras: the interface is primitive, just the way Plain TEX devotees are used to.

For example, my settings for Italian are these:

```
\definelanguage{it}{IT}{ithyph}
\refinelanguage{it}{IT}
  {\lccode`\'=`\'}{\lccode`\'=0 }
```

11 Advanced babel programming

babel works in a similar way to HyPlain. It has of course many more features: for example, functions to save the meaning of commands or the value of variables. In the LDF for Italian we can see

[5] Italian is an official language also in Switzerland, where different typography conventions could be used.

```
\addto\extrasitalian{%
  \babel@savevariable\clubpenalty
  \babel@savevariable\widowpenalty
  \babel@savevariable\finalhyphendemerits
  \clubpenalty3000 \widowpenalty3000
  \finalhyphendemerits50000000 }%
```

When the language changes from Italian to another one, the values of the listed parameters[6] are restored and possibly changed again by the new language: babel uses the 'noextras' token list and its internal mechanism to restore a clean setting and then it applies the 'extras' for the new language. Let's see how it's done:

```
\def\babel@savevariable#1{\begingroup
    \toks@\expandafter{\originalTeX #1=}%
    \edef\x{\endgroup
      \def\noexpand\originalTeX{%
        \the\toks@ \the#1\relax}}%
  \x}
```

Let the variable name be `\foo`. The macro appends to the replacement text of `\originalTeX` (which corresponds to the 'undo' list in HyPlain) the tokens `\foo=`. This is done inside a group in order to be sure not to clobber the value of `\toks@`. The `\edef` is done when the value of `\toks@` is what has just been set; after that token list, the present value of `\foo` is put. Then `\x` is executed, which closes the group and redefines `\originalTeX`.

A very similar trick is performed when we say `\babel@save\baz`, where `\baz` is a command. First `\baz` is made equivalent to a command of the form `\babel@1234` (where `1234` stands for the actual value of a counter reserved by babel). The same thing happens as before, so `\originalTeX`'s replacement text will end with

```
\let\baz=\babel@1234
```

(there is no problem in interpreting that strange token, because it has already entered the scanning mechanism). Finally, the counter is stepped, providing a fresh number for the next `\babel@save`.

We can apply this method to modify the behavior of a command without forcing users to change their input. A silly example is the following:

```
\makeatletter
\addto\extrasitalian{%
  \babel@save\emph\let\emph\textbf}
\makeatother
```

In this way, typing `\emph{ciao}` in an Italian context would print the word in bold face, while keeping the abstract nature of the command. This could be obtained also by

```
\let\origemph\emph
\renewcommand{\emph}{%
  \iflanguage{italian}%
    {\textbf}{\origemph}}
```

but the method with `\babel@save` is of course more robust and does not require a long chain of nested `\iflanguage` calls if we need different effects for several languages.

I've said before that `\declare@shorthand` takes as the first argument a language name, but this is not strictly true. There is the concept of 'shorthand group'. In the present version of babel there are three levels: (1) user, (2) language, and (3) system. The package checks in that order when it is resolving a shorthand.

Let's make an example: German uses the double quote as a shorthand character, for instance `"A` to get 'Ä'. It is not necessary to define every combination `"`⟨*char*⟩, because there is already a definition of the active double quote at the system level (it expands to a double quote, of course).

The default system level shorthands are `"`, `'`, `` ` ``, and `~`. When an LDF introduces a new shorthand character, it ought to define its behavior at the system level. For example the LDF for Esperanto says

```
\declare@shorthand{system}{^}{%
  \csname normal@char\string^\endcsname}
```

because it uses `^` for shorthands. The same is true of `frenchb.ldf`, where there is

```
\declare@shorthand{system}{:}{\string:}
```

along with similar lines for `!`, `?` and `;`. If a user says `\defineshorthand{"A}{\hat{A}}`, this shorthand would take precedence over a possible definition of `"A` by the LDF. If the LDF defines a `"` shorthand, this takes precedence over the system one.

With the development of input encoding support, especially Unicode, for TEX these devices are less useful, because it is possible to input directly any character. On the other hand, other babel features remain invaluable.

The most recent versions of pdfTEX allow a different treatment for the typographic conventions of French, for example, making it possible to reduce the number of active characters. Some support for this is available through the microtype package.

◇ Enrico Gregorio
Dipartimento di Informatica, Settore di Matematica
Università di Verona, Italy
Enrico dot Gregorio (at) univr dot it
http://profs.sci.univr.it/~gregorio

[6] The setting of `\clubpenalty` is wrong, it should refer to `\@clubpenalty`. A bug report has been mailed.

Macros

Writing numbers in words in TEX

Edward M. Reingold

We present TEX macros to write integers, even extremely large integers, in words according to the American English nomenclature [2, pp. 12 and 22–24], [3, p. 1549]; the method here is easily adapted to the British English nomenclature, or that of other languages. The imaginative nomenclatures of [1, pp. 14–15] or [4, pp. 311–312] are also easy to accommodate with the ideas presented here. Although macros for writing numbers in words are already available on CTAN, none of them has the generality of those presented here. Our approach, which covers the full range of American English, $(-10^{66}, 10^{66})$, is based on [6, sec. 8.1]; [5, p. 6] has a similar method.

We want to be able to capitalize the first word produced, as well as insert spaces, commas, and hyphens between words appropriately. Because the words produced are written by various macros and at various levels of recursion, we centralize the production of text by calling a macro `\@String` that does the actual insertion of the word into the output. We use global flags to indicate whether the next word produced will be the first word (which should not be preceded by a space and which may need to be capitalized),

```
\def\@firstwordtrue{%
   \global\let\if@firstword\iftrue}
\def\@firstwordfalse{%
   \global\let\if@firstword\iffalse}
```

and similar global variables to indicate whether a capital letter is needed,

```
\def\@capitalfirstwordtrue{%
   \global\let\if@capitalfirstword\iftrue}
\def\@capitalfirstwordfalse{%
   \global\let\if@capitalfirstword\iffalse}
\@capitalfirstwordfalse
```

or whether a hyphen or comma is needed,

```
\def\@needhyphentrue{%
   \global\let\if@needhyphen\iftrue}
\def\@needhyphenfalse{%
   \global\let\if@needhyphen\iffalse}
\def\@needcommatrue{%
   \global\let\if@needcomma\iftrue}
\def\@needcommafalse{%
   \global\let\if@needcomma\iffalse}
\@needcommafalse
```

The macro that inserts a word into the output then is

```
\def\@String#1{%
  \if@firstword
    \expandafter\@firstoftwo\else
    \expandafter\@secondoftwo\fi
    {\if@capitalfirstword
      \Capitalize{#1}%
      \@capitalfirstwordfalse
    \else
      {#1}%
    \fi}%
   {\if@needhyphen
      {-#1}%
    \else
      \if@needcomma
          {, #1}%
          \@needcommafalse
      \else
          { #1}%
      \fi
    \fi}%
  \@firstwordfalse}
```

where capitalization is done by

```
\def\Capitalize#1{%
  \edef\@tempa{#1}%
  \expandafter\@capitalize
  \expandafter{\@tempa}\@EndOfString}
\def\@capitalize#1{%
   \ifx\@EndOfString#1%
     \expandafter\@firstoftwo\else
     \expandafter\@secondoftwo\fi
     {}{\@@capitalize#1}}
\def\@@capitalize#1#2\@EndOfString{%
   \uppercase{#1}#2}
```

Numbers less than twenty are idiosyncratic, so we handle them with a case statement:

```
\def\Small@Number#1{% Less than 20
  \relax
  \ifcase#1
    \@String{zero}\or
    \@String{one}\or
    \@String{two}\or
    \@String{three}\or
    \@String{four}\or
    \@String{five}\or
    \@String{six}\or
    \@String{seven}\or
    \@String{eight}\or
    \@String{nine}\or
    \@String{ten}\or
    \@String{eleven}\or
    \@String{twelve}\or
    \@String{thirteen}\or
    \@String{fourteen}\or
    \@String{fifteen}\or
    \@String{sixteen}\or
    \@String{seventeen}\or
    \@String{eighteen}\or
    \@String{nineteen}%
```

```
  \else
    \errmessage{Small number out of range}
  \fi}
```

We need some scratch counters to do our work:

```
\newcount\@number
\newcount\@@number
\newcount\@@@number
\newcount\@millenary
```

We use `\Medium@Number` to handle numbers smaller than 1000:

```
\def\Medium@Number#1{% At most three digits
  \@number=#1\relax
  \ifnum\@number>99
    \@@number=\@number
    \divide\@@number by 100
    \Small@Number{\the\@@number}%
    \@String{hundred}%
    \multiply\@@number by 100
    \advance\@number by -\@@number
  \fi
  % \@number is now \number mod 100
  \ifnum\@number>19
    \@@number=\@number
    \divide\@@number by 10
    % \@@number is now the tens digit
    \@Decade{\the\@@number}%
    \multiply\@@number by 10
    \advance\@number by -\@@number
    % \@number is now the ones digit
    \@needhyphentrue
  \fi
  % \@number is now 19 or less
  \ifnum\@number>0
      \Small@Number{\the\@number}%
  \fi
  \@needhyphenfalse}
```

where the "decade" is written by

```
\def\@Decade#1{%
  \ifcase#1
    \errmessage{Decade out of range}\or
    \errmessage{Decade out of range}\or
    \@String{twenty}\or \@String{thirty}\or
    \@String{forty}\or \@String{fifty}\or
    \@String{sixty}\or \@String{seventy}\or
    \@String{eighty}\or \@String{ninety}%
  \else
    \errmessage{Decade out of range}
  \fi}
```

Some usage requires the word "and" after the word "hundred", especially for the rightmost three digits of a number (for example, "one hundred and twenty"); this would require another global variable and a slight modification of `\Medium@Number`.

Numbers with four or more digits are handled recursively. To express $n \times 1000^i$ in words, we

express $\lfloor n/1000 \rfloor \times 1000^{i+1}$ in words,

express n mod 1000 in words, and

write the name of 1000^i in words.

The last step, writing the name of 1000^i, is done with

```
\def\@Millenary#1{%
  \ifcase#1\or
    \@String{thousand}\or
    \@String{million}\or
    \@String{billion}\or
    \@String{trillion}\or
    \@String{quadrillion}\or
    \@String{quintillion}\or
    \@String{sextillion}\or
    \@String{septillion}\or
    \@String{octillion}\or
    \@String{nonillion}\or
    \@String{decillion}\or
    \@String{undecillion}\or
    \@String{duodecillion}\or
    \@String{tredecillion}\or
    \@String{quattuordecillion}\or
    \@String{quindecillion}\or
    \@String{sexdecillion}\or
    \@String{septendecillion}\or
    \@String{octodecillion}\or
    \@String{novemdecillion}\or
    \@String{vigintillion}%
  \else
    \errmessage{Number too large for words}
  \fi
  \ifnum#1>0 \@needcommatrue\fi}
```

A "vigintillion" (10^{63}) is as high as American nomenclature goes; this is far larger than a TEX counter can go, but we are aiming high! With `\@Millenary` we translate our recursive structure into

```
\def\Big@Number#1#2{%
  \@@number=#2\relax    % number to be written...
  \@millenary=#1\relax  % times this power of 1000
  \ifnum\@@number>0
    \@@@number=\@@number
    \divide\@@@number by 1000
    \begingroup% Preserve \@millenary value
      \advance\@millenary by 1
      \Big@Number{\the\@millenary}%
                 {\the\@@@number}%
    \endgroup
    \multiply\@@@number by 1000
    \advance\@@number by -\@@@number
   % \@@number is now #2 mod 1000
    \ifnum\@@number>0
       \Medium@Number{\the\@@number}%
       \@Millenary{#1}%
    \fi
  \fi}
```

Calling `\Big@Number` produces `#2` $\times 1000^{\texttt{\#1}}$ in words, so the initial call to `\Big@Number` should have

a first parameter of zero. Thus we write the public macros

```
\def\inwords#1{%
  \edef\@tempa{#1}%
  \expandafter\@inwords\expandafter{\@tempa}}
\def\Inwords#1{%
  \@capitalfirstwordtrue
  \edef\@tempa{#1}%
  \expandafter\@inwords\expandafter{\@tempa}%
  \@capitalfirstwordfalse}
```

where

```
\def\@inwords#1{%
  \@firstwordtrue
  \@needcommafalse
  \@needhyphenfalse
  \ifnum#1<0
    \@String{minus}%
    \@number=-#1\relax
    \Big@Number{0}{\the\@number}%
  \else\ifnum#1=0
    \Small@Number{0}%
  \else%
    \Big@Number{0}{#1}%
  \fi\fi}
```

For example, `\inwords{-1234567890}` produces

> minus one billion, two hundred thirty-four million, five hundred sixty-seven thousand, eight hundred ninety

and `\Inwords{31415926}` produces

> Thirty-one million, four hundred fifteen thousand, nine hundred twenty-six.

The size limitation of TEX count registers, $2^{31}-1$, means that we get an error in trying to write 8018018851 in words (Conway and Guy [1, p. 15] call this "Knuth's number", the first prime number in the alphabetic ordering of the natural numbers [5, p. 4]). To write larger numbers in words we need to use the recursive structure of `\Big@Number` without resorting to count registers. This means that we have to trap a minus sign and ignore leading zeros, before we can split the number into the rightmost three digits and everything to their left. Hence we redefine

```
\def\Inwords#1{%
  \@capitalfirstwordtrue
  \inwords{#1}%
  \@capitalfirstwordfalse}
\def\inwords#1{%
  \@firstwordtrue
  \@needhyphenfalse
  \Trap@Minus{#1}}%
```

where

```
\def\Trap@Minus#1{%
  \edef\@tempa{#1}%
  \expandafter\@Trap@Minus%
  \expandafter{\@tempa}\@EndOfString}
\def\@Trap@Minus#1{%
  \ifx\@EndOfString#1%
     \expandafter\@firstoftwo\else
     \expandafter\@secondoftwo\fi
     {}{\@@Trap@Minus#1}}
\def\@@Trap@Minus#1#2{%
   \ifx\@EndOfString#2%
     \expandafter\@firstoftwo\else
     \expandafter\@secondoftwo\fi
     {\ifx#1-%
        \errmessage{Orphan minus sign}%
      \else
        \Small@Number{#1}\fi}%
     {\@@@Trap@Minus#1#2}}
\def\@@@Trap@Minus#1#2\@EndOfString{%
   \ifx#1-%
     \expandafter\@firstoftwo\else
     \expandafter\@secondoftwo\fi
     {\@String{minus}%
      \@TrapLeadingZeros{#2\@EndOfString}}%
     {\@TrapLeadingZeros{#1#2\@EndOfString}}}
```

traps a leading minus sign and then traps leading zeros with the similar

```
\def\@TrapLeadingZeros#1{%
   \ifx\@EndOfString#1%
     \expandafter\@firstoftwo\else
     \expandafter\@secondoftwo\fi
     {}{\@@TrapLeadingZeros#1}}
\def\@@TrapLeadingZeros#1#2{%
   \ifx\@EndOfString#2%
     \expandafter\@firstoftwo\else
     \expandafter\@secondoftwo\fi
     {\Small@Number{#1}}%
     {\@@@TrapLeadingZeros#1#2}}
\def\@@@TrapLeadingZeros#1#2\@EndOfString{%
   \ifx0#1%
     \expandafter\@firstoftwo\else
     \expandafter\@secondoftwo\fi
     {\@TrapLeadingZeros{#2\@EndOfString}}%
     {\@numberinwords{0}{#1#2\@EndOfString}}}
```

before calling a version of `\Big@Number` that avoids the use of count registers by splitting a number into the rightmost three digits and everything else:

```
\def\@numberinwords#1#2{%
   % #1 = power of 1000
   % #2 = the next token,  either
   %      a digit or \@EndOfString
   \ifx\@EndOfString#2%
     \expandafter\@firstoftwo\else
     \expandafter\@secondoftwo\fi
   {}{\@@numberinwords{#1}{}#2}}
\def\@@numberinwords#1#2#3#4{%
   % #1 = power of 1000
   % #2 = string digits so far,
   %      excluding the final one, #3
```

```
   % #3 = the next digit
   % #4 = the next token, either another
   %      digit or \@EndOfString
   \ifx\@EndOfString#4%
      \expandafter\@firstoftwo\else
      \expandafter\@secondoftwo\fi
      {\Small@Number{#3}\@Millenary{#1}#2}%
      {\@@@numberinwords{#1}{#2}#3#4}}
\def\@@@numberinwords#1#2#3#4#5{%
   % #1 = power of 1000
   % #2 = string digits so far,
   %      excluding the final two, #3#4
   % #3#4 = the final two digits so far
   % #5 =  the next token, either another
   %       digit or \@EndOfString
   \ifx\@EndOfString#5%
     \expandafter\@firstoftwo\else
     \expandafter\@secondoftwo\fi
     {\Medium@Number{#3#4}\@Millenary{#1}#2}%
     {\@@@@numberinwords{#1}{#2}#3#4#5}}
\def\@@@@numberinwords#1#2#3#4#5#6{%
   % #1 = power of 1000
   % #2 = string digits so far,
   %      excluding the final three, #3#4#5
   % #3#4#5 = the final three digits so far
   % #6 = the next token, either another
   %      digit or \@EndOfString
   \ifx\@EndOfString#6%
     \expandafter\@firstoftwo\else
     \expandafter\@secondoftwo\fi
     {\@millenary=#1\relax
      \advance\@millenary by 1
      \@numberinwords{\the\@millenary}
                     {#2\@EndOfString}%
      \advance\@millenary by -1
      \ifnum#3#4#5>0
        \Medium@Number{#3#4#5}%
        \@Millenary{#1}%
      \fi}%
     {\@@@@numberinwords{#1}{#2#3}#4#5#6}}
```

Given this machinery, we can write Knuth's number using `\inwords{8018018851}`, eight billion, eighteen million, eighteen thousand, eight hundred fifty-one, and $2\times10^{63}+2\times10^{36}+2\times10^{12}+2293$, the last prime in alphabetical order [5, p. 12], two vigintillion, two undecillion, two trillion, two thousand, two hundred ninety-three. Or,

$$2^{219} = 842498333348457493583344221469363 \\ 458551160763204392890034487820288,$$

which in words is

> Eight hundred forty-two vigintillion, four hundred ninety-eight novemdecillion, three hundred thirty-three octodecillion, three hundred forty-eight septendecillion, four hundred fifty-seven sexdecillion, four hundred ninety-three quindecillion, five hundred eighty-three quattuordecillion, three hundred forty-four tredecillion, two hundred twenty-one duodecillion, four hundred sixty-nine undecillion, three hundred sixty-three decillion, four hundred fifty-eight nonillion, five hundred fifty-one octillion, one hundred sixty septillion, seven hundred sixty-three sextillion, two hundred four quintillion, three hundred ninety-two quadrillion, eight hundred ninety trillion, thirty-four billion, four hundred eighty-seven million, eight hundred twenty thousand, two hundred eighty-eight.

A final note: to number pages in words in LaTeX we need to `\protect` the call, as in:

```
\renewcommand*{\thepage}
      {Page \protect\inwords{\c@page}}
```

Acknowledgments The author is grateful to Nachum Dershowitz for pointing out various errors in the original macros, and to both him and Peter Wilson for suggesting the inclusion of appropriate hyphens and commas.

References

[1] John H. Conway and Richard Guy. *The Book of Numbers*. Springer-Verlag, New York, 1996.

[2] Philip J. Davis. *The Lore of Large Numbers*. Yale University, New Haven, CT, 1961.

[3] Philip B. Gove. *Webster's Third New International Dictionary of the English Language*. Merriam, Springfield, MA, 1961.

[4] Donald E. Knuth. Supernatural numbers. In David A. Klarner, editor, *The Mathematical Gardner*, pages 310–325. Wadsworth, Boston, 1981.

[5] Donald E. Knuth and Allan A. Miller. A programming and problem-solving seminar. Technical Report STAN-CS-81-863, Department of Computer Science, Stanford University, Stanford, CA, June 1981.

[6] Edward M. Reingold and Ruth N. Reingold. *PascAlgorithms*. Scott, Foresman and Company, Glenview, Illinois, 1988.

◇ Edward M. Reingold
Department of Computer Science
Illinois Institute of Technology
10 West 31st Street
Chicago, Illinois 60616-2987
USA
reingold (at) iit dot edu

ArsTEXnica
Contents of issues 2–3 (2006–2007)

Editor's note: *ArsTEXnica* is the journal of GUIT, the Italian TEX user group. The journal's web site is `http://www.guit.sasup.it/arstexnica`.

ArsTEXnica #2, October 2006

Massimiliano Dominici and Pier Daniele Napolitani, Edizione con LaTeX delle opere di Francesco Maurolico [A LaTeX edition of the works of Francesco Maurolico]; pp. 75–82

The Maurolico Project was started some years ago to publish, both in print and as electronic documents, the critical edition of Francesco Maurolico's (1494–1575) work. Within the project, a system called MAURO-TeX was built; it is able to obtain HTML output, intended for the publication on the Internet, and standard LaTeX output, intended as an intermediate format to PDF and PostScript, starting from a LaTeX-like mark-up language.

MAURO-TeX is undergoing a complete revision. We are moving the section related to text coding to XML; in addition, as soon as we started printing the work, we got feedback that helped us thoroughly revise the macros LaTeX uses to generate the PDF and PostScript *output*.

Roberta Tucci, L'edizione critica di un'opera matematica: MAURO-TeX e METAPOST [Mathematical works' critical editions: MAURO-TeX and METAPOST]; pp. 83–87

This article is in three parts: the first collects some philological problems that usually arise when starting a mathematical work's critical edition; the second part describes the MAURO-TeX and METAPOST tools chosen to edit the critical edition; the third and last part briefly shows the result obtained using the mentioned tools.

[Translation by G. Pignalberi]

ArsTEXnica #3, April 2007

Massimiliano Dominici, Editoriale [From the editor]; p. 3

A short overview of the present issue.

Frank Mittelbach, Gianluca Pignalberi and David Walden, Intervista a Frank Mittelbach [Interview with Frank Mittelbach]; pp. 4–12

Both the *Free Software Magazine* (FSM, `http://www.freesoftwaremagazine.com`) and the TeX Users Group (TUG, `http://www.tug.org/`)) both like to publish interviews. Recently, Gianluca Pignalberi of FSM and Dave Walden of TUG both approached Frank Mittelbach about interviewing him. Rather than doing two separate interviews, Mittelbach, Pignalberi, and Walden decided on a combined interview in keeping with the mutual interests already shared by FSM and TUG.

Claudio Beccari, LaTeX e la cesura delle parole in fin di riga [LaTeX and word hyphenation at line breaks]; pp. 13–20

This tutorial explains how TeX (the program) typesets paragraphs, possibly by hyphenating words at line breaks.

Specifically, this tutorial should explain LaTeX's (actually TeX's) strange behavior in certain circumstances when it apparently refuses to correctly break lines. If there is some error, unfortunately this is always a human one, and it is due to an insufficient understanding of TeX's procedures and algorithms.

Lapo Filippo Mori, LaTeXpedia: il futuro della documentazione su LaTeX [LaTeXpedia: the future of LaTeX documentation]; pp. 21—26

(Published in *The PracTEX Journal* 2007-1.)

Lapo Filippo Mori, Scrivere la tesi di laurea con LaTeX 2ε [Writing a thesis with LaTeX 2ε]; pp. 27—45

The goal of this article is to provide the tools to write a thesis with LaTeX 2ε. The article analyzes the problems that are usually encountered while writing a thesis and their solution; a particular emphasis is on the packages to use in each case. The topics are not examined in depth and, when necessary, the reader is referred to specific literature or to the manual of the suggested packages.

Salvatore Schirone, La tipografia nel taschino. Presentazione del sistema oPs4GuIt [Typography in the pocket: Overview of oPs4GuIt]; pp. 46–51

oPs4GuIt is an open source portable USB LaTeX system for Windows (9x, ME, XP), freely available on the Internet. oPs4GuIt provides a fully working LaTeX system always at hand, so that one can compile one's own `.tex` source files on any computer. In the present article I will introduce oPs4GuIt for the first time, and I will describe its structure and how to install and customize it.

[Translations by the authors.]

Les Cahiers GUTenberg
Contents of double issue 46–47 (April 2006)

Editor's note: *Les Cahiers GUTenberg* is the journal of GUT, the French TeX user group. Their web site is `http://www.gutenberg.eu.org`.

Issue 46–47 reprints a number of articles from the EuroTeX 2003 (Brest) conference. This proceedings was published as *TUGboat* 24:3, and is available online at `http://tug.org/TUGboat/Articles/tb24-3`.

Die TEXnische Komödie Contents of issues 2006/1–2007/1

Editor's note: *Die TEXnische Komödie* is the journal of DANTE e.V., the German-language TEX user group. The journal's web site is `http://www.dante.de/dante/DTK/`.

2006/1

Interview with Donald E. Knuth, Freude, die ein Maler empfindet [The joy a painter feels]; pp. 6–10

Donald Knuth has written more than a dozen books as well as the TEX typesetting system. He attained cult status among computer scientists with his multivolume magnum opus *The Art of Computer Programming*. Knuth, born in 1938 in Milwaukee, Wisconsin, started the book even before he finished studying mathematics at the California Institute of Technology.

The work was ranked by the science journal *The American Scientist* among the twelve most important scientific publications of the 20th century — yet it remains unfinished. *Technology Review* spoke with Knuth on the occasion of an honorary doctorate from ETH Zürich.

Markus Kohm, Farbig hinterlegte Kopfzeilen mit KOMA-Script [Colored backgrounds in headers with KOMA-Script]; pp. 11–18

When, in July 2005, `scrpage2` acquired the ability to color lines in headers and footers, the next logical question was, "Can I also color the entire background of the header?" Although the author of this article thinks that lines and colors in headers give them too much weight, the answer is, "Of course."

Ulrike Fischer, Trennhilfen [Hyphenation help]; pp. 19–24

In the documentation of `babel.sty -"` is explained as "an explicit hyphen sign". This is wrong: `-"` normally inserts no hyphen. This error inspired me to take a more thorough look at the word division and hyphenation commands.

Herbert Möller, Die GaPFilL-Methode zur Erzeugung von LATEX-picture-Umgebungen [The GaPFilL method for creating LATEX picture environments]; pp. 25–43

Drawing programs or geometry software and Perl filter programs are used to conveniently create even complicated figures with the LATEX picture environment. The filter programs parse PostScript files and generate LATEX code ready for use. The method will be explained via two filter programs for the geometry software Cabrigéomètre II. The first program requires only the `ebezier` package, and thus the output is driver independent. The second filter also supports the new package `pict2e`.

2006/2

The EuroTEX 2005 proceedings (previously sent to TUG members for 2006).

2006/3

Ulrich Schwarz, Was hinten herauskommt zählt: Counter Aliasing in LATEX [The result is what counts: Counter aliasing in LATEX]; pp. 6–11

For certain purposes it can be interesting to have several counters that have different *name* and `the`*name* representations, but share a counter value. We investigate this using as examples `hyperref` and the theorem environment.

Michael Niedermair and Markus Kohm, Marginalien, da wo man sie haben will! [Marginal notes where you want them]; pp. 12–17

In mailing lists and newsgroups one often reads of problems that marginal notes don't appear where one wants them, or that they have pushed footnotes to another page, or that marginal notes are needed where they can't be placed. The `marginnote` package is here to help.

2006/4

Stephen G. Hartke, Eine Übersicht freier Mathematikfonts für TEX und LATEX [An overview of free mathematical fonts for TEX and LATEX]; pp. 17–36

(Published in *The PracTEX Journal* 2006-1.)

Jan Weichold, Typographische Inszenierungen mit Textstrichen [Typographic productions with dashes]; pp. 37–39

Typography — including LATEX — recognizes a variety of dashes. The problem of hyphens and hyphenation has already been discussed in *Die TEXnische Komödie*. But which dash should be used for which case? A question that is unfortunately incorrectly answered in many cases.

Jürgen Fenn, Online-Bibliographien nutzen mit BibTEX [Using online bibliographies with BibTEX]; pp. 40–46

This article introduces the use of online bibliographies that offer data in BibTEX format. Several solutions are presented, among them the `mab2bib` converter, with which you can convert databases from MAB format to BibTEX.

Rolf Niepraschk, Tipps und Tricks: Vom LaTeX-Dokument zum einfachen Text-format [Tips and Tricks: From a LaTeX document to a simple text format]; pp. 47–49

In connection with documenting a LaTeX package, I wanted to have a list of previous corrections (`\changes` entries). For fast orientation, I wanted this as a file in simple text format in which formatting of the original was kept as much as possible. In the following, I show one way to do this.

Rolf Niepraschk, Tabulatoren ganz einfach [Tabbing made easy]; pp. 50–51

LaTeX, with the tabbing environment, makes tabbing possible in a manner similar to that of a typewriter, but the syntax conflicts with the accent macros. A package `tabto`, by Donald Arseneau, provides an alternative.

2007/1

Georg Verweyen, Von Gänsefüßen, Trottellummen und Doppelmöwchen [About Gänsefüßen, Trottellummen and Doppelmöwchen]; pp. 7–12

From handwritten marks, printing has evolved various forms of quotation marks. The naming as well as the use of these symbols is guided by various conventions. This article tries to throw some light on a gaggle of geese, sea gulls, and guillemots.

[Ed. note: *the terms for quotation marks in the title all have to do with birds; literally, "goosefeet", "guillemots", and "double sea gulls".*]

Hans Hagen, Jerzy B. Ludwichowski, and Volker RW Schaa, The New Font Project: TeX Gyre; pp. 12–20

(Published in *TUGboat* 27:2.)

Canon Deutschland, Digitale Druckvorstufe [Digital prepress]; pp. 21–39

Just about 20 years ago, more precisely in the year 1984, the page description language PostScript began a complete transformation of the production processes in prepress. All areas have been affected, beginning with typesetting, through layout, graphics, photo manipulation, to print preparation, including imposition, makeup, and exposure. Today all production steps run digitally.

Lars Madsen, Vermeidet eqnarray! [Avoid `eqnarray`!]; pp. 40–49

(Published in *The PracTEX Journal* 2006-4.)

[Translations by Steve Peter.]

***The PracTEX Journal* 2006-1–2007-2**

The PracTEX Journal is an online publication of the TeX Users Group. Its web site is `http://tug.org/pracjourn`. All articles are available there.

***The PracTEX Journal* 2006-1, 2006-02-18**

(Theme issue on fonts.)

Lance Carnes, From the editor

From the readers, Feedback

Robin Laakso, Invitation to PracTEX'06

Tamye Riggs, Typographic opportunities

This introductory article describes the current market in digital type, offers some insights on the use of different typefaces, and offers some tips for everyone interested in typefaces, both novice and experienced users. Several on-line typeface resources are given, including type foundries, font development tools, and typographic organizations, conferences, and discussion groups.

Walter Schmidt, Font selection in LaTeX
(Published in this issue.)

Gerben C. Th. Wierda, Thomas A. Schmitz and Adam T. Lindsay, Mac OS X fonts in pdfTeX

Installing a new font with your TeX installation can be a challenging task. This article documents an attempt to provide an automated solution for users running TeX on Apple's OS X. Using the fonts described here will only be possible for those who run this operating system; the way of making these fonts work with TeX should be of interest for all users. The article's level can be described as intermediate to advanced; it assumes some previous knowledge of TeX and fonts.

Mike Spivak, The MathTime Professional Fonts: Or, How I Wasted the Last Twenty Years of My Life

I am a computer innocent who, through a series of historical accidents, ended up writing the `amstex` macro package, and a font design innocent who, through desperation for fonts that I was willing to use for my *Calculus* and *Differential Geometry* books, ended up designing the MathTime Professional fonts. It is a sobering reflection that these activities seem to have occupied a significant amount of my time during the last 20–25 years.

Will Robertson, An exploration of the Latin Modern fonts
(An updated version appears in this issue.)

Helmer Aslaksen, Chinese TEX: Using the CJK LATEX package, Unicode TrueType fonts and pdfTEX under Windows

The goal of this article is to help users with no past experience with CJK (Chinese, Japanese, Korean) typesetting to include some pieces of CJK text in a TEX document using the CJK LATEX package, TrueType fonts and pdfTEX under Windows.

Stephen Hartke, A survey of free math fonts for TEX and LATEX

We survey free math fonts for TEX and LATEX, with examples, instructions for using LATEX packages for changing fonts, and links to sources for the fonts and packages.

David Walden, Travels in TEX Land: Using the Lucida fonts

This paper describes buying, installing, and beginning to use the Lucida fonts. Then it describes some more exploration in the world of fonts in the context of the Lucida fonts.

The Editors, Ask Nelly:

- Which fonts can be accessed from the TEX Live distribution just using usepackage?
- What is the difference between fonts and typefaces?
- Which are the best fonts for typesetting math?

The Editors, Distractions: Name-that-Font

The PracTEX Journal 2006-2, 2006-05-17

Lance Carnes, From the editor

From the readers, Feedback

Robin Laakso, Invitation to PracTEX'06

Thomas A. Schmitz, Presentations in ConTEXt

TEX is an excellent tool to produce PDF presentations. This paper will show you how to use ConTEXt for writing presentations, and it will teach you to prepare a single source file that can output a presentation, a lecture manuscript, or a handout, if you adapt one single switch. The article is suitable for beginners in ConTEXt but it should also have interesting things for more advanced users.

Jan Hlavacek, Ipe — A graphics editor for LATEX

This article talks about the author's experiences with the Ipe graphics editor. This graphics tool, which runs on Windows, Linux, and Mac OS X, is well-suited to preparing graphics for LATEX documents. The main features of Ipe are described, as well as some advanced usage. A similar editor, VRR, is also described briefly.

Stephen J. Eglen, Introduction to "A short example of how to use LATEX for scientific reports"

This short article summarises my reasons for writing a short LATEX document to act as a template for scientific reports. This document was used as a basis of practical sessions with Masters students who wished to learn to use LATEX for their reports. The outcome was quite successful in that many more students are now using LATEX for their reports, often using the document as a template.

D.V.L.K.D.P. Venugopal, My experience with learning and teaching LATEX

The present article deals with the author's experience in learning LATEX independently and disseminating the knowledge acquired through a one week structured course to research scholars in a university system.

Andy Roberts, In My Opinion: LATEX isn't for everyone but it could be for you (with responses)

This article by Andrew Roberts appeared last summer in `OSNews.com`. While the merits of LATEX Andy points out are familiar to most PracTEX Journal readers, the follow-on comments from readers pointed out some of LATEX's (and TEX's) weaknesses.

David Walden, Travels in TEX Land: LATEX for productivity in book writing

My column in this issue summarizes why I use LATEX and gives examples of some productivity benefits of using LATEX to write books.

The Editors, Ask Nelly:

- What is different when I click on the pdfLATEX rather than the LATEX icon in WinEdt?
- How do I convert a document to a publisher's requirements for double-spacing, line numbers, and figures on their own pages?
- How do I interrupt an enumerate environment and then continue it later in the document?

The Editors, Distractions: Sudoku ABC; Winners of type quizzes

The PracTEX Journal 2006-3, 2006-08-15

Dave Walden, From the editor

From the readers, Feedback

Compiled by Dave Walden, Report on PracTEX'06

Will Robertson, Productivity with macros and packages

LATEX's advantages in productivity, for me, are due to its ability to be customised. The first half

of this article discusses small macros written to ease document production, with some examples of how I use macros to save time and effort. Then, I briefly cover a selection of packages that provide a whole heap of functionality that other people have kindly implemented.

Yuri Robbers, Markus Kohm and Rasmus Pank Roulund, Replacing LaTeX 2ε standard classes with KOMA-Script

KOMA is a complete replacement of the standard LaTeX 2ε classes. It is aimed more at European typography, but is easily configurable. Some of KOMA's extensions and ways to configure document layout are also available in other document classes, such as the LaTeX 2ε standard. This paper discusses just a few parts of KOMA, especially those that have to do with page layout and with writing letters.

Peter Wilson, The memoir class
(An updated version appears in this issue.)

Didier Verna, LaTeX curricula vitae with the CurVe class

CurVe is a LaTeX 2ε class for writing curricula vitae (cv). It provides a set of commands to create headers, rubrics, entries in these rubrics, etc. CurVe will then format your cv with a consistent layout while you can just concentrate on the contents. The layout of a CurVe cv is highly customizable. CurVe also has a very special feature known as the flavor mechanism: it is able to manage different "flavors" (versions) of your cv simultaneously. CurVe is distributed under the terms of the LPPL. This paper describes the features available in version 1.11.

Joe Hogg, ConTeXt starters

This article presents two beginning projects using ConTeXt. The first project is a letterhead that can be used for business and personal correspondence mailed in an envelope with an address pane. The second project is a four-page brochure done by the Botany Committee for Ape Awareness Day last November at the Los Angeles Zoo and Botanical Gardens.

D.V.L.K.D.P. Venugopal, Creating pocket-size books using LaTeX

This article deals with creating pocket-sized books of A7 size using LaTeX in a quick and dirty method.

David Walden, Travels in TeX Land: Experiences refining page layout for a book

In this column I describe my experience with taking final steps of turning a book manuscript into a published book.

D.V.L.K.D.P. Venugopal, Book review: *Formatting Information — A beginner's introduction to typesetting with LaTeX*, by Peter Flynn

The reviewer sees Peter Flynn's *Formatting Information* as a more practical introduction to LaTeX than many other popular introductions.

The Editors, Ask Nelly:

- What are the differences among MiKTeX, ProTeXt, and LaTeX?
- How do I add change bars?

The Editors, Distractions: LaTeX wordplay — 3 crosswords; math font quiz answers

The PracTeX Journal 2006-4, 2006-11-30

Lance Carnes, From the editor

From the readers, Feedback

The Editors, News from around

Sindhu Singh, Our Introduction to LaTeX

This report presents the experiences of three participants in a LaTeX course given by D.V.L.K.D.P. Venugopal at Banaras Hindu University in India (see `http://tug.org/pracjourn/2006-2/venugopal`). This popular course, which runs for one week, was conducted three times in the year 2006. The three participants discuss their initial reaction to LaTeX, and compare it to other formatting systems.

Jim Hefferon, What I wish I had ... when I was a lad: Using LaTeX resorces
(Published in *TUGboat* 28:1.)

Peter Flynn, Rolling your own Document Class: Using LaTeX to keep away from the Dark Side
(Published in *TUGboat* 28:1.)

Boris Veytsman, Design of presentations: Notes on principles and TeX implementation
(Published in *TUGboat* 28:1.)

Jürgen Fenn, Managing citations and your bibliography with BibTeX

This article gives a brief introduction to managing citations and to preparing a list of references with BibTeX. Techniques for writing a bibliography file and its use in a document are presented for first-time BibTeX users. Strategies and tools for simplifying work are also described. No attempt, however, is made to provide an in-depth introduction. The article concludes with a critical note on the future of BibTeX and a list of references for further reading.

Elizabeth Dearborn, TeX and medicine
(Published in *TUGboat* 28:1.)

Boris Veytsman and Leila Akhmadeeva, Drawing medical pedigree trees with TEX and PSTricks
(Published in *TUGboat* 28:1.)

Paul Blaga, Using XY-pic

This is the first of two papers aiming to describe the use of the facilities of the package XY-pic for constructing commutative diagrams. We tried to use in a systematic way the "learning by examples" approach, without entering into the details of different constructions or trying to describe in an exhaustive way all the possibilities. The final goal is to provide the reader with enough knowledge to be able to construct by himself complicated diagrams. This first paper describes the basic possibilities of XY-pic, which are provided by the kernel of the associated language, but it also explores many of the opportunities provided by the extension arrow.

Federico Garcia, Capabilities of PDF interactivity
(An updated version appeared in *TUGboat* 28:1.)

Troy Henderson, A beginner's guide to MetaPost for creating high-quality graphics
(Published in *TUGboat* 28:1.)

Aditya Mahajan, Creating homework assignments using ConTEXt

This article shows how to create a ConTEXt environment file to typeset homework assignments and their solutions. The same source file can be used to generate two versions: without and with the solutions.

Lars Madsen, Avoid `eqnarray`!

Whenever the `eqnarray` environment appears in a question or an example of a problem on the `comp.text.tex` newsgroup or the `texhax` mailing list there is a good chance that someone will tell the poster not to use `eqnarray`. This article will provide some examples of why many of us consider `eqnarray` to be harmful and why it should not be used.

David Walden, Travels in TEX Land: Using your favorite editor with TEX

In this column in each issue Dave Walden muses on his wanderings around the TEX world. In this column he is joined in his meandering by Yuri Robbers. Walden discusses the benefits of the fact that one can use the editor of one's choice with TEX and the various system built on top of TEX, and Robbers lists a number of editors that are optionally available.

Dimitrios Filippou, Book review: A. Syropoulos et al., *Digital Typography Using LATEX*

The book *Digital Typography Using LATEX* was published by Springer-Verlag in 2002, but has been mostly ignored by the LATEX community. However, Syropoulos and his co-authors have put lots of effort in creating a very good guide for the novice and the aspiring TEXnician. Despite some objections on how the book is structured, it can be said that *Digital Typography Using LATEX* is a very good, if not excellent, everyday guide for producing beautiful documents.

The Editors, Distractions: Two TEX-themed crossword puzzles

The PracTEX Journal 2007-1, 2007-02-20

Yuri Robbers, From the editor

From the readers, Feedback

The Editors, News from around

Claudio Beccari, Graphics in LATEX

This tutorial describes some facilities offered by LATEX and its extension packages for producing line art graphics directly in the source document. Some of these facilities are standalone, in the sense that they do not require functionalities of external programs, while others rely on external programs.

Andrew Mertz and William Slough, Graphics with PGF and Ti*k*Z
(Published in *TUGboat* 28:1.)

Dirk Baechle, Square concepts

The following text contains some basic chess concepts and advice, presented in the form of tasks. It was prepared [...] using the program ChessTask, and is aimed at hobby players of any strength. The given positions are all taken from my own games for the chess club SK Soltau, so they are not constructed examples but actually happened. While the tasks themselves are kept rather terse, I try to elaborate things a bit further when I unveil the correct moves at the end of the article. So even if you did not find the solution in first place, you can hopefully learn from the additional information.

Yuri Robbers and Annemarie Skjold, Creating book covers with PSTricks

The title and the cover of the book are the very first impressions a potential reader is likely to get from a book. It is therefore of utmost importance to make sure these impressions are good ones. This paper will give some general notes on cover design,

and some specific notes on implementing such designs using PSTricks.

Paul Blaga, Commutative diagrams with Xy-pic — II. Frames and matrices

This is the second article dedicated, essentially, to the use of Xy-pic for constructing commutative diagrams. By using the same kind of approach as in the first part, we focus now on frames, matrices and other extensions of the kernel language.

Manjusha S. Joshi, Create trees and figures in graph theory with PSTricks

Drawing trees and figures in the mathematical area of graph theory is a requirement for researchers and teachers. This includes loops, arcs, nodes, and weights for edges. This article aims to get started with PSTricks by keeping two commands in mind, viz. `pstree` and `psmatrix`. Using the most useful options of these commands the reader can draw tree diagrams, loops, node labels, and add weights to edges. Once the diagrams are completed they can be added to a TEX file. With a little working knowledge about drawing figures in graph theory the reader can then produce his or her own.

Lapo Filippo Mori, LaTeXpedia: The future of LaTeX documentation

Software documentation is a very important success factor for open source software because it bolsters its diffusion. People who start learning LaTeX and even intermediate users often complain about LaTeX documentation: it is hard to find an updated, complete and well structured resource. This article evaluates advantages and disadvantages of the different sorts of resources for LaTeX documentation available and proposes a new kind of documentation source: a free-content, web-based encyclopedia, LaTeXpedia.

Lapo Filippo Mori, Tables in LaTeX 2_ε: Packages and methods

This article aims to provide the tools to correctly create tables in LaTeX 2_ε. This objective is pursued analyzing the typical problems that users find creating tables and possible solutions; particular emphasis is on the packages to be used. Examples are given for each case.

Enrico Gregorio, Babel, how to enjoy writing in different languages
(Published in this issue.)

Jin-Hwan Cho, Hacking DVI files
(Published in this issue.)

Bob Neveln and Bob Alps, Writing and checking complete proofs in TEX
(Published in *TUGboat* 28:1.)

David Walden, Travels in TEX Land: The post-typesetting phase of producing a book

In this column I give a final report on the book project I last discussed in TPJ issue 2006-3, discuss "self-publishing" at some length, and mention some of my other recent activities in TEX Land.

S. Parthasarathy, The "hacking for learning" paradigm in LaTeX — Some thoughts by a long-time LaTeX user

This article argues a case for making hacking an accepted way of learning. It uses the example of LaTeX to show why hacking is not so bad as it is made out to be. It also gives some warnings on the downside of hacking-for-learning.

The Editors, Ask Nelly: How do I create European style spacing within numbers?

The Editors, Distractions: Some chess problems created in LaTeX

***The PracTEX Journal* 2007-2, 2007-06-10**

Paul Blaga, From the editor

From the readers, Feedback

The Editors, News from around

Paul Blaga, PracTEX Journal: Making an electronic journal with web tools, wiki, and version control

This paper describes how an issue of TPJ is made up, and focuses on the software tools used. The tools used are the wiki, the subversion version control system, and Perl scripts. One of the goals of TPJ, and the reason for using these tools, is to create an environment where a small team of volunteers can put together an online journal with a minimum of time and work.

Yuri Robbers, PDFpages for editors and publishers

There are many ways in which the PDFpages package by Andreas Matthias can be helpful to editors. An obvious one is collating several papers into one. This paper will describe a few of the many ways in which PDFpages can make life easy for the editor and publisher.

Federico Garcia, LaTeX and the different bibliography styles
(Published in this issue.)

S. Parthasarathy, Demystifying LaTeX bibliographies

In this essay, we will try to explore and explain the vexing problem of including bibliographic references in LaTeX documents (reports, papers, theses, etc.). There is a plethora of literature on this subject. Unfortunately, these materials are focused on LaTeX's experts, and driven to a developer-level point of view. This current paper will examine bibliographies from a common user's point of view, trying to pass by only the essentials of this very vast and involved bibliographic topic. The author hopes that this paper will make LaTeX enjoyable for more people.

David Walden, Travels in TeX Land: Trying ConTeXt

In this column I focus on my initial efforts to learn and use ConTeXt.

The Editors, Ask Nelly:

- How do I create footnotes to tables without developing an ulcer in the process?
- What is a good way to create subfigures within one float?
- How do I typeset a critical edition?

The Editors, Distractions — From Shakespeare, with Love

Institutional Members

Aalborg University, Department of Mathematical Sciences, *Aalborg*, *Denmark*

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

Banca d'Italia, *Roma*, *Italy*

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

Certicom Corp., *Mississauga*, *Ontario Canada*

CNRS - IDRIS, *Orsay*, *France*

CSTUG, *Praha*, *Czech Republic*

Florida State University, School of Computational Science and Information Technology, *Tallahassee*, *Florida*

IBM Corporation, T J Watson Research Center, *Yorktown*, *New York*

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

MacKichan Software, *Washington/New Mexico*, *USA*

Marquette University, Department of Mathematics, Statistics and Computer Science, *Milwaukee*, *Wisconsin*

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

Moravian College, Department of Mathematics and Computer Science, *Bethlehem*, *Pennsylvania*

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

Princeton University, Department of Mathematics, *Princeton*, *New Jersey*

Springer-Verlag Heidelberg, *Heidelberg*, *Germany*

Stanford Linear Accelerator Center (SLAC), *Stanford*, *California*

Stanford University, Computer Science Department, *Stanford*, *California*

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

University College, Cork, Computer Centre, *Cork*, *Ireland*

University of Delaware, Computing and Network Services, *Newark*, *Delaware*

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

Universiti Tun Hussein Onn Malaysia, Pusat Teknologi Maklumat, *Batu Pahat*, *Johor*, *Malaysia*

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

Vanderbilt University, *Nashville*, *Tennessee*

Calendar

2007

Jun 4 – Aug 3 — Rare Book School, University of Virginia, Charlottesville, Virginia. Many one-week courses on type, bookmaking, printing, and related topics. For information, visit http://www.virginia.edu/oldbooks.

Jun 8 — NTG 39th meeting, Utrecht, Netherlands. For information, visit http://www.ntg.nl/bijeen/bijeen39.html.

Jun 18 – Jul 27 — Guild of Book Workers 100th Anniversary Exhibition: A traveling juried exhibition of books by members of the Guild of Book Workers. The Bridwell Library, Southern Methodist University, Dallas, Texas. Sites and dates are listed at http://palimpsest.stanford.edu/byorg/gbw.

TUG 2007
Practicing TEX,
San Diego, California.

Jul 17 — Workshops (free for attendees).

Jul 18 – 20 — The 28th annual meeting of the TEX Users Group. For information, visit http://www.tug.org/tug2007.

Aug 1 – 5 — TypeCon 2007, Seattle, Washington. For information, visit http://www.typecon.com/.

Aug 5 – 9 — SIGGRAPH 2007, San Diego, California. For information, visit http://www.siggraph.org/s2007/.

Aug 6 – 10 — *Extreme* Markup Languages 2007, Montréal, Québec. For information, visit http://www.extrememarkup.com/extreme/.

Aug 28 – 31 — ACM Symposium on Document Engineering, University of Manitoba, Winnipeg, Canada. For information, visit http://www.documentengineering.org/.

Sep 1 – 2 — Transylvania TEX Conference, "Babeş-Bolyai" University Cluj-Napoca, Romania. For information, visit http://math.ubbcluj.ro/~aga_team/translatex/.

Sep 12 – 16 — Association Typographique Internationale (ATypI) annual conference, Brighton, UK. For information, visit http://www.atypi.org/.

Sep 15 — DANTE TEX-Tagung, 38th meeting, Universität Ulm, Germany. For information, visit http://www.dante.de/dante/events/mv37/.

Sep 18 – 19 — Conference on "Non-Latin typeface Design", St. Bride Library, London, and the Department of Typography, University of Reading, UK. For information, visit http://stbride.org/events_education/events/.

Sep 24 – Nov 22 — Guild of Book Workers 100th Anniversary Exhibition: A traveling juried exhibition of books by members of the Guild of Book Workers. Dartmouth College Library, Hanover, New Hampshire. Sites and dates are listed at http://palimpsest.stanford.edu/byorg/gbw.

Oct 8 — GUTenberg Workshop on Unicode & LATEX, Paris, France. For information, visit http://www.gutenberg.eu.org/.

Oct 11 – 13 — American Printing History Association 2007 annual conference, "Transformations: The persistence of Aldus Manutius", University of California at Los Angeles. For information, visit http://www.printinghistory.org/.

Oct 13 — GuIT meeting 2007 (Gruppo utilizzatori Italiani di TEX), Pisa, Italy. For information, visit http://www.guit.sssup.it/GuITmeeting/2007/.

Oct 20 – 22 — The Fifth International Conference on the Book, "Save, Change or Discard: Tradition and Innovation in the World of Books", Madrid, Spain. For information, visit http://b07.cgpublisher.com/.

2008

TUG 2008 — TEX's 30th birthday
University College Cork, Ireland.

Jul 21 – 24 — The 29th annual meeting of the TEX Users Group. For information, visit http://www.tug.org/tug2008.

Status as of 1 June 2007

For additional information on TUG-sponsored events listed here, contact the TUG office (+1 503 223-9994, fax: +1 206 203-3960, e-mail: office@tug.org). For events sponsored by other organizations, please use the contact address provided.

An updated version of this calendar is online at http://www.tug.org/calendar/.

TUG 2007: Practicing TEX

Workshops and presentations on

LATEX, TEX, MetaPost, ConTEXt, LuaTEX,

and more

July 17–20, 2007

San Diego State University
San Diego, California, USA

http://tug.org/tug2007
tug2007@tug.org

Keynote address: *Peter Wilson,*
The Herries Press

Further information

Conference attendees will enjoy an opening night reception and an (optional) banquet one evening. Coffee and lunch will be served each day of the meeting. Located on the campus of San Diego State University, an easy trolley ride from downtown San Diego. Inexpensive campus housing is available.

Conference fee, hotel, and other information is available on the web site.

Sponsorship

We thank the sponsors: the German-speaking TEX users group DANTE e.V., von Hoerner & Sulger GmbH, MacKichan Software, and Adobe Systems Inc. have provided generous support; San Diego State University is our host; and special thanks to the many individual contributors.

If you'd like to support the conference, promote TEX products and services, or otherwise provide sponsorship, see the web site for donation and advertising options.

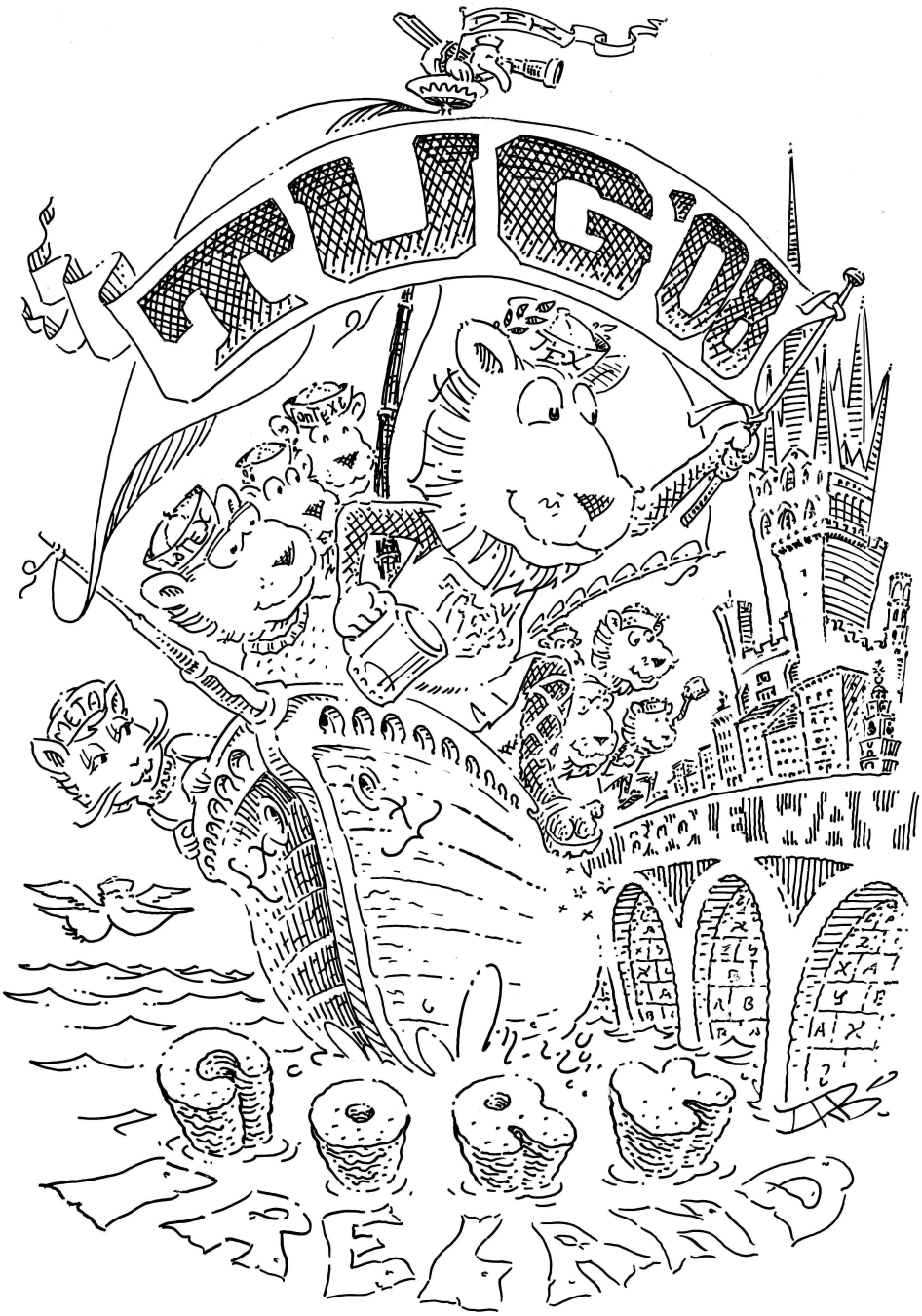
Hope to see you there!

Sponsored by the TEX Users Group

TEX Users Group 2008 Conference

University College Cork
Cork, Ireland
21–24 July 2008
http://tug2008.ucc.ie/

TEX's 30th birthday
Interfaces to TEX
Workshops
Presentations



Hosted by the Human Factors Research Group (http://hfrg.ucc.ie)

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 fill out the form at `https://www.tug.org/consultants/listing.html` or email us at `consult-admin@tug.org`. To place a larger ad in *TUGboat*, please see `http://tug.org/TUGboat/advertising.html`.

Kinch, Richard J.
7890 Pebble Beach Ct
Lake Worth, FL 33467
+1 561-966-8400
Email: `kinch (at) truetex.com`
Publishes TrueTEX, a commercial implementation of TEX and LATEX. Custom development for TEX-related software and fonts.

Martinez, Mercè Aicart
Tarragona 102 4^o 2^a
08015 Barcelona, Spain
+34 932267827
Email: `m.aicart (at) menta.net`
Web: `www.edilatex.com/`
We provide, at reasonable low cost, TEX and LATEX 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.

Peter, Steve
310 Hana Road
Edison, NJ 08817
+1 (732) 287-5392
Email: `speter (at) dandy.net`
Specializing in foreign language, linguistic, and technical typesetting using TEX, LATEX, and ConTEXt, I have typeset books for Oxford University Press, Routledge, and Kluwer, and have helped numerous authors turn rough manuscripts, some with dozens of languages, into beautiful camera-ready copy. I have extensive experience in editing, proofreading, and writing documentation. I also tweak and design fonts. I have an MA in Linguistics from Harvard University and live in the New York metro area.

Veytsman, Boris
2239 Double Eagle Ct.
Reston, VA 20191
+1 (703) 860-0013
Email: `borisv (at) lk.net`
Web: `http://borisv.lk.net`
TEX and LATEX consulting, training and seminars. Integration with databases, automated document preparation, custom LATEX packages, conversions and much more. I have about twelve years of experience in TEX and twenty-five years of experience in teaching & training. I have authored several packages on CTAN and published papers in TEX related journals.

TUGBOAT Volume 28 (2007), No. 2

Table of Contents (ordered by difficulty)

Introductory

Intermediate

Intermediate Plus

Advanced

Contents of other TEX journals

Reports and notices

TUGboat

Volume 28, Number 2 / 2007

TUGBOAT Volume 28 (2007), No. 2

Table of Contents (ordered by difficulty)

Introductory

Intermediate

Intermediate Plus

Advanced

Contents of other TEX journals

Reports and notices