TUGboat

Volume 29, Number 2 / 2008

TEX Users Group

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

Memberships and Subscriptions

2008 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/`

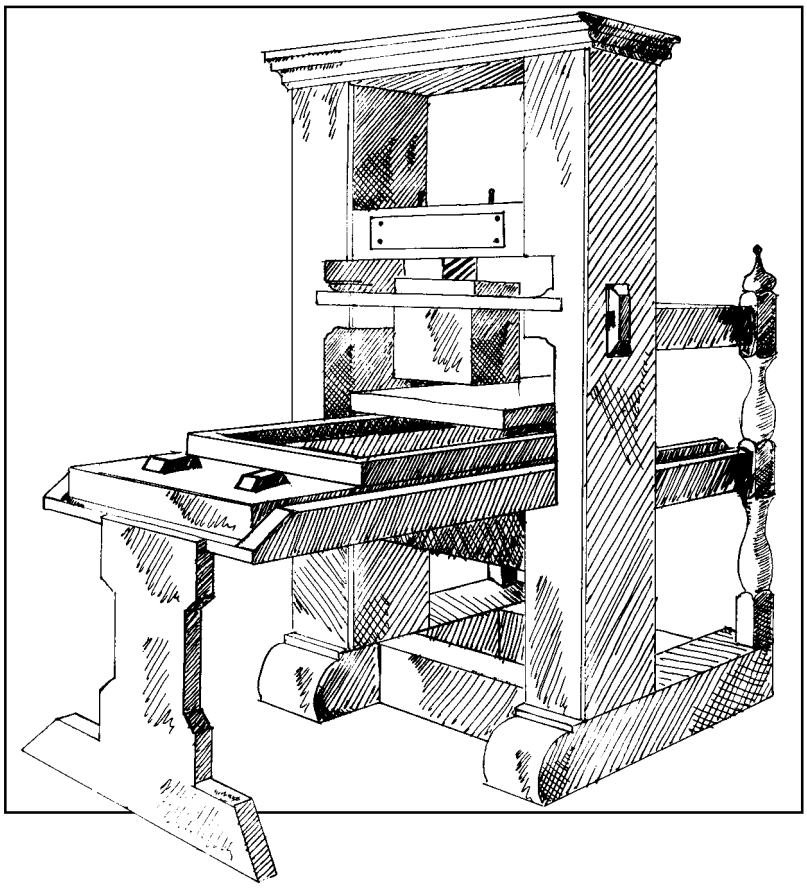
Have a suggestion? Problems not resolved?

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

[printing date: June 2008]

TUGBOAT

The Communications of the TEX Users Group



Volume 29, Number 2, 2008

TEX Users Group

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

Memberships and Subscriptions

2008 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: June 2008]

The *problem* is what interests all but beginners in typography. Its solution may be, and often is, moderately exciting; although if the problem is successfully solved no one perceives it has existed.

Daniel Berkeley Updike
Notes on the Merrymount Press
(1934)

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP
EDITOR BARBARA BEETON

VOLUME 29, NUMBER 2 • 2008
PORTLAND • OREGON • U.S.A.

TUGboat

This regular issue (Vol. 29, No. 2) is the second issue of the 2008 volume year. No. 1 was a joint publication of the EuroBachoTEX 2007 conference, and No. 3 will contain the TUG 2008 (Cork II) 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

The deadline for receipt of the final papers for the upcoming proceedings issue is August 15, 2008. Links, locations, and more information about this and all conferences are available at `http://tug.org/meetings.html`.

The next regular issue will probably be in spring 2009. 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.

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 pages:
`http://tug.org/TUGboat/advertising.html`
`http://tug.org/consultants.html`

Trademarks

Many trademarked names appear in the pages of *TUGboat*. If there is any question about whether a name is or is not a trademark, prudence dictates that it should be treated as if it is. The following list of trademarks which 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 $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX are trademarks of the American Mathematical Society.

General Delivery

From the President

Karl Berry

TUG at the 2008 Joint Mathematics Meeting

2008 started with a bang, as we were generously sponsored by the American Mathematical Society to have a booth at the (huge) JMM in San Diego this January. Robin Laakso, our always-hard-working executive director, and I staffed the booth, with TUG member Stephen Hartke joining us for some of the time. It was inspiring to meet so many people who happily use TEX every day in their work, be able to answer a few questions, and even sign up new members.

Google Summer of Code 2008

Another notable event in the first part of 2008 was TUG being chosen to participate in Google's 2008 Summer of Code. In this program, Google funds full-time work by students over the summer on free software projects. TUG was selected by Google to be one of 175 "mentoring organizations", providing an infrastructure and pairing students with mentors.

We were given three slots, and chose the following three projects from among the (generally excellent) applications:

- *Better Unicode compliance for TEX extensions*, by Arthur Reutenauer, mentored by Eric Muller.
- *Overhaul of the texshow documentation system*, by Mojca Miklavec, mentored by Taco Hoekwater.
- *Improved JavaScript support in MathTran*, by Christoph Hafemeister, mentored by Jonathan Fine.

`http://tug.org/gsoc` has more details and links about the proposals and the overall program.

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:

- Susan DeMeritt and Cheryl Ponchin are both members of the TUG board of directors (Sue is secretary of the board), give workshops on LATEX, and use TEX daily in their technical typing work.
- Peter Gordon is the editor at Addison Wesley for Donald Knuth's books and the company's (LA)TEX books.
- Jim Hefferon is one of the key maintainers of CTAN and a member of the TUG board.
- Ross Moore is a long-time TUG board member and TEX contributor, especially in the areas of mathematics and Unicode support.
- Oleg Katsitadze maintains Eplain and contributes to GNU Texinfo.
- Dick Koch is the creator and lead developer of TeXShop, a popular front end for TeX on Mac OS X, as well as being a new TUG board member.
- Rainer Schöpf was a co-founder of the LATEX 2_ε and CTAN projects, and is still a key maintainer of CTAN.

We have begun work on a printed volume of all the interviews, to be published by TUG perhaps by early 2009.

The PracTEX Journal

Issue 2008-1 of *The PracTEX Journal* was published in April. The theme of this issue is LATEXniques. All the articles and all the back issues (the journal started in 2005) are available on the journal web site, `http://tug.org/pracjourn`. As always, new articles and feedback of any kind is welcome.

Conferences

The TUG 2008 conference will take place in Cork, Ireland (for the second time!), from July 21–24 with workshops on July 20, hosted by the University of Cork; Peter Flynn is the principal local organizer. For the schedule, registration and accommodation information (inexpensive on-campus housing is available), and more, visit the conference web site at `http://tug.org/tug2008`. Please join us and the excellent line-up of workshops and speakers.

The second annual ConTEXt user conference will take place from August 20–25, 2008, in Bohinj, Slovenia, around a lake in the Alps. Web site: `http://meeting.contextgarden.net/2008`.

Looking ahead, TUG 2009 will take place at the University of Notre Dame, in South Bend, Indiana, from July 27–31, 2009, sponsored by the *Notre Dame Journal of Formal Logic*, with Martha Kummerer being the principal local organizer.

Hope to see you on both sides of the Atlantic!

◇ Karl Berry
`president@tug.org`

Editorial comments

Barbara Beeton

TEX 3.1415926 is here, and other Knuthian references

Don Knuth has completed his periodic review of bug reports, and has released the new versions of all programs, fonts and macros. His summary of the changes made in this round appears later in this issue.

The period between reviews increases by a year with each round. The review just completed was actually begun in 2007, after a five year hiatus. The next review is scheduled for six years from then — 2013. For the present, I will continue to act as Don's "TEX entomologist". However, I may relinquish this post before the next cycle ends; any change will be announced here, and posted on Don's TEX web page: `http://www-cs-faculty.stanford.edu/~knuth/abcde.html`.

Ulrik Vieth noted, at the announcement of the new release, that "we all silently missed the 30 year anniversary of TEX78 which started on March 10, 1978." He also reminds us that the saga of TEX development and updates is chronicled in the file `errorlog.tex`, which "is still an interesting read for the historians". Thanks, Ulrik.

Another note, from Nelson Beebe, points to an interview with Don at `http://www.informit.com/articles/article.aspx?p=1193856&rll=1`. Quoting Nelson, "Don talks about TEX, METAFONT, literate programming, and the writing of his famous books, *The Art of Computer Programming*, and responds to a number of questions about software development and tools, and the trend towards multicore and multithreaded CPUs."

And finally, Don appears in the comics: `http://xkcd.com/163/` and `xkcd.com/342/`.

Phyllis Winkler, RIP

Phyllis Winkler, Don Knuth's secretary for many years, passed away on December 7, 2007. She was much more than Don's secretary. It was for her, as well as for himself, that Don created TEX. (We announced her retirement in *TUGboat* 19, no. 4, page 351.) Phyllis was kind, generous, and strikingly competent. The TEX community has lost a good friend.

New domain name for CervanTEX

Late last year, the president of CervanTEX, Juan Luis Varona, informed us that their domain name, `cervantex.org` was stolen, and held for ransom, which, as a small group, they are unable to pay. Although they would greatly prefer to have an `.org` name, showing that they represent TEX for everyone who writes in Spanish, they have had to make an accommodation.

Their new site is `http://www.cervantex.es`. They ask everyone who may have a link to the old name to please update it.

Interactive typography courses by Jonathan Hoefler

The Resources page at the Typophile web site (`http://www.typophile.com/resources`) always lists interesting material. Two items of particular interest are "Typography 101" (Type Styles) and "Typography 110" (Typeface Design), interactive courses by Jonathan Hoefler.

The first course presents images of well-known typefaces, along with text placing each face in its appropriate design category and historical environment. Related faces are identified for further exploration. In the second course, the participant is asked to experiment with irregular shapes that illustrate the behavior of lettershapes (while not being themselves letters) in order to understand principles such as balance and typographic color; the shapes are then transformed to be styled like some of the typefaces presented in the first course, paying attention to the design principles that characterize these faces.

Another feature at Typophile is an extensive forum on Typographic Education. Look for `http://typophile.com/forum/16`.

A helpful CTAN feature: "get"

Although it has most likely been there for quite a while, I just learned about the "get" feature of CTAN. If you are reasonably certain of the location of files you are looking for, the command `www.ctan.org/get/` presents a nicely formatted directory list of the archive's top level, and adding a directory name takes you to that area directly, using an available mirror to spread the load. This is a convenient alternative to the search page.

Thanks to the CTAN crew whose efforts are always appreciated, if not acknowledged nearly often enough.

Recreating the Gutenberg press

In April, a documentary video was aired by the BBC chronicling the invention of the Gutenberg Press. This description was posted by RIT Professor Emeritus Michael L. Kleper.

> Stephen Fry travels to France and Germany on the trail of Johannes Gutenberg, and sets about reconstructing a replica of Gutenberg's first press. This is a must-see program. Owing to contractual restrictions, it's not available outside of the UK.

For awhile, it was posted on YouTube, but pulled in response to the BBC's copyright claim. Watch for it — it may reappear, or be released elsewhere. Before it disappeared, I managed to view most of it; Prof. Kleper's description is spot on!

Copy-editing the wayward apostrophe

Do you become exasperated reading signs in which words are (mis-)spelled in sometimes seemingly random ways, and apostrophes appear in places where they don't belong, often changing the meaning of what is being advertised?

Well, you're not alone, and someone is doing something about it.

Jeff Deck, a Dartmouth College (New Hampshire, USA) graduate, is on a mission to correct typographical errors in public places. You can read about his quest at `http://abcnews.go.com/Travel/BusinessTravel/story?id=4593597&page=1`.

He also writes a blog (`http://www.jeffdeck.com/teal/blog/`) where he ruminates on the activities of the Typo Eradication Advancement League (TEAL).

Join the hunt.

A font game for your amusement

At `http://fontgame.ilovetypography.com`, one finds a page entitled "the rather difficult Font Game".

Below this heading appears a small font sample — a word, a date, or other string of glyphs from a font — with four font names; choose the one you think the sample came from.

The game consists of 34 such samples, and it keeps score. When you have identified all 34, you can look at the correct answers, check the scores of other testees (the average score is reported to be 23 out of 34 points), or start again with new samples.

The game was created by Kari Pätilä of Joutseno, Finland. It can be played on an ordinary browser or on an iPhone.

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

Editor's note: The following note announced the periodic update of TEX and friends on March 18, 2008. (In this presentation, some of the verbatim code lines have been re-broken or otherwise adjusted to fit the narrow columns.) Work is underway to include the new versions in the next release of TEX Live. Unless you have experience in implementing the software, it's advisable to wait for that release. Please refrain from posing questions or reporting problems, to allow the implementors to pursue their work without interruptions. If you want to find out the details of the updates, those can be found on CTAN in the area `systems/knuth/errata/`.

The TEX tuneup of 2008

Donald Knuth

I've written this note while going through the long, long file of bug reports and suggestions that were submitted during the years 2003–2007. You know that I am committed to keeping TEX and METAFONT as stable as possible, while also correcting serious blunders that are likely to be harmful if left as is. It is certainly not always obvious where to draw the line; I intend to keep drawing it as close to the existing implementations as I can, without feeling extremely guilty.

The index to *Digital Typography* lists eleven pages where the importance of stability is stressed, and I urge all maintainers of TEX and METAFONT to read them again every few years. Any object of nontrivial complexity is non-optimum, in the sense that it can be improved in some way (while still remaining non-optimum); therefore there's always a reason to change anything that isn't trivial. But one of TEX's principal advantages is the fact that it does not change — except for serious flaws whose correction is unlikely to affect more than a very tiny number of archival documents.

Let me give two examples. First, David Kastrup observes that TEX doesn't do the best possible rounding when it converts units. One inch is exactly 72.27 points, which is exactly 4736286.72 scaled points. When you say '`1in`', TEX converts it to 4736286sp; when you say '`72.27pt`', TEX converts it to 4736287sp, which is about 23.6 Ångstrom units closer to the truth. With a simple change to TEX §458, namely to add '`denom div 2`' before dividing by '`denom`', the rounding would be slightly better. But that would invalidate the line-break and page-break decisions of an enormous number of documents. It's unthinkable to change TEX in such a way today.

But of course the authors of other systems should adopt superior methods when they want to.

Second, I recently installed METAPOST version 0.993, which corrected a bug in the calculation of the bounding box of its outputs. I'm a user of METAPOST, not a developer; but I'm sort of glad that the developers had fixed this bug. On the other hand it was a tremendous headache for me, because it affected nearly 200 of the illustrations for *The Art of Computer Programming*, and caused severe changes to the layouts of more than a dozen pages, even though the individual corrections to the box sizes were typically 2pt or less! I spent three days going over everything so that I could once again typeset the volumes of my main life's work. I couldn't reasonably insist that the METAPOST developers retain such a serious bug as a "feature". With TEX, on the other hand, it's a different story, because people's accumulated investment in TEX documents is more than a million times the total current investment in METAPOST documents. If a comparable bug had showed up in TEX, I would *not* have changed it.

Let me also observe that I never intended TEX to be immune to vicious "cracker attacks"; I only wish it to be robust under reasonable use by people who are trying to get productive work done. Almost every limit can be abused in extreme cases, and I don't think it useful to go to extreme pain to prevent such things. Computers have general protection mechanisms to keep buggy software from inflicting serious damage; TEX and METAFONT are far less buggy than the software for which such mechanisms were designed. For instances of the philosophy that I had while writing these programs, see for instance TEX §9 and MF §9, which say that I expected the programs to be run with arithmetic overflow interrupt turned on; also TEX §104: "TEX does not check for overflow when dimensions are added or subtracted ... the chance of overflow is so remote that such tests do not seem worthwhile"; MF §369 says that the total weight in a picture "will be less than 2^{31} unless the edge structure has more than 174,762 edges"; MF §558, "we shall assume that the coordinates are sufficiently non-extreme"; MF §930, "users aren't supposed to be monkeying around with really big values."

A proposal re file errors

I think the following change would be nice for the next versions of TEX, METAFONT, etc.: In place of the current message

```
Please type another %s file name:
```

produced by `prompt_file_name`, let's substitute

```
Please type another %s file name (or quit):
```

and then if the user's response is 'quit' we do the equivalent of control-C. If the response is null, let's give a help message.

This modification should be handled by change files, keeping the master files `tex.web` and `mf.web` and `whatever.web` as they are. I never have intended to control the aspects of user interaction on particular systems.

Maybe also introduce a finite loop, with '`(or quit)`' replaced by '`(or I'll quit)`' the third or fourth time. I agree that infinite loops are evil, and I'm sorry that `prompt_file_name` is invoked only within infinite loops in my own programs. If I had thought of this idea earlier, I'd have added a global variable like `max_prompt_repeats`, and initialized it to 3 or 4 just before those infinite loops; then `prompt_file_name` would decrement it, or give up if it's zero.

Another possibility is '`(or quit or retry)`', except the last time. That wording is a bit more suited to computer geeks, who have ideas about fixing things by repairing file permissions, etc.; if the user responds with either 'retry' or null, the intention is clearly to try again because of some reason to hope for success. Still, I prefer the non-geek version, because it reaches more people and enables the null-for-help option. Let the geeks type a few more keystrokes — they get satisfaction in other ways.

TEX

TEX version 3.1415926 corrects a few minor bugs, following major studies by David Fuchs. A summary of the noteworthy changes to the Pascal code in `tex.web` can be found near the end of the (long) file `errata/tex82.bug`. Here are the most significant ones, in decreasing order of importance:

1. Leaders with `\mskip` glue never worked properly; this feature has now been disallowed.

2. Error recovery was incorrect when an extra right brace appeared within a macro parameter.

3. TEX's inner loop now runs a bit faster.

4. The size of insert boxes is now displayed more accurately by `\showlists`.

5. A restriction on TFM files enforced by TFtoPL (namely that there must be at least one entry in each of the width, height, depth, and italic correction tables) is now enforced also by TEX, since noncompliance could cause a mess.

6. TEX used to leak four words of memory if arithmetic overflow occurred when `\multiply` or `\divide` was applied to `glue` or `muglue`.

7. The old `iniTeX` could leak four words of memory in another way (but at most four total), if "`last_glue`" pointed to a glue specification when the format file was created.

There's an undocumented feature, which is inconvenient to explain anywhere in *The TEXbook*: `\pagedepth` is cleared to zero when the current page disappears into `\box255`; but `\pagetotal`, `\pagestretch`, `\pagefilstretch`, `\pagefillstretch`, `\pagefilllstretch`, and `\pageshrink` are zeroed later, when the current page becomes nonempty. (That's the time `\pagegoal` is set, and recorded in the log file with a `%%` line if you're tracing pages.) I don't recall why there is a discrepancy, but I certainly don't want to diddle with any of that logic at this late date.

Here are some other things that I don't want to touch:

i. David Kastrup found a glitch in plain TEX's footnote-splitting mechanism. Everything works according to the documentation in *The TEXbook* and I can't possibly make a change to such a sensitive part of TEX's logic at this late date. But his example is quite interesting, and I'd like to discuss it here for the benefit of people planning other systems.

Here's his construction (to be used with plain TEX):

```
\def\testpage#1{\dimen0=#1
  \vrule height .5\dimen0 depth .5\dimen0
  \quad #1\par
  Some text.\footnote*{A bigbreak follows...
                      \bigbreak
                      A bigbreak preceded.}
  \par\vfill\supereject}
\testpage{8.17in}
\testpage{8.23in}
\testpage{8.2in}
```

The first test page is an example where the entire footnote fits fine. In the second one, the footnote needs to be split; so two pages are generated, one with the first half of the footnote, as desired.

The third test page illustrates the problem: Plain TEX uses the worst of both strategies! Namely, it generates two pages, in which the first is underfull, while the second has the text and footnote that would have fit on the first page.

Why does plain TEX screw up here? Well, TEX knows that the footnote doesn't fit, when typeset at its natural height+depth of 36pt. So it tries to split it, by choosing a height threshold: It says to the `vsplit` routine, "Please give me your best break that doesn't exceed a height of 30.089pt." (That is what's left after we start with plain TEX's vsize of 8.9in and subtract the page-total-so-far, which is 8.2in for the `vrule`, plus 1pt of `lineskip`, plus 7.5pt for the height of 'Some text.', plus 12pt to separate the text from its first footnote.) The `vsplit` algorithm discovers two ways to break the footnote: One has height 8.5pt (the height of '* A bigbreak follows...'), depth 1.94444pt, and penalty −200 (at the `bigbreak`); the other has natural height 32.5pt, depth 3.5pt (which comes from a strut placed by plain TEX), and penalty −10000 (the force-out penalty at the very bottom of the footnote). This latter break is considered viable because 4pt of glue shrinkage is available to bring the height down to 30.089pt. Naturally `vsplit` chooses the latter alternative.

Then TEX does something dumb. It records the result of the split in the list of contributions to the current page, in such a way that the first part of the split will be included on the page only if there's room for its natural height+depth, namely 36pt in this case. (And in this case, the "first part of the split" actually turns out to be the whole footnote.) Therefore, when TEX next finds a legal breakpoint, the current page limit has been exceeded, and the line with its footnote is deemed not to be permissible. The previous break, which leaves an underfull vbox, is chosen instead of "overfilling" the page — even though there is really enough shrinkability to bring the page back to size.

As I said, it's too late now to correct my age-old faulty reasoning. If I'd known about the problem twenty years ago, I may well have decided to make the change that seems most appropriate to me today, which is this:

```
@x module 974
    best_height_plus_depth:=cur_height+prev_dp;
@y
    best_height_plus_depth:=cur_height+prev_dp;
    if (best_height_plus_depth>h+prev_dp)
      and (b<awful_bad) then
      best_height_plus_depth:=h+prev_dp;
@z
```

In other words, the log file (with `tracingpages=1`) now gets the line

```
% split254 to 30.08878,36.0 p=-10000
```

but after that patch it would instead say

```
% split254 to 30.08878,33.58878 p=-10000
```

and the footnote would wind up on the first page where it belongs.

When I made the mistake ages ago, I probably wasn't thinking of shrinkability inside the footnote, only in the "virtual" amount of space within `\skip254` that separates the text from its footnotes. Indeed, the present problem goes away if one sets `\skip254=12pt minus 8pt`. But that workaround

would be appropriate only for this particular example.

ii. Section 798 could be made more robust with "`until q=cur_align`" moved down one line. Implementors can put this into a change file if they like.

iii. The format `plain.tex` leaves `\box0=\hbox {\tenex B}`; and it also defines `\\` to be a macro such that "`\\10pt`" expands to "`10`" (for example). I could have cleaned these up by saying something like

```
{\setbox1=\box0} \let\\=\undefined
```

but I decided not to change it, since `plain.tex` is so widely used as is.

iv. Frank Mittelbach reported a construction of Morten Høgholm Pedersen:

```
\parindent=0pt
\setbox0=\hbox{p} \hsize=\wd0
\discretionary{m-}{h}{p}\par
```

It gives an overfull box, because TEX doesn't see any feasible breakpoint. (More precisely, the pre-break part exceeds the line width, and TEX doesn't look ahead to see if some fairy godmother is going to save us.) Thus TEX is resigned to making an overfull box, and it takes the only legal breakpoint it knows.

This must be considered a feature of TEX's linebreak algorithm. Namely, a discretionary break is normally never taken when the pre-break part would make an overfull box; but it is always taken in the unusual case that no other feasible break is possible (without looking ahead at the third, "unbroken" alternative of the discretionary). A problem can arise only if an unhyphenated word is actually shorter than its first hyphenated fragment. What, me worry?

Amusingly, if you put the line

```
\spaceskip=0pt plus 1fill
  \discretionary{p}{\kern-2em}{}
```

before the other discretionary, you get two p's and nothing overfull.

v. Jonathan Kew mentioned some of the surprising effects that occur when you try to do things in the command line (or in the very first line of TEX's input, at the `**` prompt). There are many, many such.

Before TEX knows the job name, it outputs just to the terminal. Log file output won't happen until an `\input` command has occurred, or input line one has been processed, whichever comes first, because the log file is given its name at that time.

For example,

```
**\showhyphens{whatever}
```

will show '`what-ever`' on the terminal, but not in the log file. Same for

```
**\showhyphens{whatever} \input foo
```

but in this case the log file is called `foo.log` instead of `texput.log`. With

```
**\input foo \showhyphens{whatever}
```

you see '`what-ever`' also in `foo.log`.

plain TEX format

Version 3.141592653 of `plain.tex` is identical to version 3.14159265, except that `\errorstopmode` is no longer invoked by the `\tracingall` and `\loggingall` macros. (That mistake had been in `plain.tex` for more than 25 years, and I thank David Kastrup for the wakeup call.)

METAFONT

Turning now to METAFONT, Thorsten Dahlheimer gave the whole program a much-needed scrutiny and came up with a number of bugs that have now been corrected in version 2.718281. (Incidentally, he has also given me invaluable help finding mistakes in the darker corners of *TAOCP*.) Only one of those bugs was serious enough to affect real programs with high probability; the others are the sorts of things that a good nitpicker will spot when reading code, although the actual misbehavior requires weird scenarios. As usual, you can find details of the significant changes to Pascal code in the file `errata/mf84.bug`. The complete source file `mf.web` shows many instances of improved commentary.

1. The serious bug arose from user input such as

```
boolean b[]; b1=true=b2;
```

earlier versions of METAFONT would go into an infinite loop from such constructions, so evidently nobody ever writes code like this. (Strings, paths, and pictures have similar problems, not just booleans.) No problem would occur if the statement had been "`b1=b2=true`" instead. I forgot to include one instruction in my program, and it's a glaring error in section 1003.

This bug is also in the METAPOST source, `mp.web`, which I assume somebody else will fix. Whoever does that should also look carefully at the other changes just made to `mf.web`, since so much of the code is common to both.

2. There also were problems in the TFM files when extremely large characters or dimensions were present. For example, from

```
mode:=lowres; mode_setup; designsize:=10pt#;
beginchar("!",160pt#,-160pt#,160pt#);
  endchar; end
```

you get a TFM file with a bad character width and depth, because of an off-by-one error in my code. (TFtoPL doesn't complain about the character height, which violates some but not all of the documentation of TFM files: A `fix_word` is supposed to lie between -2048 and $2048 - 2^{-20}$, inclusive, but *The METAFONTbook* says that no TFM dimension should result in the `fix_word` value -2048. TEX has no problem inputting that value.)

3. Another TFM problem was tweaked with ultralarge design sizes:

```
fontmaking:=1; designsize:=2000;
  fontdimen 2: 3000;
  shipout nullpicture; end
```

used to set `fontdimen 2` (the `SPACE` parameter) to be about 32000 points. The correct behavior is to reduce `fontdimen 2` to just less than 2048 points.

4. Weird behavior could previously occur with

```
transform T;
T=identity xscaled 4 yscaled 3 rotated 180;
pickup pencircle transformed T;
show currentpen;
```

which always came out correctly without the (redundant) rotation by 180.

5. Another bug arose in code fragments like

```
string a.b; a.b="lost"; outer a;
  numeric a.c; showvariable a;
```

the `string a.b` was indeed now lost. (METAPOST probably fails in the same way.)

6. METAFONT now checks that serial numbers don't overflow. Actually I had recommended that the program always be run with arithmetic integer overflow trapped; but this doesn't seem to be current practice. If a user creates 2^{25} distinct numeric variables, the "`METAFONT capacity exceeded`" error now occurs; formerly, this would have caused arithmetic overflow. (Well, this correction was actually made already in TEX-live change files some years ago; I've now introduced it into the master file `mf.web`, in a slightly different way.)

Not a bug: The `init_gf` procedure has an assignment to `str_start[str_ptr+1]` that looks like it could cause a segmentation fault if `str_ptr=max_strings`. Actually, however, that can't happen. (The test "`str_ptr+3>max_strings`" in `end_name`, together with the fact that `area_delimiter=0` in that procedure because `cur_area=""`, provides the extra breathing space.) But I changed `init_gf` anyway.

Anomalies that won't be changed: Autorounding does not work properly when filling certain nonconvex shapes, such as

```
pickup makepen((-.6,0)--(.6,0)--cycle);
filldraw (2,0){up}..(0,1){down}..%
  (1,0){down}..(0,-1){down}..cycle
```

at point `(1,0)`. Pens whose width and height are not integers are deprecated; there's no point cluttering up the code with stuff that benefits only them.

One of METAFONT's (and METAPOST's) most interesting algorithms is the way it chooses control points and directions for paths that are partially specified. I ran into a curious glitch some years ago when preparing an illustration for my book *Selected Papers on Computer Languages*: The two paths

`(0,0){dir45}...(15,0)...(0,0){dir150}`

and `(0,0){dir-45}...(15,0)...(0,0){dir-150}`

turn out to have amazingly different shapes. (The first one twists around almost unbelievably, while the second looks reasonable.) I tracked this down to the equations in METAFONT's "`solve_choices`" routine, which chooses the desired "turning angle" at the point `(15,0)`. In both cases this value, `psi[1]`, is set to `n_arg(-983040,0)`; here -983040 is the internal (scaled) representation of -15, and `n_arg` is supposed to determine the value of `angle(-15,0)`. [See page 67 of *The METAFONTbook*.] The answer is 180, which is appropriate in the second case, but the first case really wants the answer to be -180.

Computer Modern

I made a noticeable change to the shape of one (and only one) letter in the CM family, namely the calligraphic F. The new one has a slightly different swash, which pleases me more when I look at it in *The Art of Computer Programming*. The change is small, yet it would be nice if people would remake the Type 1 versions of the fonts that use `calu.mf`, namely `cmsy*` and `cmbsy*`.

The lowercase Greek nu could develop a tiny notch at the bottom, especially at high resolutions of boldface versions (brought to my attention by Charles Duan, who conjectured its existence by reading the source code!). So I corrected that problem.

Duan also found a few other places where the source code was logically wrong in `greekl.mf`. I fixed those too. However, those changes don't actually show up in the generated font, since the differences in point positions are minuscule.

Karel Piška noticed that the bulbs of lowercase a and c are positioned rather differently when the "`blacker`" parameter of a mode varies. (He blamed it on varying resolution, but that's because my code was obscure.) In those characters I essentially try to move strokes apart so that there's twice as much white space as the thickness of the pen; therefore a `blacker` pen makes the strokes go further apart.

My logic was faulty, because the "`blacker`" setting was intended to compensate for differences in the device that make its apparent pen width too small, thereby making the actual appearance after printing only as black as it would have been on an ideal device; increasing "`blacker`" by 1 shouldn't make me reposition any strokes. Yet I do actually reposition them, on the lowercase a, by roughly 2 pixels per unit of `blacker`! And the bulb on c is positioned to be like that of a. Still, the repositioned bulbs look OK, and I'm happy to continue forever with this wart in the design.

TEXware

TFtoPL version 3.2 is identical to version 3.1 except that a (missing) newline character now appears after one of the warning messages.

Computers & Typesetting

Dozens of corrections were made to Volumes A, B, C, D, and E of the books *Computers & Typesetting*, bringing everything up to date with respect to the latest sources. (This includes *The TEXbook*, which is a paperback Volume A, and *The METAFONTbook*, which is a paperback Volume C.) Copies of the corrected books won't be available for sale until the publisher's stock of already-printed volumes is depleted; but I've prepared detailed errata from which you can make hardcopy inserts to paste into the books you have.

Summary

All of the results of my changes appear in the following files:

- `tex/texbook.tex` % source file for *The TEXbook*
- `tex/tex.web` % complete master file for TEX in Pascal
- `tex/trip.fot` % torture test terminal output
- `tex/tripin.log` % torture test first log file
- `tex/trip.log` % torture test second log file
- `tex/trip.typ` % torture test output of DVItype
- `texware/tftopl.web` % complete master file for TFtoPL in Pascal
- `mf/mfbook.tex` % source file for *The METAFONTbook*
- `mf/mf.web` % complete master file for METAFONT in Pascal
- `mf/trap*` % (namely `trap.fot`, `trapin.log`, `trap.log`, `trap.typ`, `trap.pl`)
- `mf/trap.fot` % torture test terminal output
- `mf/trapin.log` % torture test first log file
- `mf/trap.log` % torture test second log file
- `mf/trap.typ` % torture test output of GFtype
- `mf/trap.pl` % torture test output of TFtoPL
- `cm/calu.mf` % master source file for calligraphic capital letters
- `cm/greekl.mf` % master source file for lowercase greek letters
- `cm/symbol.mf` % master source file for special symbols
- `errata/errata.ten` % changes to Volumes ABCDE before 2001
- `errata/errata.eleven` % changes to Volumes ABCDE in 2001
- `errata/errata.tex` % changes to Volumes ABCDE since the 2001 boxed set
- `errata/tex82.bug` % changes to `tex.web` since the beginning
- `errata/mf84.bug` % changes to `mf.web` since the beginning
- `errata/cm85.bug` % changes to Computer Modern metafont sources since 1985

These files are available in directory `pub/tex/dist` of the ftp server `cs.stanford.edu`, which accepts "`anonymous`" as a login name. They are a subset of the files in `pub/tex/dist/tex08.tar.gz`, which you can compare to `pub/tex/dist/tex03.tar.gz` if you like. Hopefully they will be easy to incorporate into the major distributions of TEX, and they will presumably soon be available on CTAN.

In general the changes can be characterized as a general cleanup, especially to the documentation. The new versions don't affect old documents, except when the existing behavior was seriously incorrect. (And except for the fact that TEX will often run a bit faster now.)

To do this revision I waded through more than 600 K bytes of text files, not counting the binary `.pdf` and `.png` files that were also submitted. Barbara Beeton faithfully compiled all of this material during the years 2003–2007, and organized it so that my task wasn't hopeless. She had many volunteers helping to separate wheat from chaff; needless to say, I'm extremely grateful for all of this assistance.

The total number of independent topics about which I had to make a decision, after they had come through the filtering process, was approximately 335. Some of these needed several days of thought and careful study; some of them needed only a few seconds. More than a hundred of them were nontrivial, and I did my best.

So now I send best wishes to the whole TEX community, as I leave for vacation to the land of *TAOCP* — until 31 December 2013. Au revoir!

Hyphenation Exception Log

Barbara Beeton

This is the periodic update of the list of words that TEX fails to hyphenate properly. The full list last appeared in *TUGboat* 16, no. 1, starting on page 12, with updates in *TUGboat* 22, no. 1/2, pages 31–32, 23, no. 3/4, pages 247–248, and 26, no. 1, pages 5-6.

In the list below, the first column gives results from TEX's `\showhyphens{...}`; entries in the second column are suitable for inclusion in a `\hyphenation{...}` list.

In most instances, inflected forms are not shown for nouns and verbs; note that all forms must be specified in a `\hyphenation{...}` list if they occur in your document. The full list of exceptions, as a TEX-readable file, appears at `http://mirror.ctan.org/info/digests/tugboat/ushyphex.tex`. (It's created by Werner Lemberg's scripts, available in the subdirectory `hyphenex`.)

Like the full list, this update has been subdivided into two parts: English words, and names and non-English words (including transliterations from Cyrillic and other non-Latin scripts) that occur in English texts.

Thanks to all who have submitted entries to the list. Here is a short reminder of the relevant idiosyncracies of TEX's hyphenation. Hyphens will not be inserted before the number of letters specified by `\lefthyphenmin`, nor after the number of letters specified by `\righthyphenmin`. For U.S. English, `\lefthyphenmin=2` and `\righthyphenmin=3`; thus no word shorter than five letters will be hyphenated. (For the details, see *The TEXbook*, page 454.) This particular rule is violated in some of the words listed; however, if a word is hyphenated correctly by TEX except for "missing" hyphens at the beginning or end, it has not been included here.

Some other permissible hyphens have been omitted for reasons of style or clarity. While this is at least partly a matter of personal taste, an author should think of the reader when deciding whether or not to permit just one more break-point in some obscure or confusing word. There really are times when a bit of rewriting is preferable.

One other warning: Some words can be more than one part of speech, depending on context, and have different hyphenations; for example, 'analyses' can be either a verb or a plural noun. If such a word appears in this list, hyphens are shown only for the portions of the word that would be hyphenated the same regardless of usage.

The reference used to check these hyphenations is *Webster's Third New International Dictionary*, Unabridged.

Hyphenation for languages other than English

Patterns now exist for many languages other than English, including languages using accented alphabets. CTAN holds an extensive collection of patterns; see `http://mirror.ctan.org/language/hyphenation` and its subdirectories.

The list — English words

`acupunc-ture(ist)`	acu-punc-ture(-ist)
`aneurysm`	an-eu-rysm
`aneurys-mal`	an-eu-rys-mal
`com-putab(le,ility)`	com-put-ab(le,il-ity)
`copy-rightable`	copy-right-able
`deal-lo-cate(s,d)`	de-allo-cate(s,d)
`deal-lo-ca-tion`	de-allo-ca-tion
`der-i-va-tion`	der-i-va-tion
`deriva-tional`	der-i-va-tion-al
`essence`	es-sence
`fig-urine`	figu-rine
`home-o-static`	ho-meo-stat-ic
`home-osta-sis`	ho-meo-sta-sis
`iso-ge-o-met-ric`	iso-geo-met-ric
`isother-mal`	iso-ther-mal
`makein-dex`	make-in-dex
`mnemonic`	mne-mon-ic
`monoph-thong`	mon-oph-thong
`monospace`	mono-space
`names-pace`	name-space
`om-nipresent(ce)`	om-ni-pres-ent(ce)
`phe-nolph-thalein`	phe-nol-phthalein
`ph-tha-la-mic`	phtha-lam-ic
`ph-tha-late`	phthal-ate
`ph-thi-sis`	phthi-sis
`polyandry(ous)`	poly-an-dry(ous)
`poly-dactyl(lic)`	poly-dac-tyl(-lic)
`polyg-yny(ous)`	po-lyg-y-ny(ous)
`polyp(s)`	pol-yp
`poly-phonic`	poly-phon-ic
`presently`	pres-ent-ly
`re-al-lo-cate(s,d)`	re-allo-cate(s,d)
`re-ar-range(s,d)`	re-arrange(s,d)
`sergeant`	ser-geant
`ser-vome-chan-i-cal`	ser-vo-me-chan-i-cal
`ser-vomech-a-nism`	ser-vo-mech-a-nism
`tex-theight`	`\text-height`
`textlength`	`\text-length`
`tex-twidth`	`\text-width`
`tribesman`	tribes-man

Names and non-English words used in English text

`Malay-alam`	Ma-la-ya-lam
`Mon-treal`	Mont-real
`Pres-by-te-rian`	Pres-by-terian
`Vi-eth`	Vieth

Typography

Typographers' Inn

Peter Flynn

1 METAFONT fonts

Although the majority of the typefaces installed with a standard TEX distribution are available in Postscript Type 1 format, there are still several very useful ones available only in METAFONT format. These include the specialist fonts in the bookhands bundle, such as Rustic or Uncial, the BB Dingbats, the cartographic symbols (Karta), Ogham, and many others.

METAFONT is a font outline language, but by default TEX systems use the bitmap output from such fonts (a `.pk` font file). It is possible that these will never be rewritten as PostScript fonts, so using them will continue to require a TEX system, with a little help from *The LATEX Companion* [2].

The Adobe *Acrobat Reader* notoriously used to make a complete hames of bitmapped fonts on screen, blurring them into unreadability while printing them perfectly, but this has been improved significantly in recent versions. It is very fashionable in certain quarters to decry the use of bitmapped fonts on purely technical grounds, disregarding the fact that they do actually provide a useful — and in some cases essential — instantiation of a specific design.

There are some problems: a few of the fonts available (Ogham is one example) produce microscopically tiny glyphs by default, requiring a correction to the METAFONT code; and not all of them can yet be used directly in LATEX via a standard package. But they are all worth investigating, not just for special effects, but because they are a useful contribution to the range of typefaces available to us.

TYPOGRAPHIA
ARS ARTIUM OMNIUM
CONSERVATRIX

2 Type 1 (PostScript) fonts

Better late than never: I finally managed to rescue my Type 1 virtual font installation script from the ravages of a damaged disk drive. I've been using this for a decade or more, and I documented what it did in 'Formatting Information' [1, §8.3.2], but I never released it into the wild, as it was very specific to my own system.

I've now updated it and documented it, and it's ready to test, although the platform is very restricted: it's a *bash* (1) shell script for TEX Live on Ubuntu Gutsy, installing typefaces from the Bitstream 500–font CD. If you have all this, feel free to download it from `http://latex.silmaril.ie/fonts/cdvf` (too early for CTAN).

So what on earth am I doing? In the face of Alan Jeffrey's (and others') *fontinst*, which uses LATEX itself to install fonts, and Jonathan Kew's wonderful XETEX, which ferrets out and lets you use every font on your hard disk, isn't installing Type 1 fonts a bit, well, *retro*?

Yes and no: while a lot of people are moving to OpenType and other post-PostScript formats, there is still a vast supply of Type 1 fonts around, and still a lot of people who want to install them (to judge by the posts on the topic to `comp.text.tex`). The problem with installing them isn't the making of the `.tfm` files, it's making them play nicely with LATEX, and *that*'s what `cdvf` does.

It takes a typeface from a standard distribution mapfile (currently just `bitstrea.map`) — for example a font family name like `bun`; Bitstream's version of Univers which they call ZurichBT — and creates all the `.tfm` and `.vf` files, moves them (and the `.pfb` and `.afm` files) to the right places, writes a `.sty` (package) file and the relevant `.fd` (font definition) and `.map` files, and finally runs *updmap* and *texhash* to leave you with an immediately usable entire typeface and a report on what was installed. Figure 1 shows the result of typing the command:

```
$ cdvf bitstream univers ZurichBT s
```

It's by no means perfect, but it seems to work, and needs testing. The next stages are:

- generalize it for all Unix-like platforms;
- add support for other CDs/DVDs of typeface collections (currently working on FontSite);
- add support for other encodings (currently it only does T1);
- clean up series and shape detection;
- add isolated-font classification detection, so that arbitrary `.pfb`/`.afm` pairs can be installed without the need for a `.map` file to pre-exist;
- eventually rewrite it in something platform-independent, with a GUI front-end.

Type 1 font files notoriously lack full information about themselves (it's often impossible to machine-detect a sans-serif typeface, for example, hence the final parameter on the command given above). But it still ought to be easier to install Type 1 fonts for LATEX and have them 'just work'.

Table 1: Fonts installed

Family	Series	Shape	
bun	l	n	The quick brown fox jumped over the lazy dog
bun	l	sc	The quick brown fox jumped over the lazy dog
bun	l	it	*The quick brown fox jumped over the lazy dog*
bun	m	n	The quick brown fox jumped over the lazy dog
bun	m	sc	The quick brown fox jumped over the lazy dog
bun	m	it	*The quick brown fox jumped over the lazy dog*
bun	b	n	**The quick brown fox jumped over the lazy dog**
bun	b	sc	**The quick brown fox jumped over the lazy dog**
bun	b	it	***The quick brown fox jumped over the lazy dog***
bun	c	n	**The quick brown fox jumped over the lazy dog**
bun	c	sc	**THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG**
bun	c	it	***The quick brown fox jumped over the lazy dog***
bun	mx	n	The quick brown fox jumped over the lazy dog
bun	mx	sc	THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG
bun	bx	n	**The quick brown fox jumped over the lazy dog**
bun	bx	sc	**THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG**
bun	cx	n	**The quick brown fox jumped over the lazy dog**
bun	cx	sc	**THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG**
bun	ux	n	**The quick brown fox jumped over the lazy dog**
bun	ux	sc	**THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG**
bun	lc	n	The quick brown fox jumped over the lazy dog
bun	lc	sc	THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG
bun	lc	it	*The quick brown fox jumped over the lazy dog*
bun	mc	n	The quick brown fox jumped over the lazy dog
bun	mc	sc	THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG
bun	mc	it	*The quick brown fox jumped over the lazy dog*
bun	bc	n	**The quick brown fox jumped over the lazy dog**
bun	bc	sc	**THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG**
bun	bc	it	***The quick brown fox jumped over the lazy dog***
bun	lq	n	The quick brown fox jumped over the lazy dog
bun	lq	sc	THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG
bun	mq	n	The quick brown fox jumped over the lazy dog
bun	mq	sc	THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG
bun	bq	n	**The quick brown fox jumped over the lazy dog**
bun	bq	sc	**THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG**
bun	x	n	**The quick brown fox jumped over the lazy dog**
bun	x	sc	**THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG**

Table 2: File locations

File	Location
Font definition	/usr/local/share/texmf/tex/latex/psnfss/t1bun.fd
Style (package) file	/usr/local/share/texmf/tex/latex/psnfss/univers.sty
Font map file	/usr/local/share/texmf/dvips/config/bun.map
Font map reference	/home/peter/.texmf-config/updmap.d/10local.cfg
Adobe Font Metrics	/usr/local/share/texmf/fonts/afm/bitstrea/univers
Postscript Font Binaries	/usr/local/share/texmf/fonts/type1/bitstrea/univers
TEX Font Metrics	/usr/local/share/texmf/fonts/tfm/bitstrea/univers
Virtual Fonts	/usr/local/share/texmf/fonts/vf/bitstrea/univers

3

Figure 1: Part of the report output of the `cdvf` font installation script

3 New forum, old forum

John Coffey has started a new forum for technical questions and answers related to typesetting at `http://typesetterforum.com`. This is in bulletin-board format rather than a mailing list like TYPO-L, and has lots of interesting posts from users of all the popular systems (Quark, InDesign, 3B2-as-was, and others, as well as LaTeX).

For those of you who haven't discovered it yet, the typography mailing list TYPO-L is at `http://listserv.heanet.ie/typo-l.html` where you can join or leave, or browse the archives.

Strange that there is no global Usenet newsgroup on typography.

4 2008 TUG meeting in Cork

The Call for Papers went out in February and we already have some good abstracts submitted. If you're doing something interesting with TEX-and-friends (or with type, or in a related field), then you should let everyone know — and the best way to do that is to write it up and submit it (or send it to *TUGboat*).

Registration is now open on the web site at `http://tug.org/tug2008/`, so sign up and book early. We look forward to seeing you all here!

Afterthought

Thanks to Michael Everson for pointing this out in TYPO-L (quoted with permission from David Friedman's Ironic Sans blog, `http://www.ironicsans.com/2008/02/`):

keming. (kĕm′-ĭng). *n.* The result of improper kerning.

References

[1] Peter Flynn. Formatting Information. *TUGboat*, 23(2):115–250, 2002.

[2] Frank Mittelbach, Michel Goossens, Johannes Braams, David Carlisle, and Chris Rowley. *The LaTeX Companion*. Addison-Wesley/Pearson Education, Boston, MA, 2nd edition, 2004.

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

The Greek Font Society

Vassilios Tsagkalos

Abstract

The Greek Font Society is a part of Greece standing against a predominantly easy-going culture.

Background and goals

The text that follows is a translation of a text that originally appeared in Greek in Εὔτυπον (Eutypon), the newsletter of the Greek TEX Friends Group (see `http://www.eutypon.gr`). The text has been translated into English by the original author.

A non-profit organization committed to studying meticulously, creating superbly and offering openly: the Greek Font Society (GFS) was founded in 1992 with the express aim of contributing to the research of Greek typography and supporting Greek typefaces in a digital environment. Its Board of Directors consists of Mikhail V. Sakellariou (President), Lili Macrakis (Vice-President), Dimitris G. Portolos (Secretary), Lena G. Savidis (Treasurer), Giorgos E. Agouridis and Eleftheria Giakoumakis. GFS's type design program began through the collaboration of the then Vice-President Mikhail Macrakis and its artistic director, the painter-engraver Takis Katsoulidis, with the type designer and typography researcher George D. Matthiopoulos and the philologist Eleni Tsialta. The current working group consists of George Matthiopoulos (artistic director) and type designers Mikhail Semoglou and Natasha Raissaki. GFS operates on limited resources which leave no space for promotional campaigns; it will continue to function as long as this is manageable. Operational costs are usually covered by subventions, grants and financial support from various state and non-state bodies.

The Greek Font Society was established to fill the observed gap in systematic research of the importance of Greek typeface design for visual communication as well as to emphasize the specific weight of history, the reverberation and consolidation of visual forms in an area of artistically illiterate publishers with low typographic expectations within Greece. It seeks to offer a solution today to a problem which initially should have been addressed systematically as long ago as the late 19th century, in order to enjoy its fruits today.

GFS's work is well-described by its impressive motto: always study and continuously train your eyes! That's why they investigate the past of Greek typography, study the design rules that formed it, digitize and preserve historical Greek typefaces, and finally make them available to the public. Its typefaces provide support for the "extended" Greek block of Unicode, while older typefaces include the original ligatures and abbreviated forms — end result: as far as possible, each typeface is intended as a complete tool for the user and a historical sample to every new type designer who may wish to learn the history of his/her art before attempting to design.

Although polytonic Greek is technically supported, this does not mean that GFS's statutory goals include reversion to the polytonic system. (As of 1982, the official grammar of the Greek state is monotonic Modern Greek.) GFS is not a literary or linguistic union that can take a collective position on issues outside its area of expertise. Its goal is to reinforce and promote Greek typography in our globalized milieu, that is, it stretches out to reach users not necessarily located in Greece but every user of the Greek language around the world. Since the Greek script has evolved through several forms during the millennia, different scholars should be offered the technical prospect of using it in their own ways: be it Greek or non-Greek researchers of Homer, annotators of Papadiamantis, historians studying Thucydides or critics of El Greco's work, all need to use Greek typefaces for the special needs of their sectors.

Typefaces

The public is therefore offered a continuously widening choice of Greek OpenType fonts freely distributed under the SIL Open Font License, including important historical samples and new designs which respect the typographic tradition and avoid the insensitive and ignorant copying of Latin-based standards. As GFS survives on donations, it offers its cultural work for free, only to remind those who tend to forget that there is an alternative way of living apart from profit-making and also in its attempt to make a positive step towards the others, even to the disadvantage of one's personal gain.

In opposition to the argument made by many that the Greek alphabet is in risk of extinction, because its letters become more and more similar to the glyphs of Latin script fonts, GFS states that the main enemy is not the Latin script itself but rather an aversion to the knowledge of the past and its teachings, the statements of our present leaders which lulls our vigilance, the spiritual sybaritism typical of contemporary Greeks along with the poor quality of most Greek libraries and the taking of minimum pains for every effort. Now, 'effort' is one keyword for the Greek Font Society, along with the second keyword of 'offer': five majuscule typefaces

are offered, together with GFS Complutum Classic (with new capital letters) from the 16th century, three typefaces from the 18th century, another three from the 19th century and seven typefaces from the 20th/21st century — all free of charge! Naturally we can only expect more free GFS fonts.

As for our present (LA)TEX, GFS offers five free fonts: `GFSDidotTeX`, `GFSBodoniTeX`, `GFSPorsonTeX`, `GFSNeoHellenicTeX` and `GFSArtemisiaTeX`. That is, we do provide specific fonts and a web page dedicated to TEX, combined with a presentation of the Department of Mathematics at the University of the Aegean and the Laboratory of Digital Typography and Mathematical Applications. All this is thanks to a collaboration of the GFS with a member of our Greek TEX Friends group, Antonis Tsolomitis. Needless to say, all OpenType fonts can be used directly with XETEX/XELATEX.

Publishing

The Greek Font Society is also productive in publishing, since it has translated and designed Robert Bringhurst's renowned book "The Elements of Typographic Style,", Crete University Press, Iraklion, 2001, with a grant by the Stavros Niarchos Foundation. For the occasion of the 2004 Olympic Games in Athens, GFS designed and published, also with the kind support of the Stavros Niarchos Foundation, a set of seven bilingual not-for-sale publications (English, French, German, Modern Greek, Italian, Russian, and Spanish) of the 14 Olympic Odes of Pindar using historical Greek typefaces from the 15th to the 20th century for the ancient Greek text. The Ancient Greek text was typeset in seven digital redesigns (one for each language) of the most celebrated Greek typefaces elaborated in several European countries. This publication was aimed at asserting that the Olympic Games, far from becoming used up as an instance of immoderate commercialization, ought to be generating a cultural dialogue uniting all nations. The set was offered as a gift to all of the participating Olympic Committees in the Games and sent to a range of libraries in Greece and abroad.

In 1995 the GFS organised an International Conference, "Greek Letters: From Tablets to Pixels" at the Institute Français d'Athènes in 1995, with Greek and foreign participants who presented their research on the history, aesthetics and technological future of the Greek typefaces. The proceedings of the Conference are downloadable in Greek from the GFS's website, and their English translation by Mikhail S. Macrakis (ed.), "Greek Letters: From Tablets to Pixels", was published by Oak Knoll Press, New Castle, Delaware, 1996.

GFS is also preparing an anthology of Greek typography, presenting books which have impacted the typography of Greek publications from the era of earliest typography to mid-20th century, as well as historical information on the publishers and typographers who created them. The anthology will aim at making Greek typographic history accessible to all and enabling Greek graphic designers to come into contact with the history of their art by offering images and data kept hidden on library shelves.

Interview: George Matthiopoulos

Mr George Matthiopoulos works as a type designer for the Greek Font Society. He kindly agreed to answer the questions which Εὔτυπον put to him.

Q: What makes Greek typographic history worth attention and study?
A: The history of every aspect of the art and techniques which shaped the course of modern Greek culture is, in my view, self-evidently worthy; add to this that a better understanding of the typographic tradition aids the creative assimilation of the globalized visual communication by new generation graphic designers.

Q: Would you say that the Greek script receives its value through the ages on the simple grounds of being the carrier of the Greek language and civilization or does it have its own value due to some important people that have served it?
A: Letters are primarily communication symbols of a language, but in the course of time they develop a timeless aesthetic connection with language. The Greek alphabet originated without many modifications from the Phoenician alphabet and the Latin alphabet was derived from the Greek. [According to the American scholar Barry B. Powel, the alphabet is a Greek invention: see Homer and the Origin of the Greek Alphabet, Cambridge University Press, 1991.] So, any couplings between language and aesthetics are not metaphysical and immovable ideas (in the Platonic sense), but are rather shaped by the historical process: historical changes, such as the transition from majuscule to miniscule script (around the 9th century) or the simplification of the typographic case by removal of the Byzantine ligatures (in the mid-18th century) were not undertaken by individual persons but resulted from already existing changes in everyday practice.

Q: In an age of abundant impressive technological means, with many technical issues resolved, do you think that the artistic component of typography is served or oppressed by technological advancements?

A: Like the tools of every form of art, computers and their software may prove to be means of emancipating those who realize the capabilities and the limitations of these means, but for their lazy-minded users they only foster their shallow minds. Now, shallowness flourishes around us.

Q: Can you highlight the reasons why typographic culture is absent in the Greek society today? Could it be a matter of culture in general and of the utilitarian notion of knowledge?
A: Yes, I think the answer is there. How can there be typographic education when education in general is becoming an even more distant dream for the majority? I mean here real Culture, not the industrialized product of our grade-aholic educational system we have all come to accept. Aesthetical culture is even more demanding than the general education and this is a field in which modern Greece is still held back. Add to this the loss of the popular elegant taste which sprang spontaneously from the people until just a few years ago. The "mincing machine" called television has ground everything from one end to the other into a colorless and tasteless mass. You see, the increased money we have made in the recent decades is not enough in itself.

Q: Would you like to share your experience from contemporary typographic culture in other European countries? Are there any immediate initiatives that can be taken?
A: The states that have always had a tradition of typography continue to support typographical practice and bring new ideas. Even non-central European countries like Spain and Portugal have already organized the education of type design. Indeed, they all have the privilege of drawing upon the centuries-long research and background of Latin script typography, from which we can benefit only indirectly. The issue of typographic education is big and difficult to deal with in a general way. There has been more progress lately compared to what was happening a decade ago, but there is a long way to go. The Greek Font Society tries to lay the foundations, in order for all who become sensitive to the issue to be able to refer to sources. The Union of Greek Graphic Designers has also recently demonstrated an actual interest and we will continue to work with them. Both the School of Graphic Arts at the Athens Technological Educational Institute and the Athens School of Fine Arts currently promote the teaching of typographic art more dynamically, as well as private educational centers and other schools. Still, I am not sure whether this is a momentary interest by certain individuals today or something that might be long-lived. Greece is unfortunately not the place to make easy predictions.

Q: Given that young people use text editing in their computers from a very young age, do you think that there are proposals to introduce typographic education in at least the secondary education level by providing stimuli and asking for speculation on the aesthetics of our texts?
A: That would definitely be very good indeed, but I am afraid that Greek secondary education has other more important issues to address first.

Q: Are you bothered by the completely unaccented Greek appearing often on the TV, mainly on the news broadcast? [News captions, headlines and transcriptions on Greek TV news is almost always typed without even a tonos (the monotonic accent symbol) on vowels.]
A: No, not personally. I am rather bothered aesthetically by the monotonic Modern Greek. The GFS does not claim the role of a philological leader. In a sense we are "accent neutral", if I may use this neologism. As a Society, we offer the maximum range in our products, that is, polytonic Greek fonts, so that anyone can use them in any way they see fit.

Q: How do you explain the contradiction of a downgraded visual appearance of our texts in our age where image aesthetics and visual representations keep the leading role?
A: I think I referred to this before. Those who deal in visual communication in Greece today rarely possess any aesthetical culture which presupposes knowledge of the historical tradition. Most standards are imported to Greece and remain unassimilated.

Q: Let me ask you a reasonable question: Have any of those who are par excellence interested in the diachronic Greek language and advocate it so far embraced, promoted or supported your work? I refer to philologists, historians, archaeologists, theologians, Universities, the Church, etc. [The Eastern Orthodox Church has never adopted monotonic Modern Greek and still uses the polytonic grammar in its services and most of its communication.]
A: Not to the extent that one would actually expect, but then again what else has been institutionally embraced by them? Nothing, so this does not leave us out either. Nevertheless, individual entities from all of the above have individually expressed their solidarity with our work.

Q: Your website includes a special page devoted to TEX and offers TEX-compatible fonts. How do you appreciate this interest of people from exact science?
A: I am a type designer myself, but through my father's love for Mathematics I have developed an

intimate and warm relation with Science — although I do not particularly succeed in it. Furthermore, the late Mikhail Macrakis, the actual founder of GFS, was a physicist. This bidirectional relationship between the GFS and people from exact science came as no surprise to me, but, if we need to be objective about this, that is still a welcome mystery. Obviously, typography has indeed touched an unknown, until recently, sensitive string of the more "practical minds", as the cliché goes. I suppose we all ought to be grateful to Dr Donald Knuth who identified this sensitive string and to Dr Antonis Tsolomitis, Department of Mathematics, University of the Aegean, Samos, Greece, who makes such a remarkable effort to keep this bridge open in the geographical region to the east of Greece. Our reference to TEX would have been impossible without his initiative and contribution and I would like to take this opportunity to thank him warmly. [Naturally, with the advent of XETEX it is now possible to use any OpenType or TrueType font directly, something that has greatly simplified the job of many people.]

Q: You support the Open Source "movement — free software for all. Can you explain why to us?

A: As long as the Greek Font Society manages to survive on grants, we have to return the trust demonstrated to us by offering our work to those who ask for it. The need to spread the knowledge about Greek typography is too big to be hindered by our making economic demands on those who seek this knowledge. "You can't receive anything from someone who hasn't got anything" and "share with others what others have offered you": that is our idea on the issue.

Q: The verbal poverty of Modern Greeks inevitably affects the separate professional sectors too, who normally turn to the adoption of predominantly English terms in spite of the richness of Greek terminological variants. Since "the start of wisdom is in the study of the names" according to Antisthenes, does the Greek Font Society intend to compile an integral dictionary of typographic terminology?

A: You are right. Terminology in our field is desperately poor, but the technology of graphic arts changes at such a fast rate that it is no longer possible to catch up with the tide. Mikhail Macrakis had also wished a glossary of typographic terms, so we started an early effort during the translation and design of Robert Bringhurst's book "The Elements of Typographic Style" (Crete University Press, Iraklion, 2001), which included a relevant Appendix. I have collected much material since then, but haven't managed to get the time to organize it. In any case, I believe that apart from the descriptive and explanatory definition of lemmas, the very onomatopoeia of terms is beyond my knowledge.

Q: Finally, on the occasion of our communication, please allow me to congratulate you on behalf of all of the members of the Greek Font Society for the work you offer us and above all for the high quality level and the sensitivity it shows. I thank you for your time!

A: I thank you in my turn for the opportunity you gave us to present the Greek Font Society.

◇ Vassilios Tsagkalos
Dip TransIoL, Voula, Attica
Greece
wassily (at) ath forthnet dot gr
http://www.greekfontsociety.gr

Designing and producing a reference book with LaTeX: *The Engineer's Quick Reference Handbook*

Claudio Beccari and Andrea Guadagni

Abstract

This article describes the process of designing an Italian reference book, namely *Il prontuario dell'ingegnere*. As a reference work it shares the characteristics of any other such book; it also uses a large amount of mathematics and contains a large number of graphics insertions. Moreover, the publisher wanted to distribute the work both as a regular bound book and as a computer file to be read directly on the computer screen.

1 Introduction

In the first half of the nineties, Andrea Guadagni (AG) asked Claudio Beccari (CB) to collaborate on the production of a 'Quick Reference' handbook, in a thinner and more comfortable format than the historical and bulky landmark volume for engineers, the *Manuale dell'ingegnere* (The Engineer's Handbook) by Colombo[1] [2]. This Quick Reference handbook was planned to be published at the publishing company Hoepli, well-known in Italy for its collection of handbooks in every possible discipline. The general idea was that the Quick Reference should be a collection of records, each one on a particular subject, with a minimum of mathematics, preferably self-contained and without reference to other records.

As the editor of the huge "Manuale", AG had all the necessary experience for finding the experts, for organising the material to be published, and for managing the whole publishing process. Eventually he also took care of the source file editing.

On the other side, CB already had some experience with LaTeX design and layout, but the production of the "Quick Reference" was a new challenge that required a very close interaction between AG and CB.

Editor's note: This is a translation of the article "La progettazione di un'opera di consultazione: l'edizione del *Prontuario dell'ingegnere* con LaTeX", which first appeared in *ArsTeXnica* issue #4 (October 2007), pp. 16–24. Reprinted with permission. Translation by the author.

[1] Professor Colombo's first name was Giuseppe, but the handbook was and still is simply named "Il manuale del Colombo" (Colombo's handbook) or "il Colombo" by every engineer, no matter the specialization.

2 LaTeX in the nineties

First of all it may be useful to mention some questions concerning computers and programming that today are obsolete, but at the time were a real problem; mentioning these problems is of general interest in every cooperative work when many authors are involved. Moreover, even if advances in the computer world have eradicated several incompatibilites, often new ones arise, producing new problems. It is advisable that authors who want to produce a cooperative document be aware of these problems and know how to overcome the difficulties that they might encounter along their path.

One of the problems was the fact that CB, AG and the various collaborators used different platforms. CB was still working with a DOS PC; Windows 95 had just appeared but it was not so widespread on existing PCs. Windows 3.1 was still too rudimentary for this kind of work, for one thing because its graphic interface used most of the memory available to those PCs. Moreover, CB was still working with LaTeX 2.09, although by the end of the work LaTeX 2_ε was available; as usual at that time he was using a plain ASCII editor and ran the various LaTeX connected applications with suitable command lines.

On the other hand, AG was working with a Macintosh, whose operating system was very different from the simple DOS from many points of view, not least that paths were described in a very different way. Obviously it had a beautiful graphic interface, but this was more or less irrelevant for the goal of the production of this work. Concerning the TeX system, AG was using Textures; this was (and still is) a commercial product by BlueSky Research and it worked both as an editor and as a synchronous previewer, that is, it employs two windows open simultaneously, in one of which the usual `.tex` source is edited while the other synchronously displays the result of typesetting at the same rate that text is input in the editor window.

Today, everything is simpler; plain DOS is almost forgotten; although the command window allows one to use a more modern DOS incarnation, most users appear to ignore its existence. Apple has recently changed the operating system of its Macintoshes, passing to Mac OS X, a Unix-based operating system with the typical Macintosh beautiful graphic interface.

On the TeX system side, the TeX Users Group and most other TeX user groups distribute the free TeX system through the TeX Collection CDs and DVDs; TeX Live is available for all platforms and

the differences between the various operating systems are managed in a unified way, so that the macro collections and the packages are more and more platform independent.

But in the nineties it was necessary to fight against the various ways of indicating absolute and relative paths: DOS and Unix used the slash / while Macintosh used the colon :, and in different positions from those where DOS and Unix would use /. This was relevant for the production of the Quick Reference handbook, because the subpaths where it was handy to keep the pictures and other included material had to be accessed on both systems. Just to give an idea of the code that had to be implemented, here are the macros that would automatically change the explicit path of a file containing a figure.

```
\newif\if@path \@pathfalse
% These macros were adapted from Knuth's
% TeXbook (pag. 375)
\def\check@figurefilepath#1{%
   \@@chkfpM#1:\@@chkfpM\@%
\if@path\else\@@chkfpU#1/\@@chkfpU\@\fi}
%
\def\@@chkfpM#1:#2#3\@{%
  \ifx\@@chkfpM#2\@pathfalse\else
      \@pathtrue\fi}
\def\@@chkfpU#1/#2#3\@{%
  \ifx\@@chkfpU#2\@pathfalse\else
      \@pathtrue\fi}
%
% This is the macro that possibly prepends
% the path specification to the file name
% of a figure, then it checks if the
% folder or directory separators agree
% with the operating system; eventually it
% passes this generated full specification
% to the macro that includes the figure.
%
\parse@figure@filename#1{%
\def\figurefilename{#1}%
\ifx\def@ultpath\empty\else
  \expandafter\check@figurefilepath
  \expandafter{\figurefilename}%
  \if@path\else
     \edef\figurefilename{%
     \def@ultpath\ifMacintosh:%
        \else/\fi\figurefilename}%
  \fi
\fi
\ifMacintosh
  \expandafter\Mac@rename
     \figurefilename/\relax
\else
  \expandafter\UNIX@rename
     \figurefilename:\relax
\fi}
%
% Recursive macro that erases the slashes
% and sets the colons.
%
\def\Mac@rename#1/#2\relax{%
    \def\@tempA{#2}%
    \ifx\@tempA\empty
        \def\figurefilename{#1}%
    \else
        \def\@tempA{#1:#2}%
        \expandafter\remove@slash
            \@tempA\relax
        \expandafter\Mac@rename
            \figurefilename/\relax
    \fi}
%
% Recursive macro that erases the colons
% and sets the slashes.
%
\def\UNIX@rename#1:#2\relax{%
  \def\@tempA{#2}%
  \ifx\@tempA\empty
     \def\figurefilename{#1}\else
     \def\@tempA{#1/#2}%
     \expandafter\remove@colon
         \@tempA\relax
     \expandafter\UNIX@rename
        \figurefilename:\relax
  \fi}
%
% These macros erase the final slash
% or colon.
%
\def\remove@slash#1/\relax{%
   \def\figurefilename{#1}}
\def\remove@colon#1:\relax{%
    \def\figurefilename{#1}}
```

3 The layout of the book

For the typeset book AG and CB agreed on a layout where each record would be a freestanding section to be introduced with `\section`. Furthermore, each record should consist of a single spread: in the 'paper' version of the book, where pages are turned from right to left, it was necessary that the written part be typeset on an odd page (the recto page) on the right side of the spread, while the graphic contents of the record would appear on the verso of the previous sheet, an even page, i.e. on the left part of the spread. The sections were collected in chapters, and the latter in parts. Each part dealt with a particular branch of engineering, for example the part "Edilizia" (Building) would be divided in the chapters dealing with such chapters as "Foundations", "Pillars", "Roofs", and the like; each chapter contained a few records concerning the details, with formulas, tables, drawings, descriptions, costs, etc.

The publishing house wanted also an "e-book" version of this reference book; in this electronic version, where pages scroll from bottom to top, but reading goes from left to right, the record should occupy a single screen, with the written part on the left and the graphic part on the right.

Both versions have thumbnails on the side of the page. In the paper version traditionally it would have been necessary to make a semicircular cut in the pages, as sometimes seen in large Holy Books, or in large dictionaries or encyclopedias. However, the high cost of this operation suggested we use a simpler method, namely to let the thumbnail shadow be visible on the external margin cut, by typesetting such thumbnails across the trimming area.

The electronic version had all the necessary internal links, but for consistency with the printed version it also contained various icons (thumbnails) on the left side.

Headers, footers, part titles, records, the table of contents and the index required a particular layout, partly because each page, each contents entry, and each chapter required the name of the individual author or the name of the chapter coordinator.

The source files for both the printed and the electronic version had to be identical and the output format chosen with a single option in the main file.

4 Page layout and fonts

The trimmed page size was a standard ISO-A5, that is, 148 mm by 210 mm. The main text box, without headers and footers, was 116 mm by 175 mm. With such a compact text box, we decided to typeset the book with fonts of size 9/10 pt; we left the decision on the font collection to a later time, after attentive examination of the typesetting results with different typefaces. We eventually chose the Times fonts, because with Textures (at that time) they were easier to configure and use.

Figure 1 shows a spread of the printed version: the left page contains the drawings relative to the record or section, while the right page displays the text and the other relevant information. The thumbnail icon that characterizes the part [in the example: "Edilizia" (Building)] is on the right page. The part title is repeated on the left header, while the chapter title is only in the right header. The section title is centered above the text. The record is subdivided into small unnumbered subsections in line with the text. The section author is in the right footer while the left footer contains the book title and the publisher's name.

Figure 2 contains in one screen shot more or less the same information, except that thumbnails, text and drawings are interchanged with regard to 'left' and 'right' sides. There is just one header and one footer, the contents of which are flush to the external margins. The rest is identical to the printed version.

While the printed version had pages of size A5 that form a spread in a landscape A4, the electronic version has one screen shot in landscape with a page size (needed for the PDF output) of a landscape A4 sheet. The typesetting information for LaTeX had to be completely automatic, depending on the version to be output. If a screen shot had to be printed, either the landscape orientation must be compatible with the printer, or the latter should be capable of scaling the printed area to the paper width. This was a necessary warning at that time; today all printers are capable of printing in landscape mode.

5 LaTeX commands for inputting records

The LaTeX input of this Quick Reference handbook obviously requires specific commands in order to cope with the specific requirements of the layout.

For example, the drawings of each record must be separately assembled, possibly with recourse to a graphics editor so as to produce a single graphic object to be included in the output file. At those times it was absolutely necessary to run LaTeX and produce output in DVI format, then passed through dvips in order to get the PostScript format, and finally to get the PDF format by running the ps2pdf program. Today it would be possible to get the same result with one pass through pdflatex. Nevertheless, once suitable macros were made available, it was easy to get the desired result for both the editor and the individual authors.

The macros for starting a new spread or a new chapter required only the specification of the pertinent title and the name of the author or editor.

The macros that started a new part had to provide the pointer to the specific thumbnail and to its typesetting on the proper margin in each version. The option `draf` had to be modified so that during the preliminary work the authors and the editor could save a lot of black ink by omitting printing the thumbnail, but leaving the explicit part title in the relevant margin.

The babel package at that time did not have the functionality it has today; in addition many Italian LaTeX users did not even know they could typeset with the rules of Italian typography and with Italian hyphenation, or they did not know how to configure their system; many were still working with LaTeX 2.09. So we created the necessary tests with suitable warning messages in order to assure at least

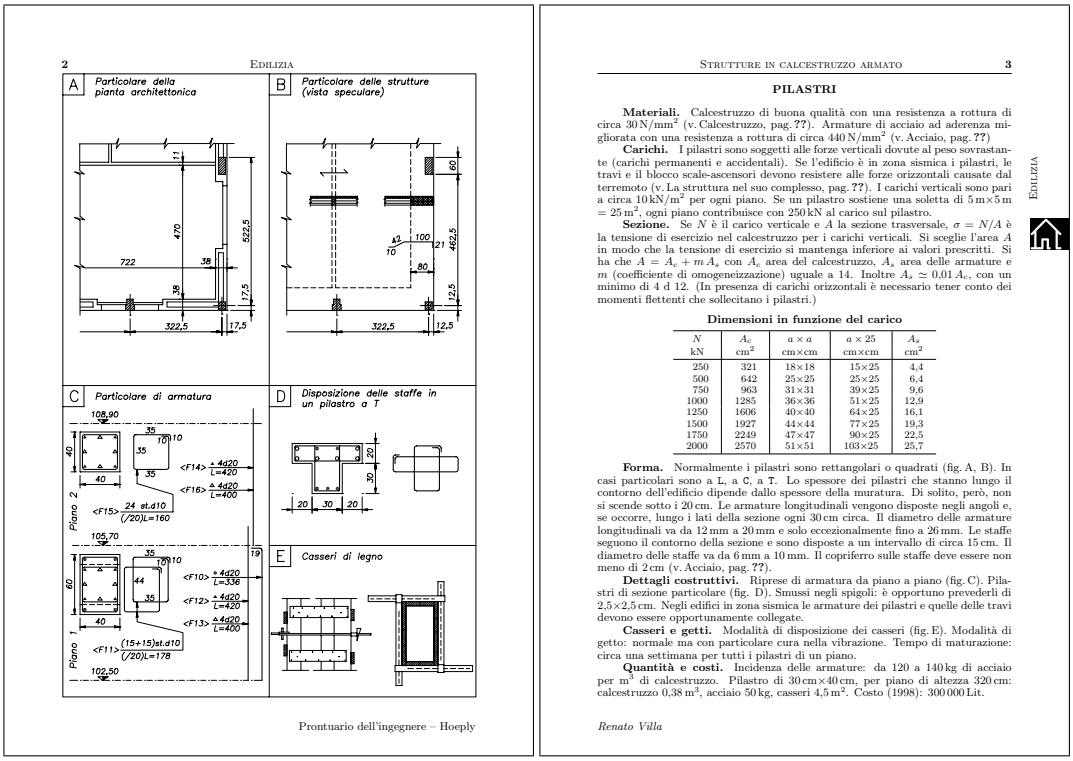


Figure 1: Printed version: a spread.

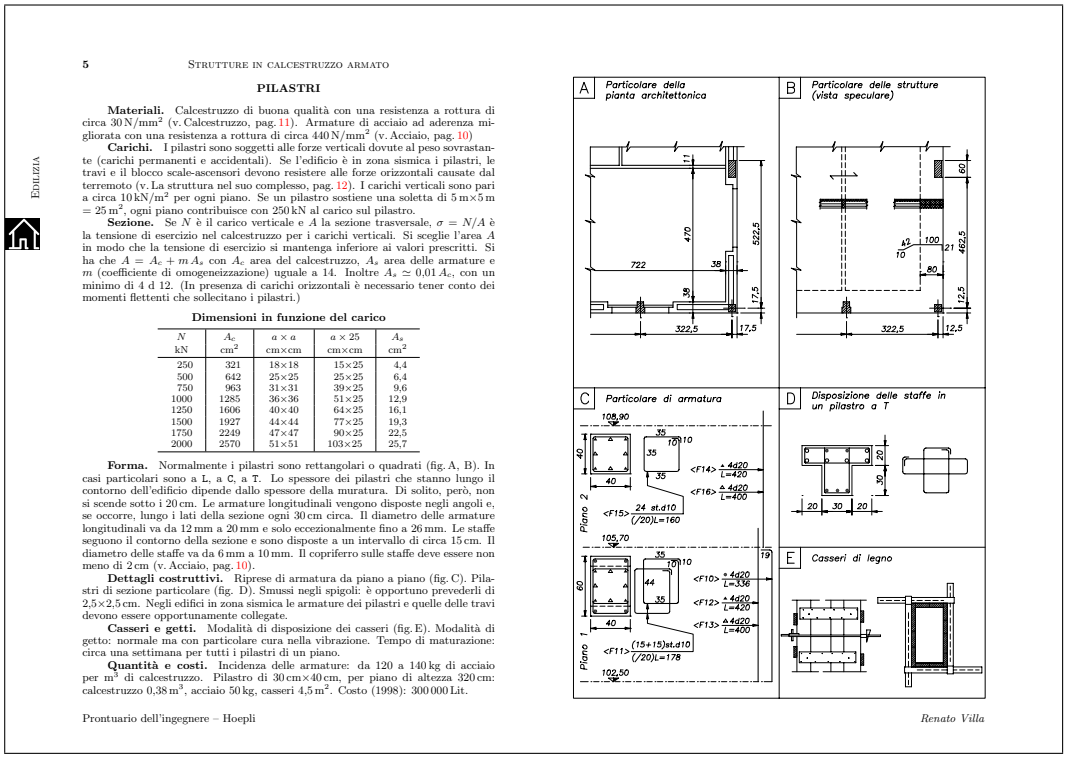


Figure 2: Electronic version: a screen shot corresponding to the same spread shown in figure 1.

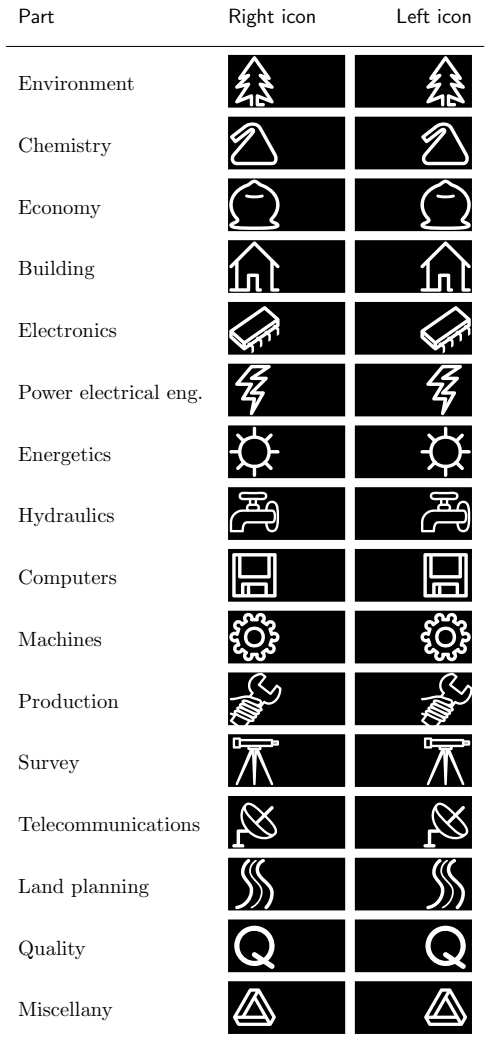


Figure 3: Right and left hand thumbnails.

the use of Italian hyphenation.[2] The code was the following:

```
\expandafter
  \ifx\csname language\endcsname\relax
\else
  \expandafter
    \ifx\csname l@italian\endcsname\relax
 \typeout{La sillabazione per l'italiano
          non e' definita.^^J
          Usero' la sillabazione di
          default e faro' tanti errori!}%
  \typeout{Verificare che la sillabazione
           per l'italiano sia stata
           caricata^^J
           e che il suo nome sia associato
           al comando \string\l@italian}%
```

[2] Even today, although everything is much simpler, many Italian users run LaTeX to typeset Italian texts without configuring it for Italian!

Figure 4: The thumbnail shadow on the trimmed side of the second edition.

```
   \language0\righthyphenmin 2\relax
           \lccode`\'=`\'
  \else
    \def\italiano{\language \l@italian
           \righthyphenmin 2\relax
           \lccode`\'=`\'}%
    \italiano
  \fi
\fi
```

Well, today even these messages are built-in to the babel package, and warning messages are much simpler to program. (The first message says: "Italian hyphenation is undefined. I'll use the default hyphenation and I'll make a lot of errors!" The second message says: "Verify that Italian hyphenation has been loaded and that its name is associated to the command `\l@italian`".)

6 Thumbnails

Concerning the thumbnail icons, we solved the problem by designing their fonts, with white strokes over a black background, and with the background extending well beyond the actual icon, so as to protrude into the right trimming area for the printed version, or to protrude well to the left of the virtual PDF sheet border for the electronic version. This solution guarantees the visibility of the inked icon shadow on the trimmed pages of the printed version, even without the typical thumbnail semicircular cut. Figure 3 shows the left and right icons, with the extending background; initially they were produced with METAFONT; but for the second edition they were traced and rendered as scalable outline Type 1 fonts.

Figure 3 displays the icons on a wide black rectangle, whose width goes beyond the trimmed page; this is why when pages are trimmed, the closed book exhibits on the right trimmed side the thumbnail shadows, figure 4; probably this fact is not so important while 'navigating' the handbook as the icon

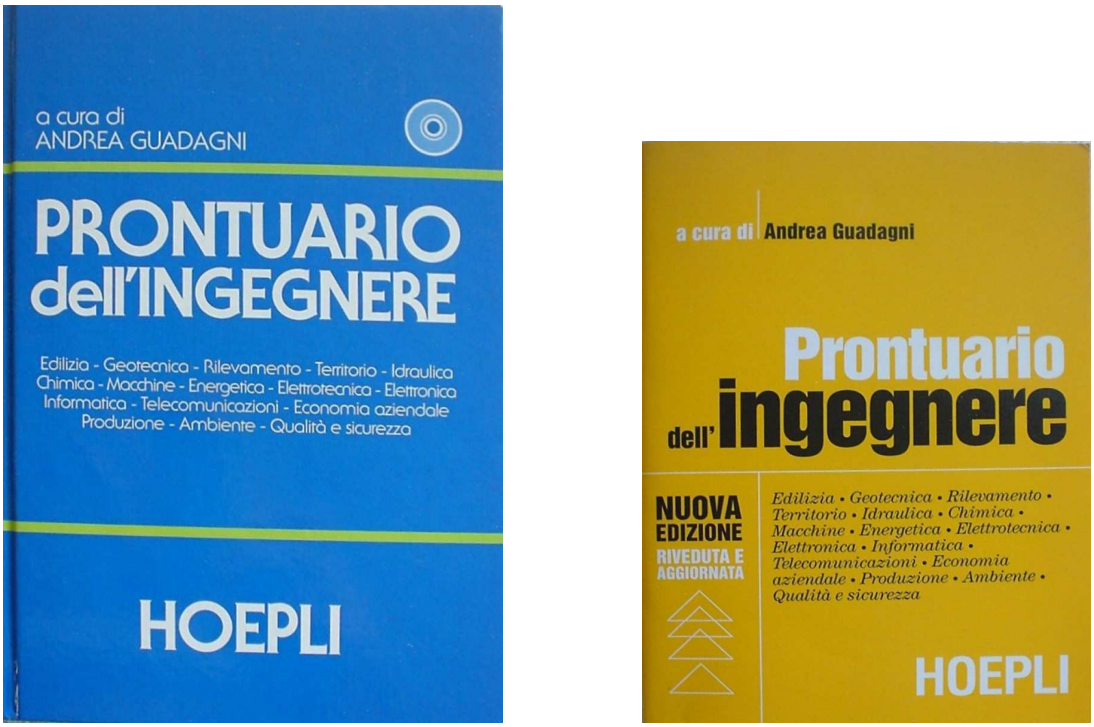


Figure 5: The published editions; on the left the first edition in size A5; on the right the second edition in size 120 mm × 170 mm.

drawn on the flat side of the thumbnail — even when rapidly flipping the handbook pages it's very easy to identify the icon one is looking for. It goes without saying that the thumbnails of every part are progressively shifted down along the margin so that they appear still when pages are rapidly flipped.

Even the electronic version, although it has its icons on the left border and does not have pages to be flipped through one's fingers, benefits from the presence of such icons when the screen view is rapidly scrolled up or down by operating on the scrolling arrows of the PDF reading program.

7 Internal hyperlinks

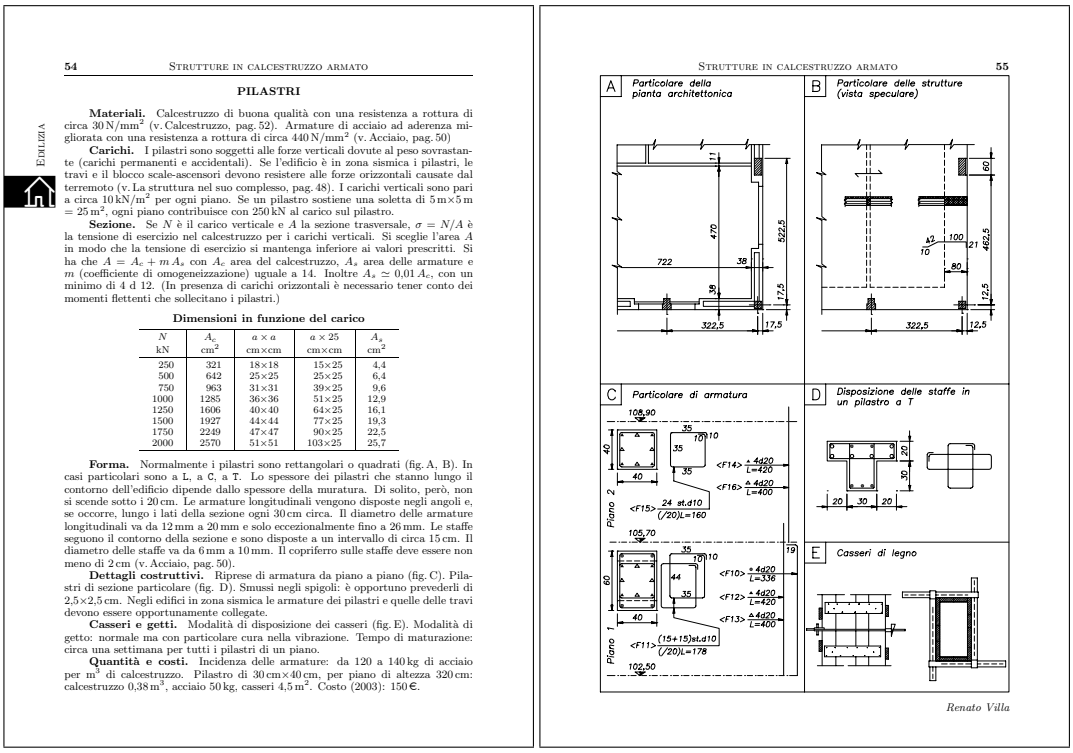
The various elements of the electronic version are completely hyperlinked with one another by means of the hyperref package, so that surfing the document gets particularly simple when the colored anchor texts are clicked upon. Even the table of contents entries and the index entries are hyperlinked with their sources. The sections that must refer to other sections are also hyperlinked. In this way the electronic version is even more easily readable than the printed one. In the nineties this was a real novelty that has been appreciated by the readers.

8 The second edition modifications

The second edition [3] underwent a few layout modifications compared to the first edition. The page size was reduced to the trimmed page size of 120 mm by 170 mm, figure 5. The written and drawn parts of each record were interchanged as may be seen in figure 6. The thumbnails were set on the left side, as in the electronic version, but in both versions they were set a little closer to the text border. Together with figure 6 it's possible to examine a screen shot in figure 7. The thumbnail shadows remain visible on the trimmed page side, figure 4.

9 The publishing house policy

The Hoepli Publishing Company is very well known for its (Italian) handbooks and, among engineers, for its *Manuale del Colombo* (Colombo's Handbook). This work started in 1875 and was published in many successive revisions so as to become almost encyclopedic, since it can no longer be used as originally intended, as an easily workable handbook. We all know that engineering sciences have evolved in an incredible way in the past century, so that the evolution of Colombo's Handbook has been unavoidable. For these reasons, in our *Prontuario dell'Ingegnere* (The Engineer's Quick Reference Handbook),



54 STRUTTURE IN CALCESTRUZZO ARMATO

EDILIZIA

PILASTRI

Materiali. Calcestruzzo di buona qualità con una resistenza a rottura di circa 30 N/mm^2 (v. Calcestruzzo, pag. 52). Armature di acciaio ad aderenza migliorata con una resistenza a rottura di circa 440 N/mm^2 (v. Acciaio, pag. 50)

Carichi. I pilastri sono soggetti alle forze verticali dovute al peso sovrastante (carichi permanenti e accidentali). Se l'edificio è in zona sismica i pilastri, le travi e il blocco scale-ascensori devono resistere alle forze orizzontali causate dal terremoto (v. La struttura nel suo complesso, pag. 48). I carichi verticali sono pari a circa 10 kN/m^2 per ogni piano. Se un pilastro sostiene una soletta di 5 m×5 m = 25 m^2, ogni piano contribuisce con 250 kN al carico sul pilastro.

Sezione. Se N è il carico verticale e A la sezione trasversale, $\sigma = N/A$ è la tensione di esercizio nel calcestruzzo per i carichi verticali. Si sceglie l'area A in modo che la tensione di esercizio si mantenga inferiore ai valori prescritti. Si ha che $A = A_c + m\,A_s$ con A_c area del calcestruzzo, A_s area delle armature e m (coefficiente di omogeneizzazione) uguale a 14. Inoltre $A_s \simeq 0{,}01\,A_c$, con un minimo di 4 d 12. (In presenza di carichi orizzontali è necessario tener conto dei momenti flettenti che sollecitano i pilastri.)

Dimensioni in funzione del carico

N kN	A_c cm^2	$a \times a$ cm×cm	$a \times 25$ cm×cm	A_s cm^2
250	321	18×18	15×25	4,4
500	642	25×25	25×25	6,4
750	963	31×31	39×25	9,6
1000	1285	36×36	51×25	12,9
1250	1606	40×40	64×25	16,1
1500	1927	44×44	77×25	19,3
1750	2249	47×47	90×25	22,5
2000	2570	51×51	103×25	25,7

Forma. Normalmente i pilastri sono rettangolari o quadrati (fig. A, B). In casi particolari sono a L, a C, a T. Lo spessore dei pilastri che stanno lungo il contorno dell'edificio dipende dallo spessore della muratura. Di solito, però, non si scende sotto i 20 cm. Le armature longitudinali vengono disposte negli angoli e, se occorre, lungo i lati della sezione ogni 30 cm circa. Il diametro delle armature longitudinali va da 12 mm a 20 mm e solo eccezionalmente fino a 26 mm. Le staffe seguono il contorno della sezione e sono disposte a un intervallo di circa 15 cm. Il diametro delle staffe va da 6 mm a 10 mm. Il copriferro sulle staffe deve essere non meno di 2 cm (v. Acciaio, pag. 50).

Dettagli costruttivi. Riprese di armatura da piano a piano (fig. C). Pilastri di sezione particolare (fig. D). Smussi negli spigoli: è opportuno prevederli di 2,5×2,5 cm. Negli edifici in zona sismica le armature dei pilastri e quelle delle travi devono essere opportunamente collegate.

Casseri e getti. Modalità di disposizione dei casseri (fig. E). Modalità di getto: normale ma con particolare cura nella vibrazione. Tempo di maturazione: circa una settimana per tutti i pilastri di un piano.

Quantità e costi. Incidenza delle armature: da 120 a 140 kg di acciaio per m^3 di calcestruzzo. Pilastro di 30 cm×40 cm, per piano di altezza 320 cm: calcestruzzo 0,38 m^3, acciaio 50 kg, casseri 4,5 m^2. Costo (2003): 150 €.

STRUTTURE IN CALCESTRUZZO ARMATO 55

A Particolare della pianta architettonica

B Particolare delle strutture (vista speculare)

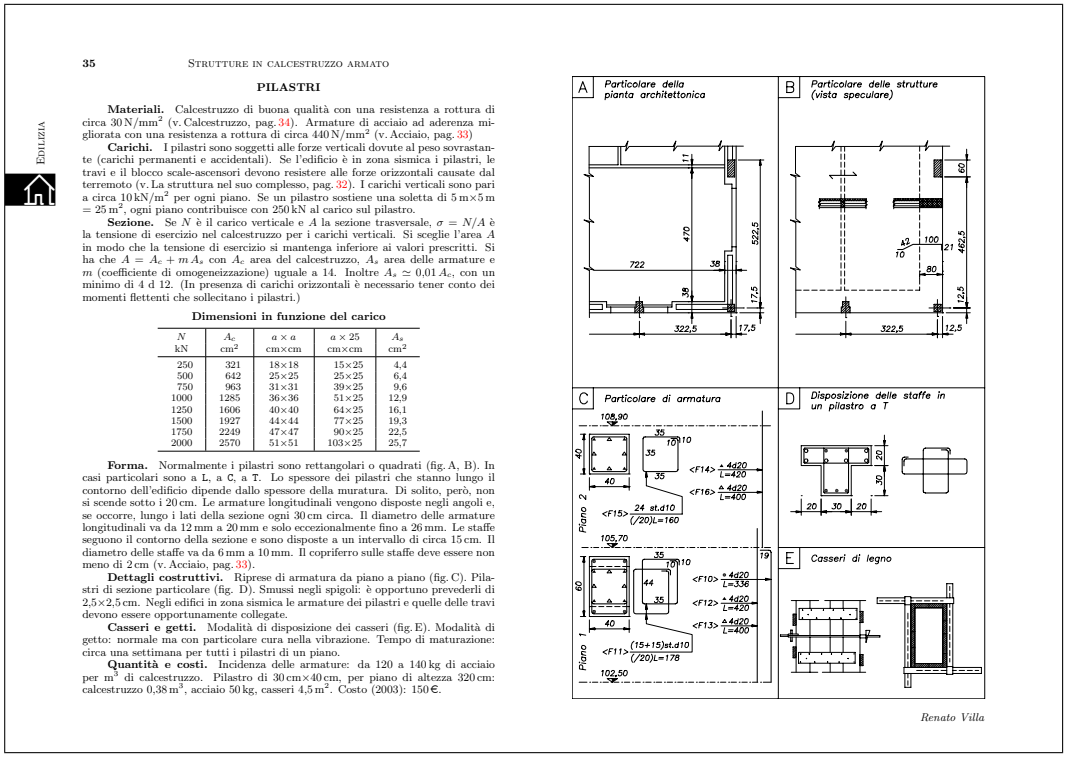
C Particolare di armatura

D Disposizione delle staffe in un pilastro a T

E Casseri di legno

Renato Villa

Figure 6: Second edition printed version: a spread.



35 STRUTTURE IN CALCESTRUZZO ARMATO

EDILIZIA

PILASTRI

Materiali. Calcestruzzo di buona qualità con una resistenza a rottura di circa 30 N/mm^2 (v. Calcestruzzo, pag. 34). Armature di acciaio ad aderenza migliorata con una resistenza a rottura di circa 440 N/mm^2 (v. Acciaio, pag. 33)

Carichi. I pilastri sono soggetti alle forze verticali dovute al peso sovrastante (carichi permanenti e accidentali). Se l'edificio è in zona sismica i pilastri, le travi e il blocco scale-ascensori devono resistere alle forze orizzontali causate dal terremoto (v. La struttura nel suo complesso, pag. 32). I carichi verticali sono pari a circa 10 kN/m^2 per ogni piano. Se un pilastro sostiene una soletta di 5 m×5 m = 25 m^2, ogni piano contribuisce con 250 kN al carico sul pilastro.

Sezione. Se N è il carico verticale e A la sezione trasversale, $\sigma = N/A$ è la tensione di esercizio nel calcestruzzo per i carichi verticali. Si sceglie l'area A in modo che la tensione di esercizio si mantenga inferiore ai valori prescritti. Si ha che $A = A_c + m\,A_s$ con A_c area del calcestruzzo, A_s area delle armature e m (coefficiente di omogeneizzazione) uguale a 14. Inoltre $A_s \simeq 0{,}01\,A_c$, con un minimo di 4 d 12. (In presenza di carichi orizzontali è necessario tener conto dei momenti flettenti che sollecitano i pilastri.)

Dimensioni in funzione del carico

N kN	A_c cm^2	$a \times a$ cm×cm	$a \times 25$ cm×cm	A_s cm^2
250	321	18×18	15×25	4,4
500	642	25×25	25×25	6,4
750	963	31×31	39×25	9,6
1000	1285	36×36	51×25	12,9
1250	1606	40×40	64×25	16,1
1500	1927	44×44	77×25	19,3
1750	2249	47×47	90×25	22,5
2000	2570	51×51	103×25	25,7

Forma. Normalmente i pilastri sono rettangolari o quadrati (fig. A, B). In casi particolari sono a L, a C, a T. Lo spessore dei pilastri che stanno lungo il contorno dell'edificio dipende dallo spessore della muratura. Di solito, però, non si scende sotto i 20 cm. Le armature longitudinali vengono disposte negli angoli e, se occorre, lungo i lati della sezione ogni 30 cm circa. Il diametro delle armature longitudinali va da 12 mm a 20 mm e solo eccezionalmente fino a 26 mm. Le staffe seguono il contorno della sezione e sono disposte a un intervallo di circa 15 cm. Il diametro delle staffe va da 6 mm a 10 mm. Il copriferro sulle staffe deve essere non meno di 2 cm (v. Acciaio, pag. 33).

Dettagli costruttivi. Riprese di armatura da piano a piano (fig. C). Pilastri di sezione particolare (fig. D). Smussi negli spigoli: è opportuno prevederli di 2,5×2,5 cm. Negli edifici in zona sismica le armature dei pilastri e quelle delle travi devono essere opportunamente collegate.

Casseri e getti. Modalità di disposizione dei casseri (fig. E). Modalità di getto: normale ma con particolare cura nella vibrazione. Tempo di maturazione: circa una settimana per tutti i pilastri di un piano.

Quantità e costi. Incidenza delle armature: da 120 a 140 kg di acciaio per m^3 di calcestruzzo. Pilastro di 30 cm×40 cm, per piano di altezza 320 cm: calcestruzzo 0,38 m^3, acciaio 50 kg, casseri 4,5 m^2. Costo (2003): 150 €.

A Particolare della pianta architettonica

B Particolare delle strutture (vista speculare)

C Particolare di armatura

D Disposizione delle staffe in un pilastro a T

E Casseri di legno

Renato Villa

Figure 7: Second edition electronic version: a screen shot of the same spread shown in figure 6.

only the essential elements of the various engineering disciplines were included, and those condensed. The purpose is to give essential information to those who are not experts in a particular discipline, but want to understand the basic principles. This necessity arises also from the always increasing cooperation between engineers and architects, each expert in their respective fields. Therefore the Quick Reference meets the need of understanding one another and easing communication among technical staff.

The graphic solution by records (a spread in the printed version) came naturally in order to show in the most simple and schematic way the various subjects. The work had to have a synthetic and essential look. The importance of the technical drawings required a whole page for each subject, so the other page was dedicated to the text. This unusual arrangement was of great importance for the LaTeX structure and for the very management of the editing work.

AG had significant experience as the editor of Colombo's handbook with about one hundred and fifty collaborators, so he had no difficulty finding the forty plus collaborators for the Quick Reference. The difficulty was in the coordination and the choice of what to write in this Quick Reference and in which detailed form to write it in the records. A general idea of the text length was given to the authors, but it was the editor's task to cut down lengthy contributions or to request some additional material for excessively short records.

To coordinate a few dozen authors is an interesting task, because authors are very different from one another from both the scientific and a human point of view. They range from the very precise and pedantic personality to the excessively ingenious and chaotic one. All of them, though, are very interested to see their experience take form and become a real printed object. Moreover, they were passionate about extracting a synthesis of their knowledge for the book. Many of them are true professionals, but with little experience as technical writers. For these reasons the coordination task revealed itself as heavier that foreseen, but at the end the editor and the authors were very satisfied.

At the moment (2007) at the publishing house they are working on the third edition, whose publication should be around May 2008. At the beginning of 2007 the publishing house opened a new Internet site, `http://www.manualihoepli.it/`, intended to give information on the various published handbooks, and to publish integrations and updates that users may freely download. This site contains several pages from the Quick Reference, since they are evidently assumed to be both interesting and good general advertising on behalf of the Hoepli Publishing Company.

This third edition will contain few modifications, compared to the second one, the most relevant of which is an increase in size, so that it is possible to increase also the font size and the legibility.

10 Conclusion

The Quick Reference has been a wonderful occasion for both AG and CB to learn how to use to its best advantage such a powerful instrument as LaTeX. The book received a well deserved success so that, as said before, a third edition is under way; it contains not only upgraded information, but also the correction of some remaining typos.

Today, to create a similar work, from the point of view of the typesetter, would be much simpler, when one considers that most operations that were necessary to 'invent' when the first edition was published, are now mostly available with the countless packages, produced in the past 10 years, that have extended LaTeX's functionality.

Today the available packages allow us to create the necessary macros in a more orderly way, but most important, in a well-documented manner [5]; they allow building up typesetting structures in a simpler way by means of high level user commands made available by the extension packages: the graphic formats can be chosen more easily by means of the geometry package [8]; fonts can be chosen and configured with one statement, for example using the extended Times fonts [7], or the extended Palatino ones [6]; the style of titles and running titles can be defined with high level user commands [4]; the same is true for headers and footers [9]; even the handling of figure sets does not require the manual graphic editing as previously described [1].

Nevertheless all these packages, although extremely useful, if used by amateurs allow the creation of graphic designs that are not the best one can achieve.

Luckily, in this work the interaction between the LaTeX programmer and those giving the design specification had a particular positive synergy. This type of collaboration allowed setting reasonable limits to the graphic novelty involved, thanks to both the experience of AG and the interaction with CB who could indicate what was and was not feasible with LaTeX and the typesetting interpreter TeX.

Thanks to this cooperation between the editors, specifying unrealizable requirements for the graphic design and the page layout was avoided. In truth, even though LaTeX and TeX are extremely powerful,

they are computer programs that operate in batch mode, not interactively. This is their strength and their weakness, since they are not (yet) at the level of the modern interactive pagination programs mostly used by publishing companies, but are far superior in typesetting paragraphs and mathematical expressions.

References

[1] Steve Douglas Cochran. *The Subfig package*, 2005. Normally distributed with the TEX system, documentation in file `doc/latex/subfigure/subfig.pdf`.

[2] Giuseppe Colombo. *Manuale dell'ingegnere*. Hoepli Editore, Milano, 84 edition, 2003.

[3] Andrea Guadagni, editor. *Prontuario dell'ingegnere*. Hoepli Editore, Milano, 2 edition, 2003.

[4] Ulf A. Lindgren. *FncyChap*, 2005. Normally distributed with the TEX system, documentation in file `doc/latex/fncychap/fncychap.pdf`.

[5] Frank Mittelbach. *The doc and shortvrb packages*, 2006. Normally distributed with the TEX system, documentation in file `doc/latex/base/doc.pdf`; see also `doc/latex/base/docstrip.pdf`.

[6] Young Ryu. *The PX fonts*, 2000. Normally distributed with the TEX system, documentation in file `doc/fonts/pxfonts/pxfontsdocA4.pdf`.

[7] Young Ryu. *The TX fonts*, 2000. Normally distributed with the TEX system, documentation in file `doc/fonts/txfonts/txfontsdocA4.pdf`.

[8] Hideo Umeki. *The geometry package*, 2002. Normally distributed with the TEX system, documentation in file `doc/latex/geometry/manual.pdf`.

[9] Piet van Oostrum. *Page layout in LaTeX*, 2004. Normally distributed with the TEX system, documentation in file `doc/latex/fancyhdr/fancyhdr.pdf`.

◇ Claudio Beccari
`claudio dot beccari at gmail dot com`

◇ Andrea Guadagni
`andrea dot guadagni at tin dot it`

Suggestions on how *not* to mishandle mathematical formulæ

Massimo Guiggiani and Lapo F. Mori

Abstract

Quite frequently mathematical formulæ in reports, theses or even articles are not written correctly. Indeed, the basic rules for composing the formulæ are almost never explicitly stated. This article provides some suggestions to fill this gap.

1 Introduction

Formulæ are an excellent tool to express (some) mathematical concepts. This statement may seem obvious, but scientists realized it only after a long historical process. Natural language might at first seem easier to use than formulæ. However, scientists know very well that formulæ are an essential tool to communicate clearly, unambiguously and concisely. Formulæ are then part of a language and therefore should follow some rules. Unfortunately formulæ are often written "badly", that is not clearly, ambiguously, and not concisely.

The purpose of this article is to give some indications on how to write, or better on how *not* to write formulæ. In fact, children need to go to school, make mistakes, and be corrected to learn how to write in natural languages. Typically, uneducated people can communicate orally, but make many mistakes in writing. The same applies to mathematical formulæ. Only occasionally do people think about which rules to follow and which errors to avoid. The examples reported in the article are taken from publications and theses. Most suggestions and remarks have a general validity. However, LaTeX users will also find indications on how to implement them.

This article is mostly aimed to authors of technical/scientific documents (engineers, experimental physicists, chemists, and in general anyone who uses mathematics as a tool) and not to mathematicians. In pure mathematics there is much more attention to the proper typographic rendering of mathematical concept, and each field has its own rules (which do not necessarily comply with ISO standards (Gregorio, 2007)).

2 General rule

The general rule is very easy: formulæ must be unambiguous and concise, that is clear and simple. This is (or should be) true for many forms of communication, not only the written ones. When doing science, however, the writer should be even more careful about how to explain concepts in order to make them easy to understand. Further details on how to write mathematical formulæ can be found in the ISO 31 standard (ISO 31/11, 1982; ISO 31/12, 1982), which is discussed in depth in Wikipedia (2008a,b,c,d) and in Beccari (1997).

3 Operations

3.1 Multiplication

One of the most common errors is using the "dot" to indicate multiplication between scalars. For example the following formulæ

$$a \cdot x^2 + b \cdot x + c = 0, \qquad \sigma \cdot \varepsilon = 2 \cdot \alpha$$

should be written as

$$ax^2 + bx + c = 0, \qquad \sigma\varepsilon = 2\alpha.$$

In fact, for the sake of simplicity, the standard multiplication between letters, or between a number and a letter, does not require any symbol. If, on the other hand, the multiplication is between two numbers, the $\times$ or $\cdot$ symbols are required to avoid ambiguity. For example you should write

$$2 \times 3 = 6 \quad \text{and not} \quad 2\,3 = 6.$$

The same symbol should also be used when a line break occurs at a product, as shown in Eq. (1). The asterisk is absolutely not acceptable:

$$a * x^2 + b * x + c = 0, \qquad \sigma * \varepsilon = 2 * \alpha.$$

The dot has to be used to indicate a scalar product between vectors. For this purpose either a thin or a thick dot can be used. The first one is provided by `\cdot`, the second one by `\bcdot`, which is defined as

```
\newcommand{\bcdot}{\boldsymbol{\cdot}}
```

The two commands produce respectively

$$\mathbf{a} \cdot \mathbf{c} = 0 \qquad \text{and} \qquad \mathbf{a} \boldsymbol{\cdot} \mathbf{c} = 0.$$

3.2 Mathematical operators

Mathematical operators, including the differential, must be indicated with upright characters, as in the following examples:

$$\int \sin x \,\mathrm{d}x, \qquad \frac{\mathrm{d}y}{\mathrm{d}x}, \qquad \lim_{x\to 0} \cos x = 1.$$

They are indeed abbreviations of words and the upright font reminds us of this fact and avoids any ambiguity with products. For example, $\tan x$ is undoubtedly the tangent of x, while $tanx$ might be the product of four scalars.

LaTeX and the amsmath package define the most common mathematical operators, including `\lim`, `\sin`, `\min`, etc. In order to define new ones, the amsmath package provides the `\DeclareMathOperator` command, to be used only in the preamble, and the

\operatorname that allows one to define an operator directly in the text. For example

$$\operatorname{argmax} f(x)$$

can be obtained in two ways:

- declaring in the preamble

```
\DeclareMathOperator{\argmax}{argmax}
```

  and then using in the text

```
\argmax f(x)
```

- directly using in the text

```
\operatorname{argmax} f(x)
```

Defining an operator is different than using \mathrm[1] because the first one also takes care of the space between the operator and its argument. Note for example the difference between

$$\sin x \qquad \text{and} \qquad \mathrm{sin}x$$

obtained respectively with

```
\sin x
```

and

```
\mathrm{sin}x
```

Moreover, operators correctly handle superscripts and subscripts as for limits. For example

$$\lim_{x\to\infty} \frac{1}{x}$$

can be easily obtained with

```
\lim_{x\to\infty}\frac{1}{x}
```

A peculiar definition is required to properly write the differential symbol. It is in fact an operator that has a space only on its left. In Beccari (2007b) the following solution is proposed:

```
\newcommand{\ud}{\mathop{}\!\mathrm{d}}
```

It uses an empty operator and eliminates the space on its left with \!.[2] Note the difference between

$$\int \sin x \mathrm{d}x \qquad \text{and} \qquad \int \sin x \,\mathrm{d}x,$$

where the differential is obtained respectively with \mathrm{d} and \ud.

[1] \mathrm can be used to obtain an upright font in mathematical mode. The other methods to use an upright font in mathematical mode are described in par. 5.1.1.

[2] \! inserts a negative space as wide as the one that TEX inserts between an operator and a variable (Beccari, 2007a).

3.3 Multi-line equations

Long equations sometimes need to be split over several lines. The general rule is to indicate the operation or relation symbols *only once* at the end of the line for text style formulæ and at the beginning of the line for display style formulæ[3]

$$f = \frac{a}{b}\left[\sin\left(\frac{c}{d}\right) + \tan\left(\frac{c}{d}\right)\right] \times \left[1 + \sin\left(\frac{e}{f}\right) - \cos\left(\frac{g}{h}\right)\right]. \quad (1)$$

Operation and relation symbols should always be indicated only once because it is simpler and, more importantly, to avoid ambiguity. If, for example, you write

$$a = b + c + d- \\ - e - f,$$

it is not clear which of the following two you mean:

$$a = b + c + \boxed{d - e} - f$$
$$a = b + c + \boxed{d + e} - f.$$

This rule also applies to equalities:

$$\begin{aligned} a &= b + c + d \\ &= -e - f \\ &\simeq g. \end{aligned}$$

3.3.1 LaTeX commands

While LaTeX automatically splits long text style formulæ, the author has to do this manually with display style equations. The amsmath package provides several environments to split equations, such as split, multline, gather, align, aligned, alignat; refer to the amsmath package documentation (American Mathematical Society, 2002) for the details.

3.3.2 How to handle parentheses

When using the \left and \right commands to let LaTeX automatically choose the bracket size, problems may arise if the formula is split between \left and \right. For example the code

```
\begin{multline*}
   f(x,y,z)= (1+x+y-z)\left[\pi\\
      +\sin\left(\frac{a}{b}\right)
      +\cos\left( \frac{c}{d}\right)\right].
\end{multline*}
```

[3] Text style formula means a formula that appears inside a line of text (Knuth, 1992), such as $c = a + b$, while display style formula means a formula that occupies a line by itself (Knuth, 1992), such as

$$c = a + b.$$

The first ones can be obtained in LaTeX with the $...$ command, while the second ones can alternatively be obtained with the \[...\] command or the equation environment.

cannot be compiled because LaTeX expects all the `\left` to be closed by the respective `\right` inside each line.

A solution is to close `\left` and `\right` respectively with `\right.` and `\left.`

```
\begin{multline*}
   f(x,y,z)= (1+x+y-z)\left[\pi\right.\\
      \left.+\sin\left(\frac{a}{b}\right)
      +\cos\left( \frac{c}{d}\right)\right].
\end{multline*}
```

but now the bracket dimension for each couple `\left` and `\right` are computed independently and this can lead to awful results such as

$$f(x,y,z) = (1+x+y-z)\left[\pi + \sin\left(\frac{a}{b}\right) + \cos\left(\frac{c}{d}\right)\right].$$

A possible solution is to find the tallest term, in this case

$$\left(\frac{a}{b}\right),$$

and to insert it inside a `\vphantom` command in the other line

```
\begin{multline*}
   f(x,y,z)= (1+x+y-z)\left[\pi\vphantom{
      \left(\frac{a}{b}\right)}\right.\\
      \left.+\sin\left(\frac{a}{b}\right)
      +\cos\left( \frac{c}{d}\right)\right].
\end{multline*}
```

obtaining

$$f(x,y,z) = (1+x+y-z)\left[\pi + \sin\left(\frac{a}{b}\right) + \cos\left(\frac{c}{d}\right)\right].$$

Another solution is to use fixed height brackets[4] instead of looking for the tallest element. For example the code

```
\begin{multline*}
   f(x,y,z)= (1+x+y-z)\Bigl[\pi\\
      +\sin\Bigl(\frac{a}{b}\Bigr)
      +\cos\Bigl( \frac{c}{d}\Bigr)\Bigr].
\end{multline*}
```

gives

$$f(x,y,z) = (1+x+y-z)\Big[\pi + \sin\Big(\frac{a}{b}\Big) + \cos\Big(\frac{c}{d}\Big)\Big].$$

Note that this solution is applicable only to cases in which brackets are not taller than `\Bigg`.

In addition to the previous solutions, Michael Downes and Morten Høgholm developed the breqn package to automatically handle this problem. When the package is loaded, the `\left` and `\right` commands work properly even when the equation is split.

3.4 Punctuation

There are two approaches on how to handle external punctuation, that is, the punctuation in display style formulæ. Some authors think that it should never be used (Beccari, 2007a), others that it is necessary and essential.[5] Whichever of these two conventions you choose, the fundamental thing is to be consistent and always use the same method.

The authors of this article believe that formulæ, both in display and text style, are part of the argumentation and so punctuation should be used to help the reader. An example of good use of punctuation is:

Since
$$a = b$$
and
$$b = c,$$
it is proven that
$$a = c.$$

4 Variables

4.1 Vectors and matrices

4.1.1 Synthetic notation

Vectors are usually indicated with bold face lower case letters[6]

$$3\mathbf{a} + \alpha\mathbf{c} = \mathbf{v}.$$

Other notations such as

$$\vec{a}, \quad \underline{a}, \quad \{a\}, \quad [a]$$

should be avoided.

The same applies to matrices which can be indicated with upper case bold face letters

$$\mathbf{Ax} = \mathbf{b}$$

avoiding expressions such as

$$[A]\{x\} = \{b\}.$$

In a linear algebra text, which deals entirely with matrices and vectors, it is perfectly legitimate to get rid of the bold face and simply write

$$Ax = b.$$

[4] Fixed height brackets can be obtained with the commands `\big`x, `\Big`x, `\bigg`x, and `\Bigg`x (where x is 'l' for left brackets, 'r' for right brackets, 'm' or nothing for a regular symbol).

[5] Most mathematics books, new and old ones (including books typeset with metal movable type), use, although not always consistently, semicolon, comma, and full stop in display style equations (Bellacchi, 1894; Bertini, 1907; Bianchi, 1899; Bonola, 1906; Burali-Forti, 1904; Caprilli, 1912; Cesaro, 1894; Dini, 1878; Fubini, 1908; Gazzaniga, 1903; Peano, 1887; Sacchi, 1854; Veronese, 1891; Vivanti, 1916).

[6] Upright bold face characters in mathematical mode can be obtained with the `\mathbf` command.

4.1.2 Component notation

In some cases it is necessary to show explicitly vector and matrix components. Books and theses often report expressions like

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_2 \\ b_3 \end{Bmatrix}.$$

This notation should be avoided because it uses different brackets for matrices and vectors and also because it uses curly brackets for vectors. Much better is the following:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}. \tag{2}$$

Although there is no rule or customary practice on the type of brackets to use for vectors and matrices in component notation, it is advisable to:

- use either round or square brackets and avoid all the other types (curly, angle, etc.);
- maintain the same notation for both matrices and vectors[7] throughout the document.

Matrices and vectors should be written in LaTeX with the following environments provided by the amsmath package: pmatrix (for round brackets) and bmatrix (for square brackets). For example Eq. (2) can be obtained with

```
\begin{bmatrix}
    A_{11}&A_{12}&A_{13}\\
    A_{21}&A_{22}&A_{23}\\
    A_{31}&A_{32}&A_{33}\\
\end{bmatrix}
\begin{bmatrix}
    a_1\\ a_2\\ a_3
%%\end{bmatrix}
\end{bmatrix} =
\begin{bmatrix}
    b_1\\ b_2\\ b_3
\end{bmatrix}
```

When a column vector has to appear in the text in component notation, it is advisable to represent it as a transposed row vector in order not to add too much space between lines. There are at least two notations for row vectors in the text. The first one consists in using the notation for display style vectors. For example you could write $\mathbf{v} = [v_1\ v_2\ v_3]^T$ with

```
\mathbf{v} = [v_1\;v_2\;v_3]^T
```

The second one, adopted for example by Strang (2005), is more compact and consists in separating the component with commas without indicating the transposition, which is implicit. For example you could write $\mathbf{v} = (v_1, v_2, v_3)$ with

```
\mathbf{v} = (v_1,v_2,v_3)
```

In either case, the pmatrix and bmatrix environments should not be used since they would produce too big brackets.

The choice between round and square brackets depends only on personal taste.

4.2 One symbol, one letter

A mathematical symbol is usually indicated by *one* letter, not two or three. If, for example, we want to suggest that the *factor of safety* is equal to three, we should not write

$$FoS = 3$$

because FoS looks like the product between three scalars. It is much better to use a subscript

$$F_{\mathrm{s}} = 3.$$

CG is quite often used to indicate the center of gravity. Wouldn't G be better?

ISO standards admit some exceptions, such as Mach number Ma and Reynolds number Re, which are represented by two letters. In these rare cases in which the italic font is used for a symbol of several letters, the `\mathit` command should be used to avoid spacing problems between the letters. Note for example the difference between Ma and Ma, obtained respectively with `$Ma$` and `$\mathit{Ma}$`. In the first case the spacing between M and a might induce one to consider it as a product while in the second case the spacing is correct.

Formulæ should never include whole words such as

$$mass \times acceleration = force.$$

First of all, this should be a vector relation and the formula does not suggest it. Moreover the words should be written in an upright font

$$\mathrm{mass} \times \mathrm{acceleration} = \mathrm{force},$$

in order to have the right spacing between letters. However it is still not as clear and concise as

$$m\mathbf{a} = \mathbf{F}.$$

Another common error consists in using in a formula the same symbols employed in a computer program. For example

$$A\_fl + B\_fl = d\_k\_fl$$

instead of

$$A + B = d.$$

[7] In fact, in component notation, a vector is just a one column (or one row, depending on the convention adopted) matrix.

4.3 Superscripts and subscripts

Subscripts should be used only if necessary and, even in these cases, they should not be abbreviations. For example you should not write $\sigma_{\mathrm{nominal}}$ or σ_{nom}, where "nominal" has the regular English meaning; in this case you could simply write σ_{n}. Obviously it is allowed to use subscripts (or superscripts) for indices, as in a third order tensor σ_{ijk}. In this case mathematical italic characters should be used with the regular command `\sigma_{ijk}`.

Subscripts and superscripts follow the same rules that apply for symbols (par. 5.1); they must be in an italic font if representing physical quantities or mathematical variables, in an upright font otherwise. For example you should write A_T if T represents the temperature but A_{T} if T represents a name such as "trajectory". You should write A_i if i represents a summation index.

Using words as subscripts can disfigure even simple formulæ such as

$$T_{wheel,braking} = \frac{(T_{engine,supplied} - T_{engine,absorbed}) * C}{N}$$

which should have been written as

$$T_{\mathrm{b}} = \frac{C(T_{\mathrm{s}} - T_{\mathrm{a}})}{N}. \qquad (3)$$

Since the "b", "s", and "a" subscripts are abbreviation of respectively "braking", "supplied", and "absorbed", they must be in an upright font according to the previous rule.

4.4 Appropriate notation

Appropriate notation is fundamental to make a scientific text easy to understand. To realize how important the notation is, you can just try to compute a product with Roman numerals

$$\mathrm{MMCDXXVIII} \times \mathrm{XIX} \quad \text{instead of} \quad 2428 \times 19.$$

A common error consists in using different *types* of letters for quantities of the same kind; for example indicating some lengths with Latin letters (l, s, a) and others with Greek letters (α, γ) does not help the reader. Subscripts should be chosen carefully as well. Some books contain figures in which ω_1 is the angular velocity of body 2 and ω_2 that of body 1.

5 Physical and mathematical quantities

5.1 Symbols

ISO standards prescribe the use of italic symbols for all the physical and mathematical quantities that can assume different values; for example the names

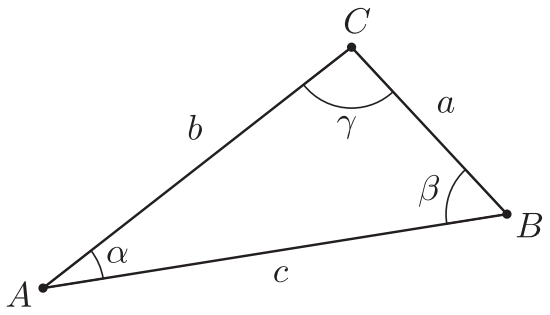


Figure 1: Example of mathematical variables indicated with italic letters.

of geometrical entities[8] (Fig. 1) and all the physics constants whose values are not "constant" because more accurate measures might change it.[9]

All the other mathematical and physical quantities must be written in an upright font. Among these the imaginary unit $2 + 4\mathrm{i}$, the base of natural logarithms $\mathrm{e} = 2.718\ldots$, and pi $\pi = 3.141{,}592\ldots$ (Beccari, 2007a).

As already mentioned in the introduction, "pure" mathematics does not always follow these rules (Gregorio, 2007).

5.1.1 Character types in LaTeX

In mathematical mode, LaTeX typesets letters in italic by default and so they represent non-constant mathematical and physical quantities. For example `$a=b$` gives $a = b$.

In order to write constants, that is upright letters in mathematical mode, the `\mathrm` command can be used. For example `$a=\mathrm{b}$` gives $a = \mathrm{b}$. `\mathrm`, although using upright characters, is still in mathematical mode so it ignores spaces. If spaces are needed, it is convenient to use the `\text` command from amsmath. Note for example the difference between

$$v_{\text{reduced section}} \quad \text{and} \quad v_{\mathrm{reducedsection}}$$

obtained respectively with

```
v_\text{reduced section}
```

and

```
v_\mathrm{reduced section}
```

[8] Points and angles, being mathematical variables, must be written in italic. Most authors indicate points with Latin upper case letters (A, B, C, etc.), segments with Latin lower case letters (a, b, c, etc.), and angles with Greek lower case letters (α, β, γ, etc.).

[9] For example the charge of the electron e, Planck's constant h, the reduced Planck's constant $\hbar$, and Boltzmann's constant k.

\mathrm, as opposed to \text, does not let superscripts and subscripts become italic in an italic context. For example

```
\textit{This produces $L_\text{eff}\ne L_\mathrm{
    eff}$}
```

gives

This produces $L_{\mathit{eff}} \neq L_{\mathrm{eff}}$

(Both \mathrm and \text with their arguments can be used directly as subscripts and superscripts as shown here; ordinarily, such multi-token sequences would need to be enclosed in braces.)

Upright Greek letters are provided by the upgreek package. Note the difference betwen $\upi$ and π obtained respectively with \uppi and \pi.

The previous examples are only meant to show how to use the various commands but, as explained in the previous paragraphs, superscripts and subscripts with full words should be avoided.

5.2 Numbers

Numbers must always be in an upright font (123), not in italic (*123*).[10] This is LaTeX's default behavior in mathematical mode, but problems might arise in the text. For this reason it is always advisable to write numbers in mathematical mode even in the text. If, for example, a number appears in an italic sentence, the mathematical mode enforces the upright font. For example

```
walk \emph{at most 2 km} north
```

gives

walk *at most 2 km* north

instead of the correct form

walk *at most* 2 *km* north

that can be obtained with

```
walk \emph{at most $2$ km} north
```

Lesina (1986) provides some rules for numbers in the text. They can be summarized in (Gregorio, 2005):

- *Numbers less than twenty*[11] must be in words. For example: the pinion has eleven teeth.
- If a number represents a *precise measure*, it must be in figures. For example: the long side is 1.5 m.
- If a number represents an *approximate measure*, it must be in words, unless the number is too long (when written). In this case it is easy to round it because no one is interested in knowing that a building is one kilometer or 1123 m distant (except in some specific cases). When reporting approximate numbers, the units must be explicitly written as well. You can write then "in about twenty meters" but not "in about 20 m".
- A number must be written in figures if it is coupled with another one that is in figures. For example you can write: there are 10 subjects of the first type and 154 of the second one.
- Never begin a sentence with a number written in figures.

Roman numerals, although quite rare in scientific texts, must always be written in an upright font, like the Arabic numbers. You should then write XVI instead of *XVI*.

Long numbers should have a comma every three digits:[12]

125,362

0.398,276

12.345,4.

In scientific documents the comma is usually substituted by a thin space (\,) (Wright, 2008a). Luckily this can be done automatically with the command \np of the numprint package. Here are some of the features of this package:

- Addition of a *separator every three digits*. The separator (, or . or \, or ~) depends on the language in use (defined by the babel package). The English separator is a comma , and so \np{15000000} gives 15,000,000.
- Substitution of the *decimal symbol*. Independently from the decimal symbol used in the LaTeX code, numprint represents numbers with the one that is appropriate for the language in use: . in English and , for the other European languages. For example, when writing in English, \np{3,15} becomes 3.15.
- *Approximation of decimal numbers* to the desired number of significant figures. If using three significant figures, \np{2,742647826672} becomes 2.743.
- Conversion of the E, e, D, and d characters into the *exponential format*.[13] For example \np{1,234E-4} becomes 1.234×10^{-4}.

[10] Dates are an exception to this rule and might be in italic if their context is.

[11] Other authors suggest numbers of at most one digit.

[12] Often four-digit numbers are not separated. Thus you would write 1234 but 12,345 (Wright, 2008a).

[13] This function is particularly useful when reporting data produced by software such as FORTRAN, MATLAB, etc.

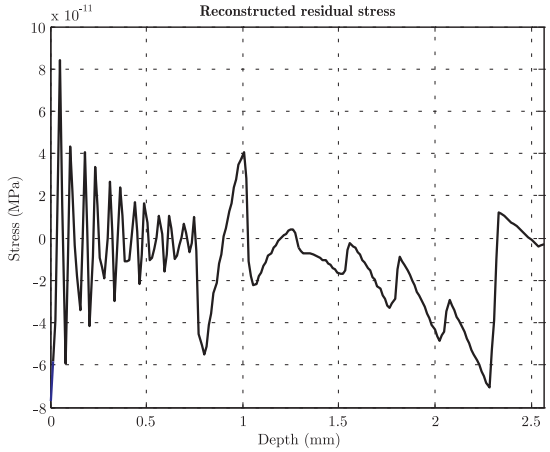


Figure 2: Units enclosed in round brackets.

- *Automatic addition of zeros*, if necessary. For example `\np{,12}` becomes 0.12.

5.3 Significant figures and scientific notation

Experimental sciences often express evaluations and results with numerical quantities. Some numerical quantities, such as π, e, $\sqrt{2}$, 2/3, are exact. Quantities deriving directly or indirectly from measures or estimates are, on the contrary, approximate. The representation, both in floating-point and in fixed-point, of approximate quantities is very important because it provides information about the precision with which the quantities are known. For example

$$h = 1.23\,\mathrm{m} \qquad \text{and} \qquad h = 1.230{,}00\,\mathrm{m}$$

might seem similar but have quite different meanings. In the first case h is a length known with a margin of ±5 mm and then it derives from quite inaccurate measures, in the second case the margin is ±5 µm and then it derives from very accurate measures.

Therefore, the number of significant digits of a numerical quantity is very important, sometimes even more important than the digits themselves.[14] Unfortunately, numerical quantities are very often represented with too many significant digits, especially when reporting results of calculations.

Scientific notation, which is a way of expressing approximate numbers with integer powers of ten, provides a concise notation and allows to immediately identify the number of significant digits.

5.4 Units

Units (ISO 1000, 1982; Bureau International des Poids et Mesures, 2006) are often made of multiple letters (kg, Pa, mm, MN, rad, etc.) and must follow some rules:

- they must be written with an upright font;
- they cannot be separated by a line break from the numeric quantity to which they refer;
- they cannot be enclosed in square brackets.

Square brackets cannot be used with units because in metrology they mean "unit of" (Beccari, 2007a). For example

$$a = 25\ \mathrm{m/s^2} \qquad \text{and} \qquad [a] = \mathrm{m/s^2}$$

can be written, but not

$$a = 25\ [\mathrm{m/s^2}].$$

On the other hand, round brackets can be used in tables and graphs when units appear next to a symbol of the corresponding physical quantity instead of the numeric value to which they refer (Tab. 1 and Fig. 2).

χ	C_1 (N/rad)	C_2 (N/rad)	a_1 (m)
0	73,000.0	90,000.0	0.912
0.05	70,899.3	89,143.7	0.912
0.10	68,565.1	88,850.5	0.912

Table 1: Units enclosed in round brackets.

Some conceptual errors can be added to the previous typographic errors. One of the most common is to indicate the second (unit of time) with "sec" (that means secant in trigonometry) instead of the symbol "s". A good guide on units is the manual of the siunitx package (Wright, 2008b). With this package the user does not have to format units by hand. For example

```
\unit{32.1}{\micro\meter}
```

gives

32.1 µm

The package writes the units with upright characters and adds an indivisible space between the number and the unit.

Another common error is using the upper case "K" to indicate kilo. In the context of units, "K" can only mean "Kelvin" while kilo is written with the lower case "k". Thus you should write kg, kB, etc. to indicate kilogram, kilobyte, etc. In general the name of the units of the International System have an upper case initial when named after scientists (e.g. N, W, Pa, J), lowercase initials in all other cases (e.g. m, lm, cd). The liter, which can be written both upper and lower case (l, L) in Europe and must be upper case in the USA (L), is the only exception to

[14] For example, in some contexts, there might be more difference between 2.1 and 2.150,2 than between 2.1 and 2.9.

this rule (Ceraolo, 2007; Bureau International des Poids et Mesures, 2006). (Also, the official spelling is 'litre', although 'liter' is typical in the USA, as seen in the present article.) When units are spelled out, they must always start with a lowercase letter, e.g. newton, siemens (Ceraolo, 2007; NIST, 2001).

6 Ambiguities

The digit "zero" should not be indicated by ∅ in handwriting. The context will clearly show that it is not a lower case "o". However, the symbol ∅ should be used when writing program code by hand because, in this particular case, it would be difficult to distinguish "O" from "0".

With electronic typesetting it is advisable to indicate liters with the capital letter "L" instead of the lower case "l". Although the standards allows both forms (ISO 1000, 1982; Giacomo, 1980; Bureau International des Poids et Mesures, 1979),[15] the lower case letter could be confused with the number one and the upper case "i". Note the similarity between the following: l, 1, I.

7 Conclusions

These notes are meant to contribute to reducing the "illiteracy" with which, sometimes, very interesting and profound mathematical formulæ are written. The importance of clarity and efficacy of communication, especially scientific communication, should never be underestimated.

8 Acknowledgments

We would like to thank Valeria Angeli, Luca Baldini, Riccardo Bartolozzi, Barbara Beeton, Marco Beghini, Karl Berry, Massimo Ceraolo, Andrea Domenici, Massimiliano Dominici, Beatrice Lazzerini, Caterina Mori, Pier Angelo Mori, Antonio Sponziello and Joseph Wright for their suggestions during both the writing and reviewing process of this article. We would also like to thank Claudio Beccari and Enrico Gregorio for their enlightening discussions on the GUIT forum (http://www.guit.sssup.it/phpbb/index.php) about some of the topics of this article.

Bibliography

American Mathematical Society. "User's Guide for the amsmath Package". 2002. ftp://ftp.ams.org/pub/tex/doc/amsmath/amsldoc.pdf.

Beccari, C. "Typesetting mathematics for science and technology according to ISO31XI". *TUGboat* **18**(1), 39–48, 1997. http://www.tug.org/TUGboat/Articles/tb18-1/tb54becc.pdf.

Beccari, C. Private communication, 2007a.

Beccari, C. *Introduzione all'arte della composizione tipografica*. 2007b. http://www.guit.sssup.it/downloads/GuidaGuIT.pdf.

Bellacchi, G. *Introduzione storica alle funzioni ellittiche*. Barbera, Firenze, 1894.

Bertini, E. *Introduzione alla geometria proiettiva degli iperspazi*. E. Spoerri, Pisa, 1907.

Bianchi, L. *Lezioni sulla teoria dei gruppi di sostituzioni*. E. Spoerri, Pisa, 1899.

Bonola, R. *La geometria non-euclidea*. Zanichelli, Bologna, 1906.

Burali-Forti, C. *Lezioni di geometria metrico-proiettiva*. Fratelli Bocca, Torino, 1904.

Bureau International des Poids et Mesures. "Resolution 6". In *Comptes Rendus de la 16^e Conférence Générale des Poids et Mesures*, page 101. 1979.

Bureau International des Poids et Mesures. *The International System of Units (SI)*. Pavillon de Breteuil, Sèvres, France, 8th edition, 2006.

Caprilli, A. *Nuove formole d'integrazione*. Belforte, Livorno, 1912.

Ceraolo, M. Private communication, 2007.

Cesaro, E. *Corso di analisi algebrica con introduzione al calcolo infinitesimale*. Bocca, Torino, 1894.

Dini, U. *Fondamenti per la teorica delle funzioni di variabili reali*. Nistri, Pisa, 1878.

Fubini, G. *Introduzione alla teoria dei gruppi discontinui e delle funzioni automorfe*. E. Spoerri, Pisa, 1908.

Gazzaniga, P. *Gli elementi della teoria dei numeri*. Drucker, Verona-Padova, 1903.

Giacomo, P. *Metrologia* **16**(1), 55–61, 1980.

Gregorio, E. Discussione sul forum di GUIT, 2005. http://www.guit.sssup.it/phpbb/viewtopic.php?t=1220.

Gregorio, E. Private communication, 2007.

ISO 1000. "SI units and recommendations for the use of their multiples and of certain other units". In *ISO Standards Handbook N. 2*. International Organization for Standardization, Geneva, 2nd edition, 1982.

ISO 31/11. "Mathematical sign and symbols for use in physical sciences and technology". In *ISO Standards Handbook N. 2*. International Organization for Standardization, Geneva, 2nd edition, 1982.

ISO 31/12. "Dimensionless parameters". In *ISO Standards Handbook N. 2*. International Organization for Standardization, Geneva, 2nd edition, 1982.

[15] The ISO standards allow the use of liters although the International System unit for the volume is m^3 and its multiples (Bureau International des Poids et Mesures, 2006).

Knuth, D. *The TEXbook*. Addison-Wesley, 1992.

Lesina, R. *Il manuale di stile*. Zanichelli, Bologna, 2nd edition, 1986.

NIST. *NIST Special Publication 330: The International System of Units (SI)*. 2001.

Peano, G. *Applicazioni geometriche del calcolo infinitesimale*. Bocca, Torino, 1887.

Sacchi, G. *Sulla geometria analitica delle linee piane*. Bizzoni, Pavia, 1854.

Strang, G. *Linear Algebra and Its Applications*. Brooks Cole, 4th edition, 2005.

Veronese, G. *Fondamenti di geometria a più dimensioni e a più specie di unità rettilinee esposti in forma elementare*. Tipografia del Seminario, Padova, 1891.

Vivanti, G. *Elementi della teoria delle equazioni integrali lineari*. Hoepli, Milano, 1916.

Wikipedia. 2008a. http://en.wikipedia.org/wiki/Typographical_conventions_in_mathematical_formulae.

Wikipedia. 2008b. http://en.wikipedia.org/wiki/ISO_31.

Wikipedia. 2008c. http://en.wikipedia.org/wiki/Mathematical_notation.

Wikipedia. 2008d. http://en.wikipedia.org/wiki/Mathematical_symbols.

Wright, J. Private communication, 2008a.

Wright, J. *siunitx: a comprehensive (SI) units package*, 2008b. http://ctan.org/get/macros/latex/exptl/siunitx/siunitx-manual.pdf.

◇ Massimo Guiggiani
Dipartimento di
Ingegneria Meccanica,
Nucleare e della Produzione
Università di Pisa
Pisa, Italy
guiggiani (at) ing dot unipi dot it

◇ Lapo F. Mori
Mechanical Engineering Department
Northwestern University
2145 Sheridan Road
Evanston IL 60208
USA
mori (at) northwestern dot edu

Electronic Documents

Wikipublisher: A Web-based system to make online and print versions of the same content

John Rankin

Abstract

Web pages and print documents exist as two solitudes: information created as a Web page may print poorly; information created as a print document may translate into an unappealing Web page. The Wikipublisher system lets authors create content online first, as Web pages, and lets readers turn individual pages or page collections into print documents. It uses wiki software as a lightweight and extensible content management system, so any page can be edited using any Web browser. It then uses LaTeX as the typesetting engine, thus providing print output of the highest quality. This paper examines the reasons for developing Wikipublisher, techniques and challenges faced in transforming Web content into print, and some wishes for the future. The project's home is `www.wikipublisher.org`.

1 Web-centred text

Wikipublisher was born of frustration over the gap between how I wanted to work and the capabilities of the tools I was using. In discussing this with like-minded colleagues, we came to the view that by re-thinking how we created and published information, we could be more productive and effective. We expressed this aspiration as a set of principles which underpin and inform every aspect of Wikipublisher's design:

Online first The World Wide Web enhances our ability to communicate, so most of our work ought to appear online first. Creating content online first makes it instantly and widely accessible, and encourages linking to other online resources. Yet most of our authoring tools are "print first" and turning print documents into HTML for publishing on the Web is hard to do well. Too many long documents are simply posted to the Web as PDF files. The links in an online document make it easy to navigate, yet print first authoring tools do little to encourage rich inter-document linking. So the first requirement is a system with support for the direct creation and editing of Web pages.

Print still matters If a Web page is worth reading, it is worth printing. The longer and richer the content, the more likely the reader is to print it. We may skim read a 50 page report online, but if we want to study it, we print it. If we want to deliver a printed and bound version, it needs to look good and be laid out for optimal readability. Yet few Web site designs appear to care what the printed form of the site looks like. Most appear to be designed assuming all the information on the site can be chunked into short, easily digested pieces. As readers, experience has taught us to have low expectations of the printed Web page.

One authoritative source The most up-to-date version is what appears on the Web page; the typeset PDF is a snapshot taken at a point in time. This means the printed page can never be newer than the Web page. This is in direct contrast to most publishing systems, where there are often 3 (and sometimes more) versions: the word processing or other source, a PDF snapshot of the word processing source, and a collection of Web pages generated (perhaps with edits) from the source. The more frequently the content changes and the more authors involved in creating it, the more important it becomes to have one source.

Wikipublisher is designed for people who write long, complex, richly linked documents, who wish to publish these in an accessible form on the Web and in print. The reader presses a "Typeset" button on the Web page and the system returns a PDF. By using LaTeX as the typesetting engine, Wikipublisher produces printed output of the highest quality.

2 We built it because ...

My company, Affinity Limited, is a small IT management consultancy. Like most professional services firms, our document production systems, based on Adobe FrameMaker, were geared to efficiently producing print documents such as letters, proposals, short papers, and reports. Our Web presence was essentially an online brochure — who we are, what we do, and how to contact us. Communication with clients primarily used e-mail to send document attachments back and forth. Several years ago, three things happened:

- we decided to make our work processes Web-centric wherever possible;
- open source wiki software became widely available, easy to install, and easy to use;

- Adobe decided not to offer a Mac OS X version of FrameMaker.

Apple's switch to Intel processors meant that OS 9 was dead and with it FrameMaker on Mac. Either we had to switch to Windows or find a replacement. We concluded that the least-worst alternative was to use LyX as a front-end to LaTeX, and while this would meet our print needs, it didn't really advance our aim to be more Web-centric. We had received very positive comments from clients when we introduced wiki software for some of our work. For example, when writing up interview notes, it is much easier and faster for everyone if we send a link to a Web page of the write-up, which the interviewee can edit using a Web browser. Wikis use simple textual markup to describe a page, which gets translated into HTML when the page is displayed. Each page carries an Edit link, which when clicked displays the content of the page as an editable Web form.

However, we have to deliver the interview notes as part of a printed and bound report, typically as an appendix. We experimented with "printable views" of the Web pages and quickly learnt that while the quality is good enough for an appendix, HTML plus CSS (cascading style sheets) produces printed output of a quality far lower than that produced by software such as FrameMaker or LaTeX. We also felt that it ought to be a lot easier to bundle up 10 interview Web pages into a single printed document. So we had succeeded in improving the doing part of the process, but at the cost of lowering the final output quality, and we still needed a solution for the main body of our reports. We concluded that if we were to make further advances in this direction, we needed a way to turn a Web page or page collection into a print document of at least the same quality as one expects from a modern word processor, with minimal manual intervention.

We decided to investigate the possibility of creating a report as a set of linked wiki Web pages and using some form of typesetting engine to repurpose this into a printable document, complete with cover page, table of contents, running headers and footers, and other print-oriented structures. With the help of a research and development grant from New Zealand's Foundation for Research, Science and Technology (FRST), we teamed up with the School of Mathematics, Statistics and Computing Sciences at Victoria University of Wellington to carry out the project. The grant allowed us to hire a graduate student on a part-time basis for just over a year, while he worked on a MSc. After a further round of enhancements and bug fixes, we can now produce letters, articles, reports, presentations and books, including images, tables, equations, citations and bibliographies — in other words, it has become a fully-fledged publishing system for Web and print documents. We have released the source code (a plug-in for the wiki engine and a separate typesetting server) and discovered that a surprising (to us) number of people find it useful.

3 TeXnical matters

Without TeX, the project would have been impractical. We considered using XSL-FO (formatting objects), but our not very scientific assessment found that at the time, none of the books on FO had been written using FO. We suspected that somewhere between the beautiful examples in the books and typesetting an arbitrary Web page, we would encounter impenetrable problems. TeX seemed to us a low risk and proven approach. Various books and PDF documents on LaTeX written with LaTeX gave us confidence that any problems we might find would have ready solutions.

3.1 Typesetting becomes a Web service

Wikipublisher works by re-purposing Web pages to a form of XML designed to describe printed material and then transforming the XML into LaTeX for typesetting. It builds on the following open source projects.

PmWiki We chose the PmWiki engine[1] because it is *markup agnostic* — that is, the administrator can control the input markup for structuring the content *and* the output markup the engine produces. The wiki continues to generate HTML for normal browsing, but when the reader requests a PDF document, we invoke a Wikipublisher plug-in which causes the wiki engine to generate Wikibook XML instead. The wiki's design means we can do this without any changes to the original source code.

tbook The tbook project[2] is an XML-based system designed to allow an author to create documents in XML and then transform these into a variety of different formats, including LaTeX. The original tbook system did not accommodate all the markup available in PmWiki, so we have extended the XML syntax quite significantly. We call the revised DTD (document type definition) "Wikibook" — any valid tbook XML document is valid Wikibook, but Wikibook supports constructs not found in tbook.

[1] www.pmwiki.org
[2] sourceforge.net/projects/tbookdtd

When a reader issues a request for a PDF, this goes to the Wikibook server, which asks the PmWiki server to generate the requested page(s) as Wikibook XML. The Wikibook server transforms the XML into LaTeX and thence into a PDF, which it returns to the reader. All an author or reader needs is a Web browser and PDF viewer; everything else happens on a Web server.

For sites requiring equations, we provide a plugin for PmWiki that lets authors write TeX equations into a page, including automatically-generated equation numbers if required. On the Web, each equation is transformed into an image; in the PDF, the equations are just part of the document. We use the open source latexrender php library for this.[3]

3.2 Headings determine structure

Authors generally use heading markup to structure their pages. In HTML, there are no controls over how headings are used. Wikipublisher assumes that a page has sections, subsections and subsubsections; whatever 3 kinds of heading it finds on the page, it maps into this structure. In other words, it transforms the absolute HTML and wiki heading levels into relative section levels. This means different headings on different pages can be rendered the same way in print. PageA might start with `<h2>` while PageB starts with `<h3>`: both become sections. It also means the same heading might be rendered differently in different print contexts. If a page is being typeset as part of a list of pages, each page becomes a section, so the first heading on the page now becomes a subsection.

In other words, different authors can use different heading conventions (or the same author may use different conventions at different times), and let Wikipublisher sort it out and produce consistently laid out print documents. An author can create a long structured document using multi-level lists, like this (each item is a separate wiki page):

- Section A
  - Subsection A.1
  - Subsection A.2
- Section B
  - Subsection B.1
  - Subsection B.2

And so on. On a page like Subsection A.1, the first 3 heading levels now become subsubsection, paragraph and subparagraph. A list item can also be designated as an appendix, which means it and subsequent items are unnumbered.

[3] `www.mayer.dail.pipex.com/tex.htm`

3.3 Adapt the output on demand

Suppose that some of the readers want output on US letter paper while others need A4; some have duplex printers while others do not; some prefer indented paragraphs while others prefer space between paragraphs. Perhaps the author wishes to include a "draft" watermark. Wikipublisher provides a `<meta>` XML element with name/value attribute pairs to control these and other settings. An "options" button on the typesetting request form lets the reader control the look of the finished document, over-riding the default meta settings. A Canadian reader with a duplex printer and an Australian reader with a one-sided printer can each request the output he or she prefers.

The print metadata capability does the simple things like creating mirrored headers and footers on odd and even pages for duplex printing, or making all headings serif. It also does more sophisticated things; for example, if a reader requests A5 paper, it not only reduces the font size to compensate for the shorter line length, it also makes sure any images are shrunk, if they are too big to fit the smaller page. Indeed, whatever the page size, it makes sure the images fit the page.

While many of these print metadata settings are common to all document classes, such as choosing the fontset or watermark, some are class-specific. For example, when typesetting a book, the reader can choose the chapter heading style; when typesetting a letter, the reader can include or omit a return address; when typesetting an article, the reader can choose whether or not to number the sections.

3.4 Citations and bibliographies

Because the wiki content is "online first", we cannot use any of the traditional bibliography tools like natbib. The problem we face is that LaTeX bibliography tools assume the author writes some variant of `\cite{key}` and LaTeX works out how to render the reference and sort the bibliography. On the other hand, to render a Web page with citations and a bibliography from wiki markup, the wiki engine has to solve these and other problems, then Wikipublisher has to tell LaTeX that we already know how to typeset the result. The wiki way to add a new entry to a bibliography is to cite it, just as the way to create a new page is to link to it:

- if the cited key exists, the wiki links to that entry in the page's bibliography
- otherwise, it links to a "new citation" form — fill in the form and press Save

The Wikibook XML we generate contains everything LaTeX needs to know — the keys, the text of the link (numbered or author–year), the items (labelled) in the bibliography, sorted in the correct order. All LaTeX has to do is typeset the data. For example, the body text might contain:

```
<cite kind="sic" refid="Smith:2001">see
Smith (2001, p.6)</cite>
```

The corresponding bibliography entry text might be:

```
<item id="Smith:2001">Smith, A. 2001. ...
</item>
```

In the PDF, we want the `<cite>` to link to the `<item>` and we want the item to print with a hanging indent. To achieve this, we created a new LaTeX command:

```
\citesic{see Smith (2001~p.6)}{Smith:2001}
```

and a new kind of list environment for a preformatted, presorted bibliography, containing:

```
\abibitem{Smith:2001} Smith, A. ...
```

The `\citesic` command uses the `\hyperlink` command and `\abibitem` uses `\hypertarget` from the hyperref package. We use a similar approach for numbered references, except the `<item>` includes an attribute `style="n"` where `n` is the number of the reference.

Finally, Wikipublisher invites URL addresses to break on selected special characters, avoiding ugly white space in bibliographies (or indeed in regular text) containing references to Web sites. We achieve this by substituting `<discy kind="x" />` in the URL link text, where `x` is: hyphen (`-`), full stop (`.`), equals (`=`), underscore (`_`), or forward stroke (`/`); and generating:

```
\discretionary{}{x}{x}
```

This means the hyphenation character starts the line following the break point, thereby avoiding the ambiguity of an address line ending in a hyphen, which could cause a reader to wonder whether the hyphen is part of the address or was inserted during text hyphenation.

3.5 Tables

Wikipublisher currently provides support for three kinds of tables:

- simple tables float, their column widths are determined automatically, and text in the cells can be left justified, centred, or right justified; the caption, if there is one, is numbered
- long tables are like simple tables, except that they do not float and if the first row is headings (the wiki markup for the HTML `<th>` tag), this becomes a running header on the second and subsequent pages
- complex tables can contain any wiki markup the author deems it necessary to use (including simple tables); to handle these, we wrap the cell content in a LaTeX minipage environment, set all cell text to ragged right, and hope; if the author supplies percentage cell widths, we use these, otherwise we make the columns equal width

Authors are generally used to the fluid nature of wiki and HTML tables, and tend to assume that anything a Web browser can handle, the typesetting engine ought to handle too. By changing HTML attribute values, authors have fine control over table ruling, shading, colouring, and spacing on a Web page. There may also be site-wide table styles set in an external CSS file. Wikipublisher ignores almost all of these settings, and instead sets what we hope are reasonable and consistent defaults. In essence, it takes the *structure* of the table and ignores the *style*. This can be a strength, rather than a weakness — all the tables in a long document will look similar, whereas the original Web tables may show a great deal of stylistic variation.

We are investigating ways to handle rotated tables, especially long (multi-page) tables. The aspect ratio of modern computer screens is typically greater than 4:3 (the screen I am using at the moment is 1.6:1), so cell column widths which look fine through a Web browser tend to be too narrow on a portrait A4 page. Ideally, an author could assign a "wide" class to a table and Wikipublisher would print it rotated 90°, automatically splitting it across several pages if necessary, with column widths calculated from the available text height.

4 If we had a magic wand ...

And could make three wishes, what would we like to improve? Most of the time, "it just works"; once Wikipublisher is installed and configured, users can forget it is there until they need it, and when they need it, they get what they need, although this may not be precisely what they think they want.

4.1 Colour cite links correctly

I wish we could tell the hyperref package that the special `\citesic{link text}{key}` command we have defined is just a variation on the `\cite{key}` command, even though it invokes the `\hyperlink` command. At the moment, if you run Wikipublisher with the colorlinks option turned on, citation links come out in red (`linkcolor`) instead of green (`citecolor`). There is probably a simple way

to modify the `wikibib.sty` file to check whether `colorlinks` is on and if so, to use `citecolor` for the colour of the generated hyperlink. Or preferably to define the `\citesic` command in such a way that hyperref recognises it as a citation link and colours it correctly. Any reader thinking "Oh, that's *easy*" might like to contact me. The `\citesic` command is defined as follows:

```
\newcommand\citesic[2]{\hyperlink{#2}{#1}}
```

4.2 Publish with style

I wish we had a general method for mapping the wiki's HTML style information into equivalent LaTeX print structures. Here are some of the things people do with wiki styles which at the moment do not work properly in Wikipublisher.

1. Define a style with a light green background, a dotted dark green border, text aligned right, with 0.5 em padding, applied to a paragraph.
2. Define styles for text decoration underline and line-through, even though there is a perfectly good structural markup to designate inserted, deleted, and highlighted text.
3. Set up "zebra tables" where alternate rows or columns are shaded. Combine alternate row and column stripes to create "hatched tables". Define cell ruling and padding.
4. Float text around a table using the table attributes `align="left"` or `align="right"`. Apply the `rowspan="n"` attribute to a table cell (although `colspan` works, `rowspan` does not).

An author can use wiki markup to define a named style consisting of any combination of HTML style attributes, and apply this style to any block, line or inline piece of text. Wikipublisher correctly handles styles like text size and colour, background colour, list item numbering and numbering style, and text alignment. There is a project awaiting us to analyse all the HTML style attributes, map these to their LaTeX equivalents (where these exist), and systematically define attribute transformations for any validly-styled text.

4.3 Typeset any Web page

I wish we could typeset pages from MediaWiki,[4] WordPress,[5] and other Web sites. In principle, if we can transform wiki markup, we ought to be able to transform HTML into Wikibook XML and then on into LaTeX. While this is superficially attractive, there are also some obvious problems. The HTML syntax makes no provision for semantic elements such as:

- figure captions
- footnotes
- marginal notes
- citations and references
- chapters, sections, subsections, appendices and other parts of a book

It also includes the widely used (and meaningless) `<div>` and `<span>` tags. In practice, one would have to rely on inferring structure from site-specific conventions for how class attributes are used. For example, if an image has `alt` text (as images should), use that as the caption. However, as more Web sites become database-driven, rather than using handcrafted HTML, we can envisage developing different handlers (sets of rules) for different kinds of content management software, controlling the actions of a general-purpose HTML to Wikibook conversion engine. For example, suppose a university publishes an online journal in HTML or if *TUGboat* were to publish online first. A reader could choose articles of interest, from several online volumes, and request these as a single typeset PDF file. Potentially, high quality "print runs of one" become economic, perhaps with the ability to use a wider choice of LaTeX document classes than those currently supported.

We plan to seek some research and development funding to pursue this proposal. Unfortunately for us, FRST allocated almost all its 2007/08 grants budget in the first three months of the financial year and is only funding big companies at the moment. We have to wait until the 2008/09 financial year to apply.

5 Summing up

This article has described an open source software project which combines easy-to-use wiki software with the typesetting capabilities of LaTeX to create a simple, flexible, and powerful collaborative authoring and publishing system. Before the project started, I was an absolute LaTeX beginner; I have learnt how it works by looking at what the tbook XSL transformation did and watching how Donald Gordon, our part-time developer, enhanced this to handle the PmWiki markup set.

How does a small business justify investing this level of effort, when it would be easier to use Microsoft Word like almost everyone else? It is said that businesses change direction out of fear, greed, or boredom. It is true that we have won additional business as a result of the work, but not as much as we had hoped. It is also true that our clients see

[4] `www.mediawiki.org`
[5] `wordpress.org`

the project as innovative and pretty cool, so it has enhanced our reputation. As a research and development project it was a success, but commercialising the investment has been problematic. Currently, it is not generating revenue for us, but it has never been boring, it makes our working lives simpler, and it is a lot more fun than many of the things I get paid to do.

The project has taught us three important and in hindsight self-evident lessons.

Publishing online first alters the rules Like most other project-based consultancy practices, we write regular progress reports to our clients. These used to be paper letters; now they are wiki Web pages which can be typeset as letters. The shift from "documents" to "pages" creates a new perspective on the content. For example: they can contain links to related materials; the client can annotate them with comments; the entire history of the project is instantly accessible and searchable; and we can take print snapshots to store in a document management system or bind for physical distribution.

Open beats closed Using open source (free) software reduces the barriers to entry, allowing us to stand on the shoulders of giants. Open standards, especially the XML set of standards, enable interoperability between disparate systems like PmWiki and LaTeX. Open means that when we strike a problem, Google knows the solution: someone will have seen the problem before and left a trail on the Web for us to follow. The wiki markup and file format specifications are open and fully documented, so long term content curation and preservation are simply not a problem. In contrast, when we replace our last PowerPC-based Mac, a decade's worth of documents, stored in FrameMaker's proprietary format, will become inaccessible, unless we first convert them to a format like rtf.

Most people are not interested It is a minority of people who share our enthusiasm for what Wikipublisher can do. People who are familiar with HTML often do not see the point of transforming Web content into LaTeX to create a high quality print document. They see print as a disposable afterthought and consider that generating a printable page view using CSS is good enough. I find it surprising that people who pay careful attention to accessibility and readability principles for Web sites are happy to ignore these for printed material. On the other hand, most authors are comfortable with their word processor and see no reason to change their practice. If what they are writing will be published on a Web site, converting the document to HTML is someone else's problem. When their document is finished, they toss the dead sheep over the fence into the next paddock and forget about it.

The success of Microsoft Office in bringing typesetting to the mass market has had the unfortunate side-effect of entrenching typographic mediocrity in our culture. Most people are unaware of, and hence do not value, the correct use of ligatures, text justification algorithms, inter-paragraph separation, and the many other details which TeX looks after on our behalf. Competitors such as OpenOffice and Google Office are often judged by how well they replicate the layout of Word documents. So in a sense, Wikipublisher is over-engineered — it does a better job than it needs to.

If you are interested in Wikipublisher, you can try it online at `www.wikipublisher.org`, using any Web browser, or download the software and install it on any Unix or Linux server. I have PmWiki and the Wikibook PDF server running on my Apple MacBook. If you have any questions, you can contact me via the web site or the address below.

◇ John Rankin
Affinity Limited, Wellington, New Zealand
`john dot rankin (at) affinity dot co dot nz`

Character encoding

Victor Eijkhout

Have you ever wondered what goes on between the 'A' you hit on your keyboard, the 'A' stored in your file, and the 'A' that comes out of your printer? Why does that letter still come out of the printer if the file is printed by your friend in Egypt who doesn't use the letter 'A'? Maybe you know that 'A' is character 65 (decimal) in ASCII; if you put it on a web page, and it's visited by someone in Japan, why don't they get character number 65 in the Kanji alphabet? Do you remember the DOS days when your Mac owning colleague would send you a file and what were supposed to be accented characters would turn into smiley faces? Have you ever pasted text from MS-Word into Emacs, and Emacs wanted to save the document as UTF-8? Just what is that about?

All this, and more, will be explained in this article.

1 History in one byte

Somewhere in the depths of prehistory, people in the Western world agreed on a standard for character codes under 127, ASCII, the American Standard Code for Information Interchange. This standard declares that the letter 'A' is character number 65 decimal (41 in hexadecimal), so if your file contains the bit pattern for 65 (which is `01000001`), it will produce an 'A' when sent to the printer.

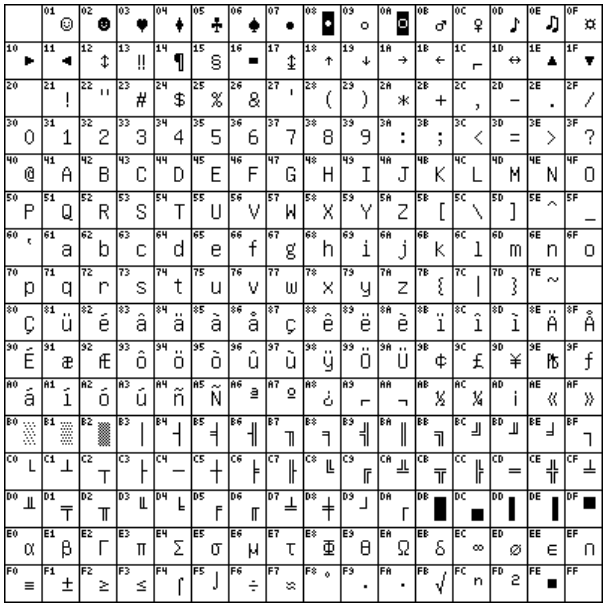
ASCII has some nice properties, some of which were lacking in another encoding scheme, EBCDIC (which was used almost exclusively by IBM):

- All letters are consecutive, making a test 'is this a letter' easy to perform.
- Uppercase and lowercase letters are at a distance of 32; this means that the Shift key on your keyboard simply toggles the sixth bit in the pattern of whatever key you are holding down.
- The first 32 codes, everything below the space character, as well as position 127, are 'unprintable', and can be used for such purposes as terminal cursor control.

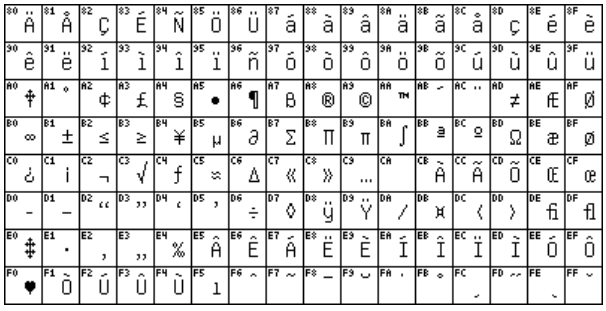
The ISO 646 standard codified 7-bit ASCII, but it left certain character positions (or 'code points') open for national variation. For instance, British usage put a pound sign (£) in the position of the dollar. The ASCII character set was originally accepted as ANSI X3.4 in 1968. ANSI is displayed in table 1.

Since a computer organizes its bits in 8-bit bytes, and ASCII only codified the codes under 128, this left the codes with the high bit set ('extended ASCII') undefined, and different manufacturers of computer equipment came up with their own way of filling them in. These standards were called 'code pages', and IBM gave a standard numbering to them. For instance, code page 437 is the MS-DOS code page with accented characters for most European languages, 862 is DOS in Israel, and 737 is DOS for Greek.

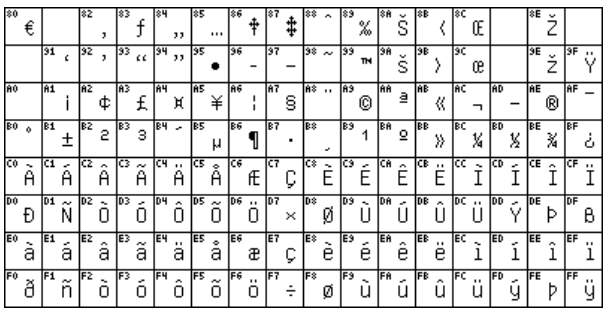
Here is cp437:



MacRoman:



and Microsoft cp-1252:



More code pages are displayed in [5].

ASCII CONTROL CODES

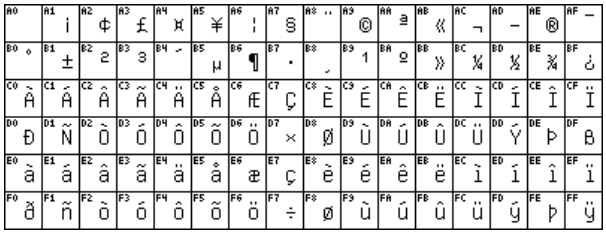
dec CHAR hex oct

b7 b6 b5 BITS b4 b3 b2 b1	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
	CONTROL		SYMBOLS NUMBERS		UPPERCASE		LOWERCASE	
0 0 0 0	0 NUL 0	16 DLE 20	32 SP 40	48 0 60	64 @ 100	80 P 120	96 ‘ 140	112 p 160
0 0 0 1	1 SOH 1	17 DC1 21	33 ! 41	49 1 61	65 A 101	81 Q 121	97 a 141	113 q 161
0 0 1 0	2 STX 2	18 DC2 22	34 ” 42	50 2 62	66 B 102	82 R 122	98 b 142	114 r 162
0 0 1 1	3 ETX 3	19 DC3 23	35 # 43	51 3 63	67 C 103	83 S 123	99 c 143	115 s 163
0 1 0 0	4 EOT 4	20 DC4 24	36 $ 44	52 4 64	68 D 104	84 T 124	100 d 144	116 t 164
0 1 0 1	5 ENQ 5	21 NAK 25	37 % 45	53 5 65	69 E 105	85 U 125	101 e 145	117 u 165
0 1 1 0	6 ACK 6	22 SYN 26	38 & 46	54 6 66	70 F 106	86 V 126	102 f 146	118 v 166
0 1 1 1	7 BEL 7	23 ETB 27	39 ’ 47	55 7 67	71 G 107	87 W 127	103 g 147	119 w 167
1 0 0 0	8 BS 10	24 CAN 30	40 (50	56 8 70	72 H 110	88 X 130	104 h 150	120 x 170
1 0 0 1	9 HT 11	25 EM 31	41) 51	57 9 71	73 I 111	89 Y 131	105 i 151	121 y 171
1 0 1 0	10 LF 12	26 SUB 32	42 * 52	58 : 72	74 J 112	90 Z 132	106 j 152	122 z 172
1 0 1 1	11 VT 13	27 ESC 33	43 + 53	59 ; 73	75 K 113	91 [133	107 k 153	123 { 173
1 1 0 0	12 FF 14	28 FS 34	44 , 54	60 < 74	76 L 114	92 \ 134	108 l 154	124 | 174
1 1 0 1	13 CR 15	29 GS 35	45 − 55	61 = 75	77 M 115	93] 135	109 m 155	125 } 175
1 1 1 0	14 SO 16	30 RS 36	46 . 56	62 > 76	78 N 116	94 ^ 136	110 n 156	126 ~ 176
1 1 1 1	15 SI 17	31 US 37	47 / 57	63 ? 77	79 O 117	95 · 137	111 o 157	127 DEL 177

Table 1: The ASCII table

The international variants were standardized as ISO 646-DE (German), 646-DK (Danish), et cetera. Originally, the dollar sign could still be replaced by the currency symbol, but after a 1991 revision the dollar is now the only possibility.

The different code pages were ultimately standardized as ISO 8859, with such popular code pages as 8859-1 (‘Latin 1’) for western European:



8859-2 for eastern European, and 8859-5 for Cyrillic:

A0	A1 Ё	A2 Ђ	A3 Ѓ	A4 Є	A5 Ѕ	A6 І	A7 Ї	A8 Ј	A9 Љ	AA Њ	AB Ћ	AC Ќ	AD -	AE Ў	AF Џ
B0 А	B1 Б	B2 В	B3 Г	B4 Д	B5 Е	B6 Ж	B7 З	B8 И	B9 Й	BA К	BB Л	BC М	BD Н	BE О	BF П
C0 Р	C1 С	C2 Т	C3 У	C4 Ф	C5 Х	C6 Ц	C7 Ч	C8 Ш	C9 Щ	CA Ъ	CB Ы	CC Ь	CD Э	CE Ю	CF Я
D0 а	D1 б	D2 в	D3 г	D4 д	D5 е	D6 ж	D7 з	D8 и	D9 й	DA к	DB л	DC м	DD н	DE о	DF п
E0 р	E1 с	E2 т	E3 у	E4 ф	E5 х	E6 ц	E7 ч	E8 ш	E9 щ	EA ъ	EB ы	EC ь	ED э	EE ю	EF я
F0 №	F1 ё	F2 ђ	F3 ѓ	F4 є	F5 ѕ	F6 і	F7 ї	F8 ј	F9 љ	FA њ	FB ћ	FC ќ	FD §	FE ў	FF џ

These ISO standards explicitly left the first 32 extended positions undefined.

Reading material: The history of ASCII out of telegraph codes [1]; a history, paying attention to multilingual use [4]; Bob Bemer, the 'father of ASCII' [2]; a detailed discussion of ISO 8859, Latin-1 [11].

2 Character sets and encodings

As you can tell from the introduction, there is quite a bit of confusion possible between characters and representations or encodings. Let us clear up the concepts a little.

Informally, the term 'character set' (also 'character code' or 'code') used to mean something like 'a table of bytes, each with a character shape'. With only the English alphabet to deal with that is a good enough definition. These days, much more general cases are handled, mapping one octet into several characters, or several octets into one character. The definition has changed accordingly:

> A *charset* is a method of converting a sequence of octets into a sequence of characters. This conversion may also optionally produce additional control information such as directionality indicators.

(From RFC 2978) A conversion the other way may not exist, since different octet combinations may map to the same character. Another complicating factor is the possibility of switching between character sets; for instance, ISO2022-JP is the standard ASCII character set, but the character sequence `ESC $ @` switches to JIS X 0208-1978.

To disentangle the concepts behind encoding, we need to introduce a couple of levels:

ACR Abstract Character Repertoire: the set of characters to be encoded; for example, some alphabet or symbol set. This is an unordered set of characters, which can be fixed (the contents of ISO 8859-1), or open (the contents of Unicode).

CCS Coded Character Set: a mapping from an abstract character repertoire to a set of non-negative integers. This is what is meant by 'encoding', 'character set definition', or 'code page'; the integer assigned to a character is its 'code point'.

There used to be a drive towards unambiguous abstract character names across repertoires and encodings, but Unicode ended this, as it provides (or aims to provide) more or less a complete list of every character on earth.

CEF Character Encoding Form: a mapping from a set of non-negative integers that are elements of a CCS to a set of sequences of particular code units. A 'code unit' is an integer of a specific binary width, for instance 8 or 16 bits. A CEF then maps the code points of a coded character set into sequences of code points, and these sequences can be of different lengths inside one code page. For instance ASCII uses a single 7-bit unit; UTF-8 uses one to four 8-bit units. We will discuss the UTF encodings below.

CES Character Encoding Scheme: a reversible transformation from a set of sequences of code units (from one or more CEFs to a serialized sequence of bytes. In single-byte cases such as ASCII and UTF-8 this mapping is trivial. With the two-byte scheme UCS-2 there is a single 'byte order mark', after which the code units are trivially mapped to bytes. On the other hand, ISO 2022, which uses escape sequences to switch between different encodings, is a complicated CES.

Additionally, there are the concepts of

CM Character Map: a mapping from sequences of members of an abstract character repertoire to serialized sequences of bytes bridging all four levels in a single operation. These maps are what gets assigned MIBenum values by IANA; see section 4.1.

TES Transfer Encoding Syntax: a reversible transformation of encoded data. This data may or may not contain textual data. Examples of a TES are base64, uuencode, and quoted-printable, which all transform a byte stream to avoid certain values.

3 Unicode and UTF encodings

The systems above functioned quite well as long as you stuck to one language or writing system. Poor dictionary makers. More or less simultaneously two efforts started that aimed to incorporate all the world's character sets in one standard: the Unicode standard (originally 2-byte), and ISO 10646 (originally 4-byte). Unicode was extended further, so that it has all code points up to `10FFFF`, which is slightly over a million.

Two international standards organizations, the Unicode Consortium and ISO/IEC JTC1/SC2, started

designing a universal standard that was to be a superset of all existing character sets. These standards are now synchronized. Unicode has elements that are not in 10646, but they are compatible where it concerns straight character encoding.

ISO 10646 defines UCS, the 'Universal Character Set'. This is in essence a table of official names and code numbers for characters. Unicode adds to this rules for hyphenation, bi-directional writing, and more.

The full Unicode list of code points can be found online, broken down by blocks [14], and downloadable [17].

3.1 BMP and other Unicode subplanes

Characters in Unicode are mostly denoted hexadecimally as `U+wxyz`; for instance, `U+0041` is 'Latin Capital Letter A'. The range `U+0000`–`U+007F` (0–127) is identical to US-ASCII (ISO 646 IRV), and `U+0000`–`U+00FF` (0–255) is identical to Latin 1 (ISO 8859-1).

The original 2-byte subset is now called the 'BMP' for Basic Multilingual Plane, or plane 0. These are the Unicode code points that are nonzero in the last two bytes. Other 'planes' have been defined that have one or more bits set outside the last two bytes.

BMP (Basic Multilingual Plane) The first plane defined in Unicode/ISO10646, designed to include all scripts in active modern use. The BMP currently includes the Latin, Greek, Cyrillic, Devangari, hiragana, katakana, and Cherokee scripts, among others, and a large body of mathematical, APL-related, and other miscellaneous characters. Most of the Han ideographs in current use are present in the BMP, but due to the large number of ideographs, many were placed in the Supplementary Ideographic Plane.

SMP (Supplementary Multilingual Plane; plane 1) This contains mostly ancient writing systems. Some of these you'll have likely heard of, such as Linear B, cuneiform, Aztec, and Mayan; others are fairly obscure, such as Tangut, a language used in Central China between 1000 and 1500.

SIP (Supplementary Ideographic Plane) The third plane (plane 2) defined in Unicode/ISO 10646, designed to hold all the ideographs descended from Chinese writing (mainly found in Vietnamese, Korean, Japanese and Chinese) that aren't found in the Basic Multilingual Plane. The BMP was supposed to hold all ideographs in modern use; unfortunately, many Chinese dialects (like Cantonese and Hong Kong Chinese) were overlooked; to write these, characters from the SIP are necessary. This is one reason even non-academic software must support characters outside the BMP.

3.2 Unicode encodings

Unicode is basically a numbered list of characters. When they are used in a file, their numbers can be encoded in a number of ways. To name the obvious example: if only the first 128 positions are used, the long Unicode code point can be truncated to just one byte. Here are a few encodings:

UTF-32 Little used: this is a four-byte encoding. (UTF stands for 'UCS Transformation Format'.)

UTF-16 A two-byte encoding. Its precursor, UCS-2, encoded the BMP; UTF-16 has a way of going beyond that to encode planes 1–16 by using 'surrogate pairs' of two-byte units.

UTF-8 A one-byte scheme; details below.

UTF-7 Another one-byte scheme, but now the high bit is always off. Certain byte values act as an 'escape', so that higher values can be encoded. Like UTF-1 and SCSU, this encoding is only of historical interest.

There is an important practical reason for a one-byte encoding such as UTF-8. Multi-byte encodings such as UCS-2 are wasteful of space, if only traditional ASCII is needed. Furthermore, they would break software that is expecting to walk through a file with `s++` and such. Also, they would introduce many zero bytes in a file, which would play havoc with Unix software that uses null-termination for strings.

Then there would be the problem of whether two bytes are stored in low-endian or high-endian order. For this reason it was suggested to store `FE FF` or `FF FE` at the beginning of each file as the 'Unicode Byte Order Mark'. Formally, `FEFF` is the Unicode 'zero width nobreak space' character, which can innocently be inserted anywhere. Conversely `FFEF` is defined to be illegal, so encountering this is a sign that bytes should be interpreted little-endian. Of course this plays havoc with files such as shell scripts which expect to find `#!` at the beginning of the file.

3.3 UTF-8

UTF-8, standardized as RFC 3629, is an encoding where the positions up to 127 are encoded 'as such'; higher numbers are encoded in groups of 2 to 6 bytes. (Tim Bray describes this as 'kind of racist' [3]: the further east a language comes from, the more overhead is involved in its encoding.) In a multi-byte group, the first byte is in the range 0xC0–0xFD (192–252). The next up to 5 bytes are in the range 0x80–0xBF (128–191, bit pattern starting with `10`).

U-00000000 - U-0000007F	7 bits	0xxxxxxx		
U-00000080 - U-000007FF	$11 = 5 + 6$	110xxxxx	10xxxxxx	
U-00000800 - U-0000FFFF	$16 = 4 + 2 \times 6$	1110xxxx	10xxxxxx	10xxxxxx
U-00010000 - U-001FFFFF	$21 = 3 + 3 \times 6$	11110xxx	10xxxxxx	(3 times)
U-00200000 - U-03FFFFFF	$26 = 2 + 4 \times 6$	111110xx	10xxxxxx	(4 times)
U-04000000 - U-7FFFFFFF	$31 = 1 + 5 \times 6$	1111110x	10xxxxxx	(5 times)

Table 2: UTF-8 encoding blocks

Note that `8` = `1000` and `B` = `1011`, so the highest two bits are always `10`, leaving six bits for encoding). All bytes in a multi-byte sequence have their high bit set. See table 2.

IETF documents such as RFC 2277 require support for this encoding in internet software. Readable introductions can be found all over the Internet [19]; see also the history of UTF-8 in [20].

3.4 Unicode tidbits

3.4.1 Line breaking

The Unicode standard describes line breaking: it has a mechanism for specifying tables of character pairs between which line breaks are allowed or forbidden [15, 18].

3.4.2 Bi-directional writing

Most scripts are left-to-right, but Arabic and Hebrew run right-to-left. Characters in a file are stored in 'logical order', and usually it is clear in which direction to render them, even if they are used mixed. Letters have a 'strong' directionality: unless overridden, they will be displayed in their natural direction. The first letter of a paragraph with strong direction determines the main direction of that paragraph [16]. See figure 1.

However, when differently directional texts are embedded, some explicit help is needed. The problem arises with letters that have only weak directionality. The following is a sketch of a problematic case:

Memory: he said "I NEED WATER!", and expired.
Display: he said "RETAW DEEN I!", and expired.

If the exclamation mark is to be part of the Arabic quotation, then the user can select the text 'I NEED WATER!' and explicitly mark it as embedded Arabic (`<RLE>` is Right-Left Embedding; `<PDF>` Pop Directional Format), which produces the following result:

> Memory: he said "`<RLE>`I NEED WATER!`<PDF>`", and expired.
> Display: he said "!RETAW DEEN I", and expired.

A simpler method of doing this is to place a Right-To-Left Mark `<RLM>` after the exclamation mark. Since the exclamation mark is now not on a directional boundary, this produces the correct result.

> Memory: he said "I NEED WATER!`<RLM>`", and expired.
> Display: he said "!RETAW DEEN I", and expired.

3.5 Unicode and oriental languages

'Han unification' is the Unicode strategy of saving space in the oriental languages (traditional Chinese, simplified Chinese, Japanese, Korean: 'CJK') by recognizing common characters. This idea is not uncontroversial [6].

4 Further tidbits

4.1 A bootstrapping problem

In order to know how to interpret a file, you need to know what character set it uses. This problem also occurs in MIME mail encoding (section 4.5), which can use many character sets. Names and numbers

Figure 1: Right-to-left Arabic text containing left-to-right numerals

for character sets are standardized by IANA: the Internet Assigned Numbers Authority [9]. However, in what character set do you write this name down?

Fortunately, everyone agrees on (7-bit) ASCII, so that is what is used. A name can be up to 40 characters from US-ASCII.

As an example, here is the IANA definition of ASCII:

name	`ANSI_X3.4-1968`
reference	RFC 1345, KXS2
MIBenum	3
source	ECMA registry
aliases	`iso-ir-6`, `ANSI_X3.4-1986`, `ISO_646.irv:1991`, `ASCII`, `ISO646-US`, `US-ASCII` (preferred MIME name), `us`, `IBM367`, `cp367`, `csASCII`

The `MIBenum` (Management Information Base) is a number assigned by IANA.[1] The full list of character sets can be found online [8], and RFC 3808 is a memo that describes the IANA Charset MIB.

4.2 Unicode in programming languages

Before Unicode, a system called the 'Double Byte Character Set' was invented to accommodate Asian languages, where some characters were stored in one, others in two bytes. This is very messy, since you can not simply write `s++` or `s--` to traverse a string. Instead you have to use functions from some library that understands these encodings. While this system is now only of historical interest, the string handling problem is back in force with UTF-8.

Many modern languages (Python, C99) have support for Unicode. In C99 (which is the new standard for C) this is done through so-called 'wide characters'. For instance, `L'x'` is a wide character and `L"xyz"` is a string of wide characters. Such strings can be manipulated through equivalents of the normal string library. For instance, `wcscpy` acts like `strcpy` but on wide strings. General Unicode characters can be represented as `\u0000` for 4-byte and `\U00000000` for up to 8-byte characters.

The two-byte UTF-16 encoding is popular with programmers, since it can handle almost any practically encountered character without extensions to longer byte sequences.

4.3 Character codes in HTML

HTML can access unusual characters in several ways:

- With a decimal numerical code: `&#32;` is a space token. (HTML 4 supports hexadecimal codes.)
- With a vaguely symbolic name [12, 7]: `&copy;` is the copyright symbol, ` ` is a non-breaking space, etc.
- The more interesting way is to use an encoding such as UTF-8 (section 3.2) for the file. For this it would be nice if the server could state that the file is

  ```
  Content-type: text/html;charset=utf-8
  ```

  but it is also all right if the file starts with (end with `/>` for XHTML):

  ```
  <meta http-equiv="Content-Type"
        content="text/html;charset=utf-8">
  ```

It is a requirement of user agents that they can at least parse the `charset` parameter, which means they have to understand US-ASCII.

Open this link in your browser, and additionally view the source: `http://www.unicode.org/unicode/iuc10/x-utf8.html`. How well does your software deal with it?

4.4 Keyboards and control characters

Unprintable ASCII codes are accessible through the control modifier key; for this reason they are also called 'control codes' or control characters. The control key, combined with a regular key, zeros bits 2 and 3 of the ASCII code of that key. For instance, you can hit `Ctrl-[` to get `Esc`.

The way key presses generate characters is typically controlled in software. This mapping from keyboard scan codes to 7 or 8-bit characters is called a 'keyboard', and can be changed dynamically in most operating systems.

Using the modifier keys, one can generate more keystrokes than can be described in 8 bits, so keyboards can send an 'escape sequence': one escape character followed by one or more regular characters. The escape character is mostly ASCII NUL or ESC [10].

4.5 Characters in email

The protocols for Internet mail are based on '7-bit ASCII', that is, the high bit in every byte transmitted is supposed to be off. This is a problem for any message that has text outside of ASCII, such as when accented characters from the various ISO 8859 character sets are used. It also makes transmitting binary data such as images impossible. For this reason the 'Multipurpose Internet Mail Extensions' (MIME) were designed. MIME uses several encoding schemes, such as base64 or quoted-printable, to turn arbitrary data into 7-bit ASCII.

The email standard RFC 822 states that anything outside 7-bit ASCII has to be encoded with

[1] Apparently these numbers derive from the Printer MIB, RFC 1759.

uuencode. This means that the sender and recipient need decoding program; it is decidedly overkill if a message is plain ASCII apart from a few accented characters.

The MIME protocol (RFC 2045 and 2046) inserts headers in a mail message, stating for each message section the content type and the encoding that is used for that section of the message. These encodings are also used for attachments, in which case the content type should give an indication what application can handle the attachment after its decoding. 'Helpful' mail programs that automatically invoke such applications have been a source of Trojans (malicious software) in the past.

4.6 FTP

FTP is a very old ARPA protocol for transferring files from one computer to another. It knows 'binary' and 'text' mode: in binary mode bytes are transferred without further interpretation, but the text mode is concerned with files that contain lines of text. Unfortunately, line ends are different between operating systems, and their transfer in text mode is not well defined. Some FTP programs adjust line ends; others, such as Fetch on the Mac, actually do code page translation.

5 Character issues in (LA)TEX

5.1 Diacritics

Before 1990, TEX was a 7-bit system: only characters 0–127 in the input could be recognized, and fonts were also limited to 127 positions. This meant that there was not enough space in fonts for letters with accents, so accents (diacritics) were implemented as things to put on top of characters, even when, as with the cedilla, they are under the letter. This leads to the problem that TEX could not hyphenate a word with accents, since the accent introduces a space in the word (technically: an explicit kern).

Both problems were remedied to a large extent with the 'Cork font encoding', which contains most common accented letters as single characters. This means that accents are correctly placed by design, and also that the word can be hyphenated, since the kern has disappeared.

These fonts with accented characters became possible when TEX version 3 came out around 1990. This introduced full 8-bit compatibility, both on the input side and in the font addressing.

5.2 LATEX input file access to fonts

If an input file for LATEX is allowed to contain all 8-bit octets, we get all the problems of compatibility that plague regular text files. This is solved by the package inputenc:

```
\usepackage[code]{inputenc}
```

where *code* is applemac, ansinew, utf8, or various other code pages.

This package makes all unprintable ASCII characters, plus the codes over 127, into active characters. The definitions are then dynamically set depending on the code page that is loaded.

5.3 LATEX output encoding

The inputenc package does not solve the whole problem of producing a certain font character from certain keyboard input. It only maps a byte value to the TEX command for producing a character. To map such commands to an actual code point in a font file, the TEX and LATEX formats contain lines such as

```
\chardef\i="10
```

declaring that the dotless-i is at position 16. However, this position is a convention, and other people — type manufacturers — may put it somewhere else.

This is handled by the 'font encoding' mechanism. The various people working on the LATEX font schemes have devised a number of standard font encodings. For instance, the OT1 encoding corresponds to the original 128-character set. The T1 encoding is a 256-character extension thereof, which includes most accented characters for Latin alphabet languages.

A font encoding is selected with

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

A font encoding definition contains lines such as

```
\DeclareTextSymbol{\AE}{OT1}{29}
\DeclareTextSymbol{\OE}{OT1}{30}
\DeclareTextSymbol{\O}{OT1}{31}
\DeclareTextSymbol{\ae}{OT1}{26}
\DeclareTextSymbol{\i}{OT1}{16}
```

5.4 TEX beyond 8 bits

The above LATEX packages allow flexible handling of (8-bit) code pages, essentially the ISO 8859 standard. For handling of other alphabets, a number of styles have been written over the years. However, their continued support is often uncertain. The first project that aimed at use of Unicode throughout TEX's code base was Omega [13]; the modern TEX extensions XETEX (http://scripts.sil.org/xetex) and LuaTEX (http://luatex.org) also do so.

References

[1] Annotated history of ASCII. http://www.wps.com/projects/codes/index.html.

[2] Bob Bemer homepage. http://www.trailing-edge.com/~bobbemer/.

[3] Tim Bray. Characters vs. bytes. http://www.tbray.org/ongoing/When/200x/2003/04/26/UTF.

[4] Brief history of character codes in North America, Europe, and East Asia. http://tronweb.super-nova.co.jp/characcodehist.html.

[5] Codepage & co. http://aspell.net/charsets/codepages.html.

[6] Han unification. http://en.wikipedia.org/wiki/Han_unification.

[7] Character entity references in HTML4. http://www.w3.org/TR/html401/sgml/entities.html.

[8] IANA character set names. http://www.iana.org/assignments/character-sets.

[9] Internet Assigned Numbers Authority. http://www.iana.org/.

[10] IBM PC keyboard scan codes. http://jimprice.com/jim-asc.shtml#keycodes.

[11] The ISO Latin 1 character repertoire. http://www.cs.tut.fi/~jkorpela/latin1/index.html.

[12] Character entities for ISO Latin 1. http://www.cs.tut.fi/~jkorpela/HTML3.2/latin1.html.

[13] Omega project home page. http://omega.enstb.org/.

[14] Unicode. http://www.fileformat.info/info/unicode/index.htm.

[15] Unicode standard annex 14, line breaking properties. http://www.unicode.org/reports/tr14/.

[16] Unicode standard annex 9, the bidirectional algorithm. http://www.unicode.org/reports/tr9/.

[17] Unicode code chart and scripts. http://www.unicode.org/charts/.

[18] Unicode line breaking rules: explanations and criticism. http://www.cs.tut.fi/~jkorpela/unicode/linebr.html.

[19] UTF-8 and Unicode FAQ for Unix/Linux. http://www.cl.cam.ac.uk/~mgk25/unicode.html.

[20] UTF-8 history. http://www.cl.cam.ac.uk/~mgk25/ucs/utf-8-history.txt.

◇ Victor Eijkhout
University of Texas at Austin
victor (at) eijkhout dot net

Fonts

lxfonts: LaTeX slide fonts revived

Claudio Beccari

Abstract

The LaTeX 2ε class `slides` used special fonts whose readability was exceptional. However, despite being part of the TeX system, they were not particularly efficient where mathematics was concerned. Since the time the LaTeX3 Team abandoned `slides`, they have almost disappeared.

This article describes the modifications and enhancements made to revive these historical fonts, and explains the package making the new font version usable with modern presentation classes, in order to produce slides that were unthinkable during the olden times of `slides`.

1 Some history

Once upon a time there was a program, SliTeX [4], when the TeX interpreter's version was 2.x. This was the 1980s, almost the prehistory of TeX, certainly LaTeX's infancy, when in order to typeset a document in a language different from English it was necessary to have suitable format files, since that old TeX could handle only one hyphenation pattern set at a time.

With version 3.0 of TeX, the ability to typeset in a variety of languages different from English spread the TeX system, with its LaTeX dialect, in the old continent, and it became so important that the Europeans set up the LaTeX3 team. This new team produced, in 1994, the new LaTeX 2ε, and with this SliTeX died for good, replaced by the standard class `slides` [5].

Actually, other than using the same interpreter and the same LaTeX set of macros, with the `slides` class things did not change very much. In the short term, in 1999 the LaTeX3 team gave up the maintenance of the `slides` class; even if it still is a standard LaTeX class distributed with every new release of the TeX system, the Team formally invited the users of TeX to develop new classes to produce presentations and slide shows. This warm invitation included the strong suggestion that such new creations should be made available to the whole TeX user community.

For these reasons, in the past several years we have witnessed the proliferation of many systems, packages and classes to produce excellent presentations, fully colored, with some animations, making use of a large variety of outline fonts, particularly well suited for use with modern video beamers.

In this article I describe the old fonts used by SliTeX and by the `slides` class, with their pros and cons. I further describe the modifications I have introduced, and in particular how I produced their PostScript versions, necessary in order to be used with the modern classes and packages. I describe the `.sty` file with which a complete substitution of the CM/EC (Computer Modern/European Modern) or LM (Latin Modern) families, together with the American Mathematical Society's fonts may be made in order to produce nice presentations that use such fonts and exploit their legibility. What I produced may be considered an alpha-release; in order to become a beta or a definitive release, some feedback is required so as to repair the various glitches that may possibly still be present (and certainly there are many...).

2 The quotation fonts

The SliTeX fonts derive from the ones that Knuth created for typesetting those witty quotations at the end of every chapter of *The TeXbook* [2] and *The METAFONTbook* [3].

They are sans serif fonts, formally designed at a design size of 8 pt, but with a large x-height and short ascenders and descenders. Anybody who handled those books knows perfectly what we are talking about, but here a specimen is replicated in order to describe the successive modifications.

> *If you can't solve a problem,*
> *you can always look up the answer.*
> *But please, try first to solve it by yourself;*
> *then you'll learn more and you'll learn faster.*
> — DONALD E. KNUTH, *The TeXbook* (1983)

The elegance and style of this sans serif font is immediately evident, simple and perfectly readable. Although it is used at size 8 pt, its x-height appears as large as that of the roman font of the main text.

Its drawback, though, is that the lower case 'l' and the upper case 'I' and, in math mode (not shown), the 'absolute value' sign are indistinguishable. This appealing font, well-suited for Knuth's quotations, is thus not suited for typesetting mathematics.

3 The slide fonts

This is why since the beginning of SliTeX, Leslie Lamport created a new font for slides by modifying

Editor's note: This is a translation of the article "I font per le slide LaTeX resuscitati", which first appeared in *ArsTeXnica* issue #4 (October 2007), pp. 82–87. Reprinted with permission. Translation by the author.

the quotation fonts, essentially by substituting the upper case 'i' with a seriffed variant.

If you can't solve a problem,
you can always look up the answer.
But please, try first to solve it by yourself;
then you'll learn more and you'll learn faster.
— DONALD E. KNUTH, *The TEXbook* (1983)

4 The new fonts for slides

As is clear from the previous example, this modified capital 'I' solves part of the problem; there remains the fact that the lower case 'l' may still be confused with the mathematical sign for marking the absolute value, or the norm, or this sort of mathematical entity.

For this reason, the new fonts under discussion here have been created with the lowercase 'l' drawn with a hook at the bottom, similar to the foot of the lower case 't'. The result is the following.

If you can't solve a problem,
you can always look up the answer.
But please, try first to solve it by yourself;
then you'll learn more and you'll learn faster.
— DONALD E. KNUTH, *The TEXbook* (1983)

Of course this also implies the modification of the metric files `.tfm`, and, more intrusive than anything else, the correction of all the ligatures and kerns where the lower case 'l' is involved. Having designed these fonts for both the old OT1 and the T1 Latin encodings, this implied also the Polish ł with its kerns, and all the other signs where some sort of diacritic mark is used with the base sign 'l'. The revisions involved both the normal and the slanted styles, as well as both the medium and the bold weights. For completeness, the same design parameters were used for the Text Companion font in encoding TS1 in all its versions, so that the new fonts could be paired harmoniously.

5 Typesetting problems in mathematics

When typesetting mathematics new problems arise; in the old SliTeX math expressions were typeset with the normal CM math fonts; but this produced undesirable effects even beyond the obvious stylistic clash, because the operators font was substituted by the upright version of the slide fonts, but all the other signs were taken from the letters, symbols, and large delimiters CM fonts. This produced a bad rendering of those signs that were produced by kerning a symbol from the operators font and another symbol from the symbols font. For example, the double long arrows are produced by joining an 'equals sign' from the operators font and a 'double arrow tip' from the symbols font. It seems complicated, but it suffices to observe the `\Longrightarrow` sign $\Longrightarrow$ to understand the problem; the mathematical axes do not match and the stroke thicknesses are different; the final result is very unsatisfying.

For math mode typesetting it was thus necessary to create math fonts that had the same design parameters as the new slide fonts. This implied the creation of the 'letters' font (essentially the math italics, the Greek upper and lower case letters and a good number of other symbols), the symbols font (most operator symbols, the old style numbers and the calligraphic letters), and the large delimiters and display math operators font. This further work did not consist simply of changing the design parameters in the master METAFONT file, but also reviewing each glyph in order to assure its adequacy to the typesetting of (hopefully) beautiful mathematics, as good as the CM math fonts do in normal text. Those unsatisfying compound math signs, so frequent in the definitions of math symbols, had to be checked one by one, in order to assure the perfect match of the new glyphs with one another.

6 Examples

Besides the last text example in section 4, I now show some examples where different series and typesetting modes are mixed.

6.1 The medium and the bold series

The preceding running title is typeset with the **bold** series, while this text is composed with the **medium** one.

Notice that the previous example, unlike the one given in section 4, is typeset at size 10 pt, as is this normal text (which is typeset with the standard roman font of the CM collection). The larger x-height induces one to think that the example is typeset with a larger font, possibly size 12 pt.

6.2 The font sizes

The new font, like the old one, is designed at just one size, and other sizes are obtained by shrinking or enlarging this single size. Table 1 shows some sizes and it's evident that the smaller sizes are definitely too thin, while the larger ones do not appear blacker as the normal CM fonts do, with their multiple design sizes.

But this is a deficiency that was already inherent in the old slide font, and it also manifests itself with the vast majority of the available Type 1 or Open Type fonts when they are excessively shrunk or magnified.

Size	Example
5 pt	ABCD abcd
7 pt	ABCD abcd
10 pt	ABCD abcd
12 pt	ABCD abcd
14 pt	ABCD abcd
17 pt	ABCD abcd
20 pt	ABCD abcd

Table 1: The new font in different sizes

As things are, the first level sub- and superscripts are certainly readable, while the second level ones may be a bit too thin. Nevertheless, some of the examples shown below contain such second-order subscripts, and one hardly notices they are so thin.

6.3 Comparison with the standard sans serif CM font

It's interesting to compare the normal sans serif font selectable with the command `\textsf`, with the new font drawn at the same optical size — see table 2. It's evident that the new font, in spite of being at the same size, appears definitely larger than the normal sans serif one.

OT1/cmss	abcdefghijklmnopqrstuvwxyz
OT1/llcmss	abcdefghijklmnopqrstuvwxyz

Table 2: The regular CM sans serif font compared with the slide font

6.4 Some mathematics

It's worth typesetting some simple math expression as if we were preparing a slide: see figure 1. Probably at first sight what strikes our attention is that the exponents are a little lighter than expected, since we are used to math expressions typeset with the CM math fonts, where exponents are drawn from the right optical size, not merely reduced versions of a larger size. But this problem, as was already pointed out, occurs with almost all outline fonts, except those specific to the TEX system collections where optical sizes are conveniently available, especially for purposes of typesetting math.

Another point is that math italics are real italics, not a slanted version of the upright font, as happens with the default settings for some classes for slide production and conference presentations. This happens with both CM sans serif fonts and with Helvetica, the two most common sans serif fonts for preparing

The second degree equation with constant real coefficients:

$$ax^2 + bx + c = 0 \tag{1}$$

has solutions

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{2}$$

with

$$\begin{cases} x_{1,2} \in \mathbb{R} & \text{se } b^2 - 4ac > 0 \\ x_1 = x_2 = -\dfrac{b}{2a} & \text{se } b^2 - 4ac = 0 \\ x_{1,2} \in \mathbb{C} & \text{se } b^2 - 4ac < 0 \end{cases} \tag{3}$$

Figure 1: Some text and math

presentations.

Another point is evident: figure 1 contains symbols, such as the blackboard bold capital letters, that belong to the AMS math font collections. Yes, the new slide font collection contains also the sans serif version of the AMS fonts, redrawn with the same graphic parameters used for the other fonts of this collection.

The equations shown in figure 1 appear out of their typical context of a slide show, since they are intermixed with the rest of this article's text, typeset with the normal Computer Modern roman font. It is better to examine a whole slide show displayed with an actual projector in order to appreciate the beautiful effect of these fonts maintaining the legibility associated with the old SLiTEX. To this end, the new fonts' distribution package contains a demo beamer show contained in the file `Slidesfonts-demo.pdf` where everything is typeset with these new fonts and where some more information is given in addition to what is provided here.

6.5 Some more math

Of course it is impossible to display here all the available LATEX math symbols, AMS math included. Nevertheless, in order to evaluate what you can get with these new fonts when typesetting math, it's worthwhile to typeset some more expressions, for example an expression where large operators appear; see figure 2.

Notice that the expression in figure 2 contains also the counterclockwise oriented circle, taken from the redrawn AMS math font collection.

Another formula, figure 3, contains a triple integral, a typical construct obtainable with standard amsmath commands, but here they directly access the newly redrawn math symbols and operators. The

The residues theorem states that if $f(z)$: $z, f \in \mathbb{C}$ is analytic in $\mathbb{D}$ except in a finite set of singularities, then it is

$$\oint_\gamma f(z)\,\mathrm{d}z = 2\pi\mathrm{j}\sum_{k=1}^{N_{\mathsf{sing}}} R_k$$

where γ is a simply connected line totally lying in $\mathbb{D}$ and N_{sing} is the number of singularities contained within γ.

Figure 2: Some more math

$$\iiint_V F(\boldsymbol{P})\,\mathrm{d}x\,\mathrm{d}y\,\mathrm{d}z$$

Figure 3: Large operators

mixture of medium and bold math symbols is obtained with the `\boldsymbol` command, but all the symbols are taken from the new fonts.

More examples could be produced endlessly — all constructs that can be typeset with LaTeX and its extension packages are compatible with the new fonts.

7 Text symbols

Since the Text Companion font [7] has also been redrawn with the graphic parameters of these fonts, all symbols available with that Companion font are accessible together with these slide fonts; some glyphs are shown in figure 4.

£ µ Ω ¥ $ € ℃

Figure 4: A specimen of Text Companion symbols

Notice in particular the Euro symbol: in contrast to the original Text Companion font, this glyph is without serifs, according to the style of these new slide fonts.

Obviously this font variety has the same shapes and series of the main text font, that is upright medium and bold, and slanted medium and bold; the italic variant is missing as is customary with sans serif fonts. In any case, with symbols the difference between slanted and italics is rather uncertain.

8 The PostScript Type 1 implementation

All fonts described so far have been transformed into PostScript Type 1 format by means of mftrace [8]. This script provides for calling a number of other programs in order to trace the contours of bitmapped fonts produced with METAFONT, clean up the results, and assemble the final `.pfb` font files.

In order to complete this transformation it is necessary to use particular encoding vectors; for the American Mathematical Society fonts such encoding vectors were not distributed with TeX Live in 2007, so it was necessary to create them. They are included in this font distribution package in the enc subdirectory; although they should not be necessary in normal use, now they are available.

9 Font installation

In order to use these fonts, it is necessary to install them. If the bitmapped versions are sufficient, then the `.mf` and the `.tfm` file should be copied to the right folders and the file name database should be updated.

Since the outline versions are available, there is no reason to avoid their installation. It is a little trickier than with the ordinary METAFONT version, but the instructions are given in the documentation contained in the distribution package. See also [6].

In fact, installation of the outline fonts is highly recommended. Even if PDF viewers are improving, it's still true that bitmapped fonts are not particularly suited for reading a document at various magnifications: enlarged bitmapped fonts appear as small tiles set close together, instead of fonts with smooth contours. Moreover, bitmapped fonts should be avoided when one uses programs, such as pdflatex, that directly use outline fonts, for the very reason described above. Presentations, which these fonts were intended for, are of course typically produced in PDF format.

10 The extension package for the fonts

Last and perhaps most important of all, it is necessary to copy `lxfonts.sty` into a suitable folder where LaTeX can find it.

This file, `lxfonts.sty`, is the keystone of the whole building, but it must be used with care. It must be invoked by means of `\usepackage` *after* all other font related packages have been loaded.

Some details:

- If no special font choices are made, the default settings of `lxfonts.sty` is to choose the `OT1` font encoding (not recommended when typesetting in languages that use many diacritical marks), and neither the Text Companion nor the AMS fonts are loaded.
- If the `T1` encoding is specified, then the package loads the `T1` encoded fonts, which is recommended when typesetting in most Latin-based languages other than English.
- If the Text Companion font was invoked, then `lxfonts.sty` loads the new substitution font,

in the sense that all normal Text Companion commands that access glyphs from the original font, after this substitution, access the glyphs from the new font.

- Finally, if the amsmath package was called for, that is, if the AMS math symbols are required, then the `lxfonts.sty` file takes care of substituting the original AMS fonts by the new ones.

There is no declaration to make in order to use these new fonts; everything is needed is provided by the new `lxfonts.sty` file. The curious ones who want to dig into this file will find only family and shape definitions that refer to just one size, 8 pt, that is magnified or shrunk according to necessity during the various typesetting stages. For this reason it is particularly convenient to use the scalable outline fonts; their installation might be a little tricky, but it is worth the effort.

11 Suggestions

The simple examples shown in the previous sections of this printed paper demonstrate the pros and cons of these new fonts. In particular, they display the extreme lightness of the smaller sizes, and this, perhaps, is the primary drawback of these fonts.

But when you use them for producing presentations, for example with the beamer class, you get the best out of them, because they appear to be particularly suited for presentations; after all, they were originally conceived with this aim as part of the old SliTeX system. And it is for this very reason that is suggested to avoid using these fonts for anything but presentations and, perhaps, for text to be printed at very large sizes.

12 Conclusion

While building these fonts I had to correct a certain number of glitches in the original METAFONT files originally produced by the American Mathematical Society. These glitches probably never showed up because nobody (I suppose) needed to produce new fonts with different graphical parameters while using the same METAFONT programs. By experience I know that more often than not the character programs are tweaked to the necessities of the graphic parameters, and probably I did the same with the AMS fonts. I do not want to blame at all the AMS experts who designed the AMS fonts, because their work is excellent and their fonts have been used for decades now, to the complete satisfaction of every user. I simply noticed that their METAFONT programming was specific for the particular glyphs that had to be produced.

The same holds true for the Text Companion fonts, where I had to 'correct' only the Euro sign — the original serifed one gave very strange results with the new parameters.

I am sure that many other corrections are necessary, but being the only user of these new fonts, I have not been able to discover more. Therefore the lxfonts package now available on CTAN [1] must be considered an alpha version, though reliably usable.

Therefore I invite all interested readers to use these new fonts, discover where they should be corrected and give me feedback on their findings. As with every piece of free and open source software, these fonts get better when constructive criticism is provided by the users, not to mention the software contribution that competent users can offer.

References

[1] Beccari, Claudio. The lxfonts package. `http://ctan.org/get/fonts/lxfonts`.

[2] Knuth, Donald E. *The TeXbook*. Addison Wesley, Reading Mass., 1990.

[3] Knuth, Donald E. *The METAFONTbook*. Addison Wesley, Reading Mass., 2000.

[4] Lamport, Leslie. *LaTeX: A Document Preparation System*. Addison Wesley, Reading Mass., 1st edition, 1984.

[5] Lamport, Leslie. *LaTeX: A Document Preparation System*. Addison Wesley, Reading Mass., 2nd edition, 1994.

[6] Philipp Lehman. The font installation guide, 2004. `http://ctan.org/get/info/Type1fonts`.

[7] Frank Mittelbach, Michel Goossens, et al. *The LaTeX Companion*. Addison Wesley, 2nd edition, 2004.

[8] Nienhuys, Han-Wen. mftrace — scalable fonts for METAFONT. `http://lilypond.org/download/sources/mftrace/mftrace-1.2.14.tar.gz`.

◇ Claudio Beccari
`claudio dot beccari at gmail dot com`

Reshaping Euler: A collaboration with Hermann Zapf

Hans Hagen, Taco Hoekwater, Volker RW Schaa

It is no secret that over the last few years Hermann Zapf has been reshaping some of his designs, most notably Palatino and Optima. Some three years ago, when Volker and Hans were talking to Hermann, they discovered he would like to improve the Euler fonts as well. These fonts were developed a few decades ago using the technology of those days, in close cooperation between Don Knuth and Hermann Zapf.

The glyphs were drawn on paper about 6cm height and these drawings were digitized using pinpoints on paper with a raster. The resulting points were translated to METAFONT and some additional math shapes were added afterwards. Later, when the fonts became popular with TEXies, virtual fonts were created using Euler and AMS Math fonts.

The resulting bitmap fonts were fine for the bitmap-oriented TEX backends of those days. Later, when bitmaps became outdated, the bitmaps became outlines, and the artifacts introduced in the digitization became somewhat more prominent (especially when the fonts were scaled).

The reasons why Hermann wanted to reshape Euler were manifold. First, he wanted to improve some details related to drawing with a broad pen. Then, the slope as well as the descenders of some glyphs needed to be adapted. The strokes had to be made more consistent too. Finally, the characters that were not Euler (but had been added afterwards) had to be redrawn: first the core characters, later (in principle) all characters that TEXies use. This last effort is still on the agenda and part of making Euler Unicode compliant.

When we met Hermann on a subsequent occasion, the topic of reshaping Euler came up again, and we decided to go ahead with an active project. Taco was willing to join in and we decided to improve the fonts by just editing the Type 1 fonts.

Because the project would take more than a year (at that time Hermann was still working at Linotype on his other projects), we decided to make this redesign into a present for Don Knuth's 70^{th} birthday. At that point the old Euler was 25 years old.

The following graphics display some of the changes. Some are more prominent than others. Even small corrections help improve the overall look and feel because they influence our perception of black on white. (It may help to have a magnifier at hand.)

In figure 1 we take a first look at some of the reshaping. The gray area is the bounding box, the white shape is the new font, the outline is the old one.

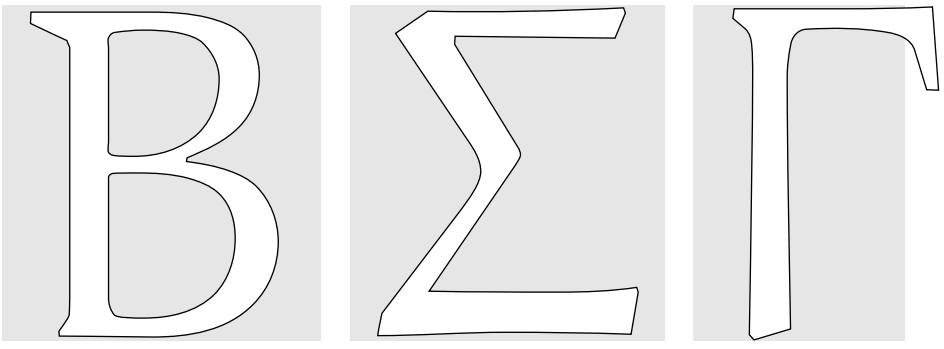


Figure 1 New Euler Roman Medium (a)

In figure 2 we see more drastic changes: shortened strokes. The bounding box is kept unchanged since we made it an initial goal for the new shapes to work well with the existing metric files; that

way, New Euler would be a drop-in replacement for the existing fonts and could be used with no fear of changing line breaks.

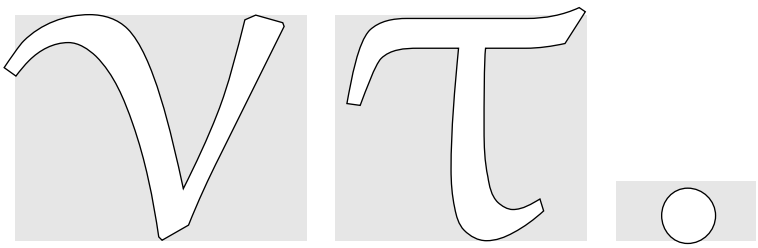


Figure 2 New Euler Roman Medium (b)

As with Palatino Nova and Optima Nova, Hermann did not hesitate to go even further than this. Figure 3 demonstrates this clearly. A nice side effect of harmonizing the font is that we can now use Euler for running text, although the text font is not yet released (due to too many holes in the usual text encoding vectors).

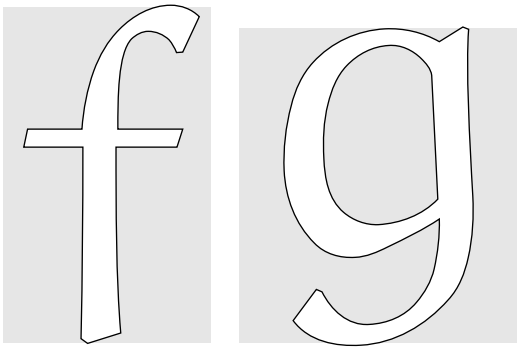
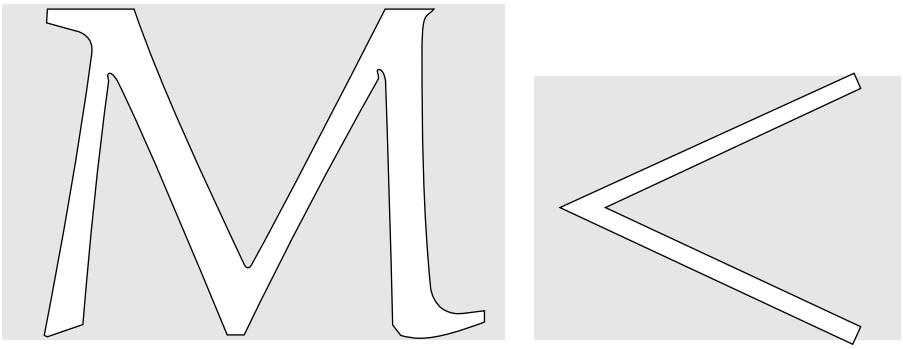


Figure 3 New Euler Roman Medium (c)

Some of the changes result directly from looking at the fonts in a larger size (see figure 4). The redesign started by printing the outlines of the fonts at sizes up to 12cm but finally Hermann decided to focus on the 6cm variant. The corrected outlines were mailed, faxed and/or presented in person. Many such corrections concerned the way corners are cut off. In that respect some of the original characters didn't qualify as Euler at all, for instance < symbols, but by cutting some corners and adapting the strokes they became eulerized.



bend in corners wrong cut-offs

Figure 4 New Euler Roman Medium (d)

When the discussions about reshaping started, the changes mostly concerned small corrections and descenders, but once we had the proper work-cycle in place Hermann went a bit further. Of course the descenders have been lowered too, as is demonstrated in Figure 5.

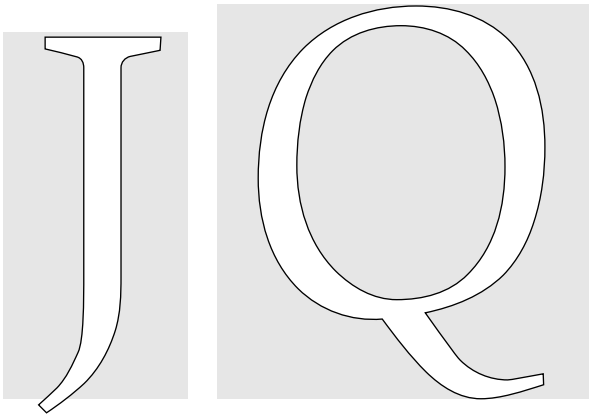
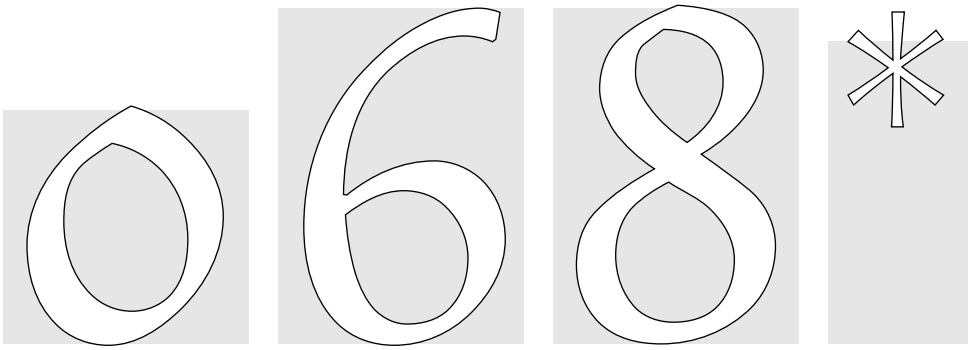


Figure 5 New Euler Roman Medium (e)

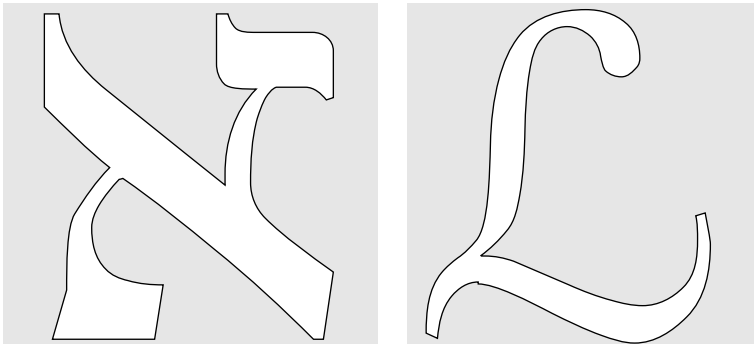
As usual, TEX math fonts have interesting ways of combining characters in fonts and so we have old style digits in the Fraktur font. The elegance of New Euler is well demonstrated by these numerals (see figure 6).



eufm: 0 eufm: 6 eufm: 8 eufm: star

Figure 6 New Euler Fraktur Medium

Most characters have been changed, some much more than others. In the symbol font, the aleph now better matches the rest (it was rather fat) and the script L is less upright (see figure 7).



eusm: Aleph eusm: Script L

Figure 7 New Euler Symbol Medium

As intended, New Euler is metric compatible with Old Euler, and of course the smaller sizes and the bold variants have been adapted too. By the end of 2007 all the medium variants at 10pt were done, and Taco had to go into overdrive. We were quite lucky that he has mastered FontForge so well (figure 8), and so we could start 2008 with a complete set of fonts.

normal bold

Figure 8 Editing Euler in FontForge

The fonts were presented to Don Knuth on January 10, 2008 on an eight-page leporello hand-made by Willy Egger, with each page showing one of the aspects of the reshaping, all of which kept us pretty busy during the holidays (figure 9).

Figure 9 Presenting New Euler

Does the project end here? No, this is just the first stage. Hermann is willing to participate in extending the Euler fonts with the Unicode math characters that make sense.

An important aspect of the project is to get the old fonts replaced by the new ones. We're happy that the AMS is considering acceptance of the new fonts as a formal substitute for the existing ones. And of course, the TEX distribution wizard Karl Berry will take care of getting the fonts in the right places and rooting out traces of old ones, when the time comes.

For quite some time Don Knuth has been asking users to get rid of the old Computer Modern delta, so in closing let's quote him from his web site on behalf of Hermann:

> If you see that your system produces the symbol δ instead of δ for the Greek lowercase delta, you should tell your system administrator immediately to upgrade your obsolete version of the Euler fonts.

And don't tell us that you don't see the difference. And, as you may expect, this quote was typeset in Euler Text.

◇ Hans Hagen, Taco Hoekwater, Volker RW Schaa

Software & Tools

Asymptote: A vector graphics language

John C. Bowman and Andy Hammerlindl

Abstract

Asymptote is a powerful descriptive vector graphics language inspired by METAPOST that features robust floating-point numerics, automatic picture sizing, native three-dimensional graphics, and a C++/Java-like syntax enhanced with high-order functions.

1 Motivation

The descriptive vector graphics language Asymptote[1] was developed to provide a standard for drawing mathematical figures, just as TEX and LATEX have become the standard for typesetting equations in the mathematics, physics, and computer science communities. Asymptote has been aptly described as "the ruler and compass of typesetting" [1]. For professional quality and portability, Asymptote natively produces PostScript or PDF output. Graphics labels are typeset directly by TEX to achieve overall document consistency: identical fonts and equations should be used in graphics and text portions of a document.

In this article we first highlight Asymptote's basic graphics capabilities with an example and then proceed to review the origins and distinguishing features of this powerful vector graphics language. Further examples of Asymptote diagrams, graphs, and animations are available in the Asymptote gallery and user-written wiki:
http://asymptote.sourceforge.net/gallery/
http://asymptote.sourceforge.net/links.html

2 An example

The following example illustrates the four Asymptote graphics primitives (draw, fill, clip, and label):

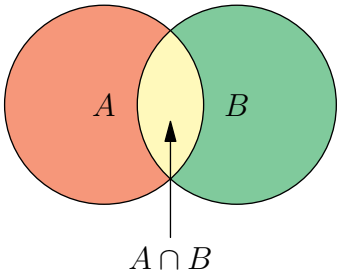
```
size(0,100);
pair z1=(-1,0);
pair z2=(1,0);
real r=1.5;
path c1=circle(z1,r);
path c2=circle(z2,r);

fill(c1,lightred);
fill(c2,lightgreen);

picture intersection;
fill(intersection,c1,lightred+lightgreen);
clip(intersection,c2);
add(intersection);

draw(c1);
draw(c2);

label("$A$",z1);
label("$B$",z2);
path g=(0,-2)--(0,-0.25);
draw(Label("$A\cap B$",0),g,Arrow);
```



3 History

Asymptote began as a University of Alberta summer undergraduate research project in 2002, after looking into the feasibility of overhauling Hobby's METAPOST[2] to use floating-point numerics. Many of the current limitations of METAPOST derive from METAFONT: numbers are stored in a low-precision fixed-point format that is adequate for representing points in a glyph but restrictive for diagrams and scientific computations. While the IEEE floating-point numeric format was not standardized until 1985, the initial development of TEX dates back to 1978. At that time, the decision to use only fixed-point (integer-based) arithmetic was perfectly reasonable: Knuth wanted to guarantee that TEX and METAFONT would produce exactly the same bit-mapped output on any existing hardware.

We quickly determined that a complete rewrite of the underlying graphics engine would be necessary. After six months of work, our compiler for a new graphics language could finally draw a sine curve. One of the four Asymptote primitives had now been implemented!

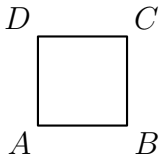
From this very humble beginning, Asymptote evolved rapidly. The basic fill operation was straightforward to implement. The most crucial advance,

[1] Andy Hammerlindl, John Bowman, and Tom Prince, available under the GNU General Public License from http://asymptote.sourceforge.net/

[2] METAPOST is a modified version of METAFONT, the program that Knuth wrote to produce the Computer Modern fonts used with TEX.

aligning TEX labels at the correct positions, was accomplished three months later, using compass directions or arbitrary angles to specify label alignments:

```
size(0,2cm);
draw((0,0)--(1,0)--(1,1)--(0,1)--cycle);
label("$A$",(0,0),SW);
label("$B$",(1,0),SE);
label("$C$",(1,1),NE);
label("$D$",(0,1),NW);
```



The idea was to leave the entire label typesetting to LATEX, inserting PostScript layers with `\includegraphics`, and thereby avoid the complications and kerning issues inherent in the METAPOST approach of post-processing the DVI file. To accomplish this, `Asymptote` communicates with TEX *via* a bidirectional pipe in two passes: the first pass is used to obtain label sizing information, while the second pass performs the final typesetting directly into DVI/PostScript or PDF. Label clipping and transforms are implemented with PostScript or PDF specials.

A third co-developer, Tom Prince, joined us in 2004. He contributed a method for embedding `Asymptote` code directly in LATEX source files. With Tom's help, we ported more reliable versions of the METAPOST algorithms for basic Bézier path operations such as splitting into subpaths, computing points of tangency, determining path bounds, and finding intersection points. Robust arc length and arc time computations were implemented with adaptive Simpson integration, which was determined to be more efficient than Bézier subdivision.

On November 7, 2004, we posted our first public release, version 0.51, on `sourceforge.net`. Since then, the user base and the list of new features have grown dramatically. The current version at the time of this writing is 1.42. Like METAPOST, `Asymptote` runs on GNU/Linux and other UNIX-like operating systems, Microsoft Windows, and Mac OS X. Precompiled `Asymptote` binaries are now included in several major Linux distributions.

4 Language features

`Asymptote` uses lexical analysis, parsing, and intermediate code generation to compile commands into virtual machine code, optimizing speed without sacrificing portability. Double-precision floating-point numbers and 64-bit integers make arithmetic overflow, underflow, and loss of precision issues much less troublesome than they are in METAPOST programs. `Asymptote` represents curves as cubic Bézier splines, but can easily handle large data values and the pathological behaviour of functions like $x\sin(1/x)$ near the origin. It also supports new path operations like computing the winding number of a path relative to a given point, which is useful for identifying the region bounded by a closed path.

Most users find the `Asymptote` language much easier to program in, with its C++/Java-like syntax (augmented to support high-order functions), than METAPOST, with its awkward and somewhat confusing `vardef` macros. `Asymptote` also borrows several ideas from `Python`, such as named function arguments and array slices. High-level graphics commands are implemented in the `Asymptote` language itself, allowing them to be easily tailored to specific user applications.

Like METAPOST, `Asymptote` is mathematically oriented. For example, one can rotate vectors by complex multiplication and apply affine transformations (shifts, rotations, reflections, and scalings) to pairs, triples, paths, pens, strings, pictures, and other transforms.

4.1 Functions

`Asymptote` is the only language we know of that treats functions as variables, but allows overloading by distinguishing variables based on their signatures. In fact, function definitions are just syntactic sugar for assigning function objects to variables:

```
real square(real x) {return x^2;}
```

is equivalent to

```
real square(real x);
square=new real(real x) {return x^2;};
```

`Asymptote` supports a more flexible mechanism for default function arguments than C++: they may appear anywhere in the function prototype. This feature underlies `Asymptote`'s greatest strength: sensible default values for the basic graphical elements allow beautiful graphs and drawings to be created with extremely short scripts, without sacrificing the flexibility for detailed customization. Default arguments are evaluated as `Asymptote` expressions in the scope where the function is defined.

Because certain data types are implicitly cast to more sophisticated types, one can often avoid ambiguities in function calls by ordering function arguments from the simplest to the most complicated. For example, given

```
real f(int a=1, real b=0) {return a+b;}
```

the call `f(1)` returns 1.0, but `f(1.0)` returns 2.0. It is sometimes difficult to remember the order in which arguments appear in a function declaration. Python-style named (keyword) arguments make calling functions with multiple arguments easier: the above examples could respectively be written `f(a=1)` and `f(b=1)`. An assignment of a function argument is interpreted as an assignment to a parameter of the same name in the function signature, not in the local scope of the calling routine.

Rest arguments allow one to write functions that take a variable number of arguments. For example, the following function sums its arguments:

```
real sum(... real[] nums) {
  real total=0;
  for(real x : nums)
    total += x;
  return total;
}
```

As in other modern languages, functions can call themselves recursively. Operators, including all of `Asymptote`'s built-in arithmetic and path operations, are just syntactic sugar for functions that can be addressed and defined with the `operator` keyword.

`Asymptote` functions are first-class values, allowing them to be defined within, passed to, and returned by other functions. This is convenient when one wants to graph a sequence of functions such as $f_n(x) = n\sin(x/n)$ for $n = 1$ to 5 from $x = -10$ to 10:

```
import graph;
typedef real function(real);

function f(int n) {
  real fn(real x) {
    return n*sin(x/n);
  }
  return fn;
}

for(int n=1; n <= 5; ++n)
  draw(graph(f(n),-10,10));
```

Anonymous functions can be created with the keyword `new`, so that the function definition in the previous example could be simplified to

```
function f(int n) {
  return new real(real x) {
    return n*sin(x/n);
  };
}
```

5 Modules

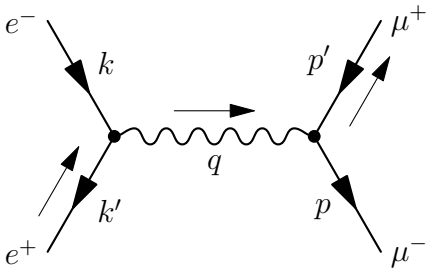
Function and structure definitions can be grouped into modules:

```
// powers.asy
real square(real x) {return x^2;}
real cube(real x) {return x^3;}
```

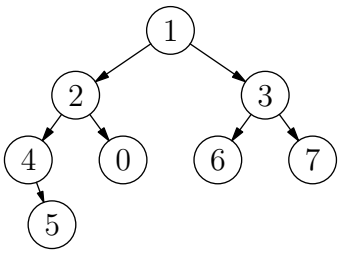
and imported:

```
import powers;
path square(real x) {
  return scale(x)*unitsquare;
}
real four=powers.square(2.0);
real eight=cube(2.0);
```

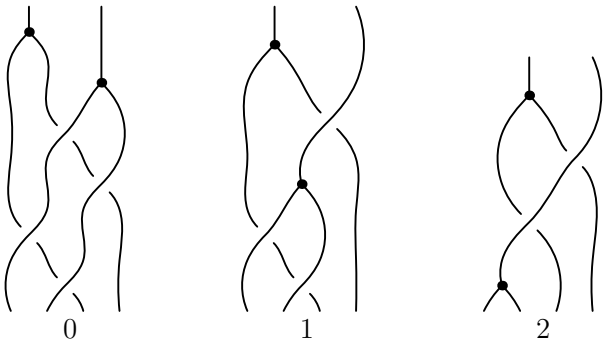
For example, `Asymptote` ships with modules for Feynman diagrams:



data structures,



and algebraic knot theory:



Modules are written in high-level `Asymptote` code. Users have contributed modules tailored to many other specialized applications (such as flowcharts and computer-aided design).

6 Graphics features

METAPOST does not support many important features of PostScript. For example, only connected PostScript subpaths are supported. Regions must be simply connected (have no holes) and can only be filled with uniform RGB colours. In addition to native support for three-dimensional graphics, also lacking in METAPOST, these missing features have been implemented in `Asymptote`.

6.1 Pens

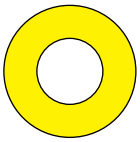
Pens provide a context for the four primitive drawing commands: they specify attributes such as color, line type, line width, text alignment, font, font size, fill rule, and filling patterns. For non-solid line types, dash lengths are by default slightly adjusted to fit the path arc length (for example, to allow publication quality legend entries in graphs with multiple line types). Interesting calligraphic effects are possible by applying transforms to the (normally circular) pen nib or even using a polygonal pen nib (which need not be convex):

The default pen, called `currentpen`, provides the same functionality as the METAPOST `pickup` command. Colors can be specified in any one of the PostScript colorspaces: grayscale, RGB, and CMYK.

6.2 Subpaths

An `Asymptote` path, being connected, is equivalent to a PostScript subpath. The binary operator `^^`, which requests that the pen be moved (without drawing or affecting endpoint curvatures) from the final point of the left-hand path to the initial point of the right-hand path, may be used to group several `Asymptote` connected paths into a `path[]` array (equivalent to a PostScript path). While this facility is merely convenient for drawing an object like the skeleton of a cube (without retracing), it is essential for filling nonsimply-connected regions:

```
path g=scale(2)*unitcircle;
filldraw(unitcircle^^g, evenodd+yellow, black);
```



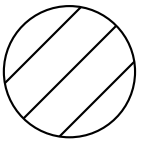
The PostScript even-odd fill rule here specifies that only the region bounded between the two unit circles is to be filled. In this example, the same effect can be achieved by using the default zero winding number fill rule, if one is careful to alternate the orientation of the paths:

```
filldraw(unitcircle^^reverse(g), yellow,
         black);
```

6.3 Patterns

One can also construct custom pictures to be used as tiling patterns for fill or draw operations. The tiling pattern can be assigned a name that can subsequently be used to construct a patterned pen. For example, a hatch pattern can be generated like this:

```
import patterns;
add("hatch",hatch());
filldraw(unitcircle,pattern("hatch"));
```



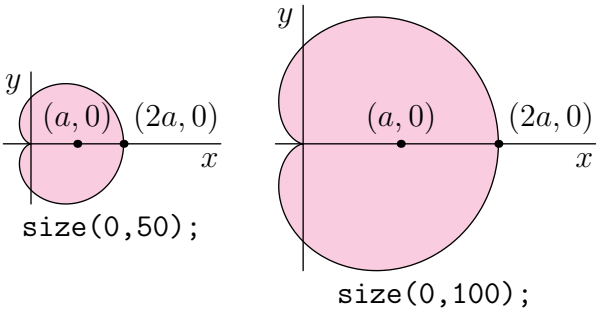
6.4 Shading

`Asymptote` supports axial and radial gradient shading, lattice shading, Gouraud shading, and shading of Coons and tensor product patches; the latter forms are essential for three-dimensional rendering in the presence of a light source. `Asymptote` also supports a true unfill operation implemented with clipping (the METAPOST unfill command simply fills with a fixed background color).

6.5 Automatic picture sizing

A frame is a canvas for drawing in PostScript coordinates, much like a `picture` in METAPOST. However, working directly in PostScript coordinates is often inconvenient, requiring the tedious introduction of manual scaling factors.

Pictures are high-level structures that provide canvases for drawing in a user-specified Cartesian coordinate system. Automatic sizing allows pictures to be constructed in user coordinates and then automatically scaled to the desired final size:



Eventually, one must fit a picture to a PostScript frame. This requires deferred drawing: a graphical object cannot be drawn until the actual scaling of the user coordinates (in terms of PostScript coordinates) is known. One needs to queue a function that can draw the scaled object later, when this scaling is known. For example, the `draw` function for pictures in scalable user coordinates is implemented in terms of the underlying PostScript-coordinate `draw` primitive for frames like this:

```
void draw(picture pic=currentpicture,
          path g, pen p=currentpen) {
  pic.add(new void(frame f, transform t) {
      draw(f,t*g,p);
    });
  pic.addPoint(min(g),min(p));
  pic.addPoint(max(g),max(p));
}
```

Here, the `addPoint` function stores bounding box information as user (e.g. path) coordinates, which scale linearly with the picture size, and true-size (e.g. pen) coordinates, which remain fixed.

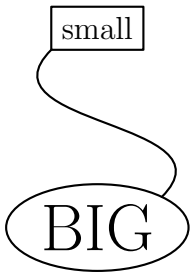
The sizing constraints that arise between scalable objects and fixed-sized attributes (typically labels, dots, linewidths, and arrowheads) reduce to a linear programming problem that is solved by the simplex method. However, a figure can easily produce thousands of restrictions, making direct application of the simplex method time consuming. In practice, most of these restrictions are redundant: in the case of concentric circles, only the largest circle needs to be accounted for. When sizing a picture, `Asymptote` first determines which coordinates are maximal (or minimal) and sends only active constraints to the simplex algorithm. The entire picture-sizing algorithm, including the simplex method, is implemented in high-level `Asymptote` code.

This example illustrates how deferred drawing can be used to draw paths around text labels and then connect them (an `object` is a unifying structure that a label or frame can be implicitly cast to):

```
size(0,100);
pair A=(0,1);
pair B=(0,0);

object small=draw("small",box,A,1mm);
object big=draw("\huge BIG",ellipse,B,1mm);

add(new void(frame f, transform t) {
    draw(f,point(small,SW,t){SW}
           ..{SW}point(big,NE,t));
  });
```

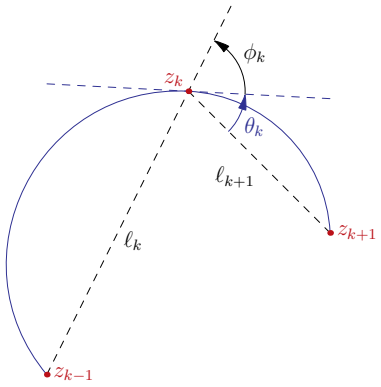


6.6 Three-dimensional graphics

We now describe our three-dimensional generalization of Hobby's prescription for drawing an aesthetically pleasing, numerically efficient, interpolating spline through a set of nodes, given optional tangent directions and endpoint curvatures [2, 3]. This generalization is shape invariant under three-dimensional rotation, scaling, and translation. In the planar case, it reduces to the two-dimensional algorithms found in METAFONT, METAPOST, and `Asymptote`.

In two dimensions, a tridiagonal system of linear equations is first solved to determine any unspecified directions θ_k and ϕ_k through each node z_k:

$$\frac{\theta_{k-1} - 2\phi_k}{\ell_k} = \frac{\phi_{k+1} - 2\theta_k}{\ell_{k+1}}.$$



The resulting shape may be adjusted by modifying the default *tension* parameters and *curl* boundary conditions (cf. [3]).

Having prescribed outgoing and incoming path directions $e^{i\theta}$ at node z_0 and $e^{i\phi}$ at node z_1 relative to the vector $z_1 - z_0$, any unspecified control points are then determined by the equations

$$u = z_0 + e^{i\theta}(z_1 - z_0)f(\theta, -\phi),$$
$$v = z_1 - e^{i\phi}(z_1 - z_0)f(-\phi, \theta),$$

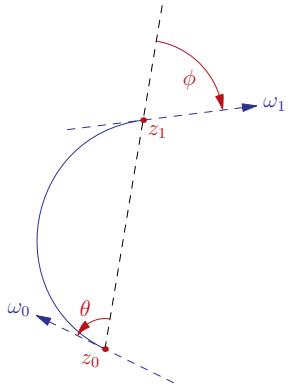
where the *relative distance* function $f(\theta, \phi)$ is given in [2] and [3].

In three dimensions, it is natural to require that our generalization reduce in the planar case to the usual two-dimensional algorithm. Therefore, any unknown incoming or outgoing tangent directions are first determined by applying Hobby's direction

algorithm in successive planes containing the three points z_{k-1}, z_k, and z_{k+1}. The only ambiguity that can arise is in the overall sign of the angles, which relates to viewing each local two-dimensional plane from opposing normal directions. A reference vector constructed from the mean unit normal of successive segments is used to resolve such ambiguities.

A formula for the three-dimensional control points u and v follows on expressing Hobby's algorithm in terms of the absolute incoming and outgoing unit direction vectors ω_0 and ω_1, respectively:

$$u = z_0 + \omega_0 \, |z_1 - z_0| \, f(\theta, -\phi),$$
$$v = z_1 - \omega_1 \, |z_1 - z_0| \, f(-\phi, \theta),$$



where we interpret θ and ϕ as the angle between the corresponding path direction vector and $z_1 - z_0$. In this case there is an unambiguous reference vector for determining the relative sign of the angles θ and ϕ.

Unfortunately, PostScript and PDF support only Bézier splines, which are shape invariant under affine (orthographic) projections (parallel lines being projected to parallel lines), but not perspective projections. In any dimension, applying an affine transformation $x_i' = A_{ij}x_j + C_i$ to a cubic Bézier curve $x(t) = \sum_{k=0}^{3} B_k(t)P_k$ for $t \in [0,1]$, where $B_k(t)$ is the kth cubic Bernstein polynomial yields the Bézier curve

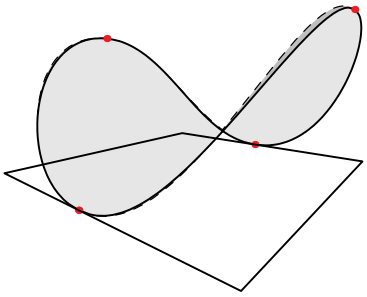
$$x_i'(t) = \sum_{k=0}^{3} B_k(t)A_{ij}(P_k)_j + C_i = \sum_{k=0}^{3} B_k(t)P_k'$$

in terms of the transformed k^{th} control points P_k', noting that $\sum_{k=0}^{3} B_k(t) = 1$. Thus, for orthographic projections, both the nodes and control points of three-dimensional Bézier curves can simply be projected to obtain their two-dimensional counterparts.

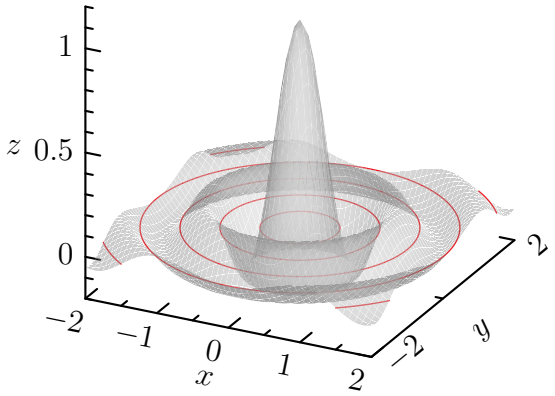
Non-uniform rational B-splines have the advantage of being invariant even in the presence of perspective distortion, since they are Bézier curves in the projective space described by *homogeneous coordinates*, where (x, y, z, w) is considered as equivalent to $(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}, 1)$. For example, the `Asymptote` syntax

```
(1,0,0)..(0,1,1)..(-1,0,0)..(0,-1,1)..cycle
```

describes a saddle-like path in three dimensions. The result of simply projecting the nodes and control points of this three-dimensional Bézier curve to a two dimensional Bézier curve is indicated by the dashed path in the following perspective projection. The true projection, described by the two-dimensional nonuniform rational B-spline represented by the solid curve, can be efficiently approximated as a two-dimensional Bézier curve by introducing additional nodes and control points. An algorithm for doing this efficiently will be presented in a future publication.



The three-dimensional graph of the sinc function below illustrates Gouraud shading, advanced contour path computations, and PDF transparency:



6.7 Scientific graphs

`Asymptote` ships with a sophisticated `graph` module that quickly allows one to draw publication-quality scientific and textbook-style graphs, such as the two-dimensional polar coordinate graph of the cardioid shown in Section 6.5 and the three-dimensional surface plot shown in Section 6.6. It supports features such as legends, custom graph markers, secondary axes, custom (e.g. base 2) axes scalings, broken axes, and custom tick label formats and locations.

7 Graphical user interface

Recent versions of `Asymptote` include an innovative graphical user interface, written in Python/TK, that allows one to modify existing graphical objects and

draw new ones. The modified figure can then be saved and processed as a normal `Asymptote` file. This allows the user to exploit the best features of the script (command-driven) and graphical-interface methods for producing figures.

8 Slide presentations

`Asymptote` also includes a convenient `slide` module for preparing slide presentations, including embedded clickable high-resolution PDF movies (with optional control panels). This module has the advantage over existing LaTeX presentation packages of providing built-in graphics support, including object alignment, in addition to the full power of TeX.

9 Animations

While `Asymptote` can create MPEG and animated GIF movies, the lossless inline PDF movies it can generate with the help of the LaTeX `animate.sty` package are of a much higher quality. Sample animations can be found in the `Asymptote` gallery.

10 Equation solving

Unlike METAFONT and METAPOST, `Asymptote` is not built on top of an implicit linear equation solver and therefore does not automatically have the notion of a `whatever` unknown. Although such an implicit equation facility could certainly be added (perhaps using the notation `?=` since `=` denotes assignment in `Asymptote`), we have noticed that the most common uses of `whatever` in METAPOST are covered by explicit functions like `extension` in the `math` module (which returns the intersection point of the extensions of two line segments). We find the use of routines like `extension` to be more explicit and less confusing, particularly to new users. But we could be persuaded to add implicit equation solving if someone can justify the need (so far no one has provided us with an example that cannot already be done elegantly in `Asymptote`). In the meantime, one can always use the explicit built-in linear solver `solve` (based on LU decomposition) or one of the numerically robust specialized solvers `tridiagonal`, `quadraticroots`, `cubicroots`, and `quarticroots`.

11 Future plans

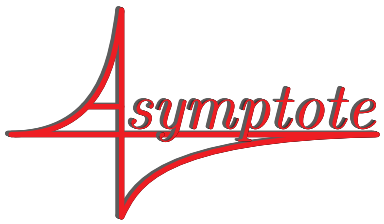
Thanks to the LaTeX `movie15` package, `Asymptote` can embed three-dimensional U3D files into PDF files. In the near future, we plan to generate U3D data (or possibly the more advanced PRC format) directly from `Asymptote`'s internal three-dimensional representations. This will provide the scientific community with a self-contained and powerful facility for generating interactive three-dimensional PDF files.

12 Credits

Financial support for the development of `Asymptote` was generously provided by the Natural Sciences and Engineering Research Council of Canada, the Pacific Institute for Mathematical Sciences, and the University of Alberta Faculty of Science.

We would like to acknowledge enthusiastically the previous work of John D. Hobby, which inspired the development of `Asymptote`, and Donald E. Knuth, for his exceptionally high quality TeX typesetting system, without which none of these later developments would have been possible. We also thank L. Nobre and Troy Henderson for suggesting techniques for approximating nonuniform rational B-splines.

The authors of `Asymptote` are Andy Hammerlindl, John Bowman, and Tom Prince. Sean Healy designed the `Asymptote` logo (below). Other contributors include Radoslav Marinov, Orest Shardt, Chris Savage, Philippe Ivaldi, Olivier Guibé, Jacques Pienaar, Mark Henning, Steve Melenchuk, Martin Wiebusch, and Stefan Knorr.



References

[1] The Art of Problem Solving: Asymptote (Vector Graphics Language). Available online at `http://www.artofproblemsolving.com`, 2007.

[2] John D. Hobby. Smooth, easy to compute interpolating splines. *Discrete Comput. Geom.*, 1:123–140, 1986.

[3] Donald E. Knuth. *The METAFONTbook*. Addison-Wesley, Reading, Massachusetts, 1986.

◇ John C. Bowman
Dept. of Mathematical and Statistical Sciences
University of Alberta
Edmonton, Alberta
Canada T6G 2G1
`bowman (at) math dot ualberta dot ca`
`http://www.math.ualberta.ca/~bowman/`

◇ Andy Hammerlindl
University of Toronto
Dept. of Mathematics
Toronto, Ontario
Canada M5S 2E4
`andy (at) math dot toronto dot edu`

The Luafication of TEX and ConTEXt

Hans Hagen

1 Introduction

Here I will present the current stage of LuaTEX around beta stage 2, and discuss the impact so far on ConTEXt MkIV that we use as our testbed. I'm writing this at the end of February 2008 as part of the series of regular updates on LuaTEX. As such, this report is part of our more or less standard test document (`mk.tex`). More technical details can be found in the reference manual that comes with LuaTEX. More information on MkIV is available in the ConTEXt mailing lists, Wiki, and `mk.pdf`.

For those who never heard of LuaTEX: this is a new variant of TEX where several long pending wishes are fulfilled:

- combine the best of all TEX engines
- add scripting capabilities
- open up the internals to the scripting engine
- enhance font support to OpenType
- move on to Unicode
- integrate MetaPost

There are a few more wishes, like converting the code base to C code but these are long term goals.

The project started a few years ago and is conducted by Taco Hoekwater (Pascal and C code coding, code base management, reference manual), Hartmut Henkel (PDF backend, experimental features) and Hans Hagen (general overview, Lua and TEX coding, website). The code development got a boost by a grant of the Oriental TEX project (project lead: Idris Samawi Hamid) and funding via the TEX Users Group. The related MPlib project by the same team is also sponsored by several user groups. The very much needed OpenType fonts are also a user group funded effort: the Latin Modern and TEX Gyre projects (project leads: Jerzy Ludwichowski, Volker RW Schaa and Hans Hagen), with development (the real work) by: Bogusław Jackowski and Janusz Nowacki.

One of our leading principles is that we focus on opening up. This means that we don't implement solutions (which also saves us many unpleasant and everlasting discussions). Implementing solutions is up to the user, or more precisely: the macro package writer, and since there are many solutions possible, each can do it his or her way. In that sense we follow the footsteps of Don Knuth: we make an extensible tool, you are free to like it or not, you can take it and extend it where needed, and there is no need to bother us (unless of course you find bugs or weird side effects). So far this has worked out quite well and we're confident that we can keep our schedule.

We do our tests of a variant of ConTEXt tagged MkIV, especially meant for LuaTEX, but LuaTEX itself is in no way limited to or tuned for ConTEXt. Large chunks of the code written for MkIV are rather generic and may eventually be packaged as a base system (especially font handling) so that one can use LuaTEX in rather plain mode. To a large extent MkIV will be functionally compatible with MkII, the version meant for traditional TEX, although it knows how to profit from XETEX. Of course the expectation is that certain things can be done better in MkIV than in MkII.

2 Status

By the end of 2007 the second major beta release of LuaTEX was published. In the first quarter of 2008 Taco would concentrate on MPlib, Hartmut would come up with the first version of the image library while I could continue working on MkIV and start using LuaTEX in real projects. Of course there is some risk involved in that, but since we have a rather close loop for critical bug fixes, and because I know how to avoid some dark corners, the risk was worth taking.

What did we accomplish so far? I can best describe this in relation to how ConTEXt MkIV evolved and will evolve. Before we do this, it makes sense to spend some words on why we started working on MkIV in the first place.

When the LuaTEX project started, ConTEXt was about 10 years in the field. I can safely say that we were still surprised by the fact that what at first sight seems unsolvable in TEX somehow could always be dealt with. However, some of the solutions were rather tricky. The code evolved towards a more or less stable state, but sometimes depended on controlled processing. Take for instance backgrounds that can span pages and columns, can be nested and can have arbitrary shapes. This feature has been present in ConTEXt for quite a while, but it involves an interplay between TEX and MetaPost. It depends on information collected in a previous run as well as (at runtime or not) processing of graphics.

This means that by now ConTEXt is not just a bunch of TEX macros, but also closely related to MetaPost. It also means that processing itself is by now rather controlled by a wrapper, in the case of MkII called `texexec`. It may sound complicated, but the fact that we have implemented workflows that run unattended for many years and involve pretty complex layouts and graphic manipulations demonstrates that in practice it's not as bad as it may sound.

With the arrival of LuaTEX we not only have a rigourously updated TEX engine, but also get MetaPost integrated. Even better, the scripting language Lua is not only used for opening up TEX, but is also used for all kind of management tasks. As a result, the development of MkIV not only concerns rewriting whole chunks of ConTEXt, but also results in a set of new utilities and a rewrite of existing ones. Since dealing with MkIV will demand some changes in the way users deal with ConTEXt I will discuss some of them first. It also demonstrates that LuaTEX is more than just TEX.

3 Utilities

There are two main scripts: `luatools` and `mtxrun`. The first one started as a replacement for `kpsewhich` but evolved into a base tool for generating (TDS) file databases and generating formats. In MkIV we replace the regular file searching, and therefore we use a different database model. That's the easy part. More tricky is that we need to bootstrap MkIV into this alternative mode and when doing so we don't want to use the `kpse` library because that would trigger loading of its databases. To discuss the gory details here might cause users to refrain from using LuaTEX so we stick to a general description.

- When generating a format, we also generate a bootstrap Lua file. This file is compiled to bytecode and is put alongside the format file. The libraries of this bootstrap file are also embedded in the format.
- When we process a document, we instruct LuaTEX to load this bootstrap file before loading the format. After the format is loaded, we reinitialize the embedded libraries. This is needed because at that point more information may be available than at loading time. For instance, some functionality is available only after the format is loaded and LuaTEX enters the TEX state.
- File databases, formats, bootstrap files, and runtime-generated cached data is kept in a TDS tree specific cache directory. For instance, OpenType font tables are stored on disk so that next time loading them is faster.

Starting LuaTEX and MkIV is done by `luatools`. This tool is generic enough to handle other formats as well, like `mptopdf` or `plain`. When you run this script without argument, you will see this:

```
version 1.1.1 - 2006+ - PRAGMA ADE / CONTEXT

--generate          generate file database
--variables         show configuration variables
--expansions        show expanded variables
--configurations    show configuration order
--expand-braces     expand complex variable
--expand-path       expand variable
                      (resolve paths)
--expand-var        expand variable
                      (resolve references)
--show-path         show path expansion of ...
--var-value         report value of variable
--find-file         report file location
--find-path         report path of file
--make or --ini     make luatex format
--run or --fmt=     run luatex format
--luafile=str       lua inifile
                      (default is <progname>.lua)
--lualibs=list      optional libraries to assemble
--compile           assemble & compile lua inifile
--verbose           give a bit more info
--minimize          optimize lists for format
--all               show all found files
--sort              sort cached data
--engine=str        target engine
--progname=str      format or backend
--pattern=str       filter variables
--lsr               use lsr and cnf directly
```

For the Lua based file searching, `luatools` can be seen as a replacement for `mktexlsr` and `kpsewhich` and as such it also recognizes some of the `kpsewhich` flags. The script is self-contained in the sense that all needed libraries are embedded. As a result no library paths need to be set and packaged. Of course the script has to be run using LuaTEX itself. The following commands generate the file databases, generate a ConTEXt MkIV format, and process a file:

```
luatools --generate
luatools --make --compile cont-en
luatools --fmt=cont-en somefile.tex
```

There is no need to install Luain order to run this script. This is because LuaTEX can act as such with the advantage that the built-in libraries are available too, for instance the Lua file system `lfs`, the `zip` file manager `zip`, the Unicode library `unicode`, `md5`, and of course some of our own.

- `luatex` — a Lua-enhanced TEX engine
- `texlua` — a Lua engine enhanced with some libraries
- `texluac` — a Lua bytecode compiler enhanced with some libraries

In principle `luatex` can perform all tasks but because we need to be downward compatible with respect to the command line and because we want Lua compatible variants, you can copy or symlink the two extra variants to the main binary.

The second script, `mtxrun`, can be seen as a replacement for the Ruby script `texmfstart`, a utility

```
$ mtxrun --script font  --list --pattern=lmtype.*regular
lmtypewriter10-capsregular   LMTypewriter10-CapsRegular   lmtypewriter10-capsregular.otf
lmtypewriter10-regular       LMTypewriter10-Regular       lmtypewriter10-regular.otf
lmtypewriter12-regular       LMTypewriter12-Regular       lmtypewriter12-regular.otf
lmtypewriter8-regular        LMTypewriter8-Regular        lmtypewriter8-regular.otf
lmtypewriter9-regular        LMTypewriter9-Regular        lmtypewriter9-regular.otf
lmtypewritervarwd10-regular  LMTypewriterVarWd10-Regular  lmtypewritervarwd10-regular.otf
```

Figure 1: Listing fonts available by pattern with `mtxrun`.

whose main task is to launch scripts (or documents or whatever) in a TDS tree. The `mtxrun` script makes it possible to get away from installing Ruby and as a result a regular TEX installation can be made independent of scripting tools.

```
version 1.0.2 - 2007+ - PRAGMA ADE / CONTEXT

--script            run an mtx script
--execute           run a script or program
--resolve           resolve prefixed arguments
--ctxlua            run internally
                     (using preloaded libs)
--locate            locate given filename

--autotree          use texmf tree
                     cf. environment settings
--tree=pathtotree   use given texmf tree
                     (def: 'setuptex.tmf')
--environment=name  use given (tmf)
                      environment file
--path=runpath      go to given path before
                     execution
--ifchanged=file    only execute when given file
                     has changed
--iftouched=old,new only execute when given file
                     has changed

--make              create stubs for
                     (context related) scripts
--remove            remove stubs for
                     (context related) scripts
--stubpath=binpath  paths where stubs
                     will be written
--windows           create windows stubs
--unix              create unix (linux) stubs

--verbose           give a bit more info
--engine=str        target engine
--progname=str      format or backend

--edit              launch editor
                     with found file
--launch (--all)    launch files
                     (assume os support)

--intern            run script using built-in
                     libraries
```

This help information gives an impression of what the script does: running other scripts, either within a certain TDS tree or not, and either conditionally or not. Users of ConTEXt will probably recognize most of the flags. As with `texmfstart`, arguments with prefixes like `file:` will be resolved before being passed to the child process.

The first option, `--script` is the most important one and is used like:

```
mtxrun --script fonts --reload
mtxrun --script fonts --pattern=lm
```

In MkIV you can access fonts by filename or by font name, and because we provide several names per font you can use this command to see what is possible. Patterns can be Lua expressions, as demonstrated in Figure 1.

A simple

```
mtxrun --script fonts
```

gives:

```
version 1.0.2 - 2007+ - PRAGMA ADE / CONTEXT
  font tools

--reload              generate new font database
--list                list installed fonts
--save                save open type font
                       in raw table

--pattern=str         filter files
--all                 provide alternatives
```

In MkIV font names can be prefixed by `file:` or `name:` and when they are resolved, several attempts are made, for instance non-characters are ignored. The `--all` flag shows more variants.

Another example is:

```
mtxrun --script context --ctx=somesetup \
  somefile.tex
```

Again, users of `texexec` may recognize part of this and indeed this is its replacement. Instead of `texexec` we use a script named `mtx-context.lua`. Currently we have the following scripts and more will follow:

`context`	controls processing of files by MkIV
`babel`	conversion tools for LaTeX files
`cache`	utilities for managing the cache
`chars`	utilities used for MkIV development
`check`	TeX syntax checker
`convert`	helper for some basic graphic conversion
`fonts`	utilities for managing font databases
`update`	tool for installing minimal ConTeXt trees
`watch`	hot folder processing tool
`web`	utilities related to automate workflows

The `babel` script is made in cooperation with Thomas Schmitz and can be used to convert babelized Greek files into proper UTF. More of such conversions may follow. With `cache` you can inspect the content of the MkIV cache and do some cleanup. The `chars` script is used to construct some tables that we need in the process of development. As its name says, `check` is a script that does some checks, and in particular it tries to figure out if TeX files are correct. The already mentioned `context` script is the MkIV replacement of `texexec`, and takes care of multiple runs, preloading project specific files, etc. The `convert` script will replace the Ruby script `pstopdf`.

A rather important script is the abovementioned `fonts`. Use this for generating font name databases (which then permits a more liberal access to fonts) or identifying installed fonts. The `unzip` script indeed unzips archives. The `update` script is still somewhat experimental and is one of the building blocks of the ConTeXt minimal installer system by Mojca Miklavec and Arthur Reutenauer. This update script synchronizes a local tree with a repository and keeps an installation as small as possible, which for instance means: no OpenType fonts for pdfTeX, and no redundant Type 1 fonts for LuaTeX and XeTeX.

The (for the moment) last two scripts are `watch` and `web`. We use them in (either automated or not) remote publishing workflows. They evolved out of the Example framework which is currently being reimplemented in Lua.

As you can see, the LuaTeX project and its ConTeXt companion MkIV project not only deal with TeX itself but also facilitate managing the workflows.

There will be more scripts. These scripts are normally rather small because they hook into `mtxrun` which provides the libraries. Of course existing tools remain part of the toolkit. Take for instance `ctxtools`, a Ruby script that converts font encoded pattern files to generic UTF encoded files.

Those who have followed the development of ConTeXt will notice that we moved from utilities written in Modula to tools written in Perl. These were later replaced by Ruby scripts and eventually most of them will be rewritten in Lua.

4 Macros

I will not repeat what is said already in the MkIV related documents, but stick to a summary of what the impact on ConTeXt is and will be. From this you can deduce what the possible influence on other macro packages can be.

Opening up TeX started with rewriting all I/O related activities. Because we wanted to be able to read from `zip` files, the web and more, we moved away from the traditional `kpse` based file handling. Instead MkIV uses an extensible variant written in Lua. Because we need to be downward compatible, the code is somewhat messy, but it does the job, and pretty quickly and efficiently too. Some alternative input media are implemented and many more can be added. In the beginning I permitted several ways to specify a resource but recently a more restrictive url syntax was imposed. Of course the file locating mechanisms provide the same control as provided by the file readers in MkII.

An example of reading from a `zip` file is:

```
\input zip:///archive.zip?name=foo.tex
\input zip:///archive.zip?name=/somepath/foo.tex
```

In addition one can register files, like:

```
\usezipfile[archive.zip]
\usezipfile[tex.zip][texmf-local]
\usezipfile[tex.zip?tree=texmf-local]
```

The last two variants register a zip file in the TDS structure where more specific lookup rules apply. The files in a registered file are known to the file searching mechanism so one can give specifications like the following:

```
\input */foo.tex
\input */somepath/foo.tex
```

In a similar fashion one can use the `http`, `ftp` and other protocols. For this we use independent fetchers that cache data in the MkIV cache. Of course, in more structured projects, one will seldom use the `\input` command but use a project structure instead.

Handling of files rather quickly reached a stable state, and we seldom need to visit the code for fixes. Already after a few years of developing the first code for LuaTeX we reached a state of 'Hm, when did I write this?'. When we have reached a stable state I foresee that much of the older code will need a cleanup.

Related to reading files is the sometimes messy area of input regimes (file encoding) and font encoding, which itself relates to dealing with languages. Since LuaTeX is UTF-8 based, we need to deal with file encoding issues in the frontend, and this is what Lua based file handling does. In practice users of

LuaTEX will swiftly switch to UTF anyway but we provide regime control for historic reasons. This time the recoding tables are Lua based and as a result MkIV has no regime files. In a similar fashion font encoding is gone: there is still some old code that deals with default fallback characters, but most of the files are gone. The same will be true for math encoding. All information is now stored in a character table which is the central point in many subsystems now.

It is interesting to notice that until now users have never asked for support with regards to input encoding. We can safely assume that they just switched to UTF and recoded older documents. It is good to know that LuaTEX is mostly pdfTEX but also incorporates some features of Omega. The main reason for this is that the Oriental TEX project needed bidirectional typesetting and there was a preference for this implementation over the one provided by ε-TEX. As a side effect input translation is also present, but since no one seems to use it, that may as well go away. In MkIV we refrain from input processing as much as possible and focus on processing the node lists. That way there is no interference between user data, macro expansion and whatever may lead to the final data that ends up in the to-be-typeset stream. As said, users seem to be happy to use UTF as input, and so there is hardly any need for manipulations.

Related to processing input is verbatim: a feature that is always somewhat complicated by the fact that one wants to typeset a manual about TEX in TEX and therefore needs flexible escapes from illustrative as well as real TEX code. In MkIV verbatim as well as all buffering of data is dealt with in Lua. It took a while to figure out how LuaTEX should deal with the concept of a line ending, but we got there. Right from the start we made sure that LuaTEX could deal with collections of catcode settings (those magic states that characters can have). This means that one has complete control at both the TEX and Lua end over the way characters are dealt with.

In MkIV we also have some pretty printing features, but many languages are still missing. Cleaning up the premature verbatim code and extending pretty printing is on the agenda for the end of 2008.

Languages also are handled differently. A major change is that pattern files are no longer preloaded but read in at runtime. There is still some relation between fonts and languages, no longer in the encoding but in dealing with OpenType features. Later we will do a more drastic overhaul (with multiple name schemes and such). There are a few experimental features, like spell checking.

Because we have been using UTF encoded hyphenation patterns for quite some time now, and because ConTEXt ships with its own files, this transition probably went unnoticed, apart maybe from a faster format generation and less startup time.

Most of these features started out as an experiment and provided a convenient way to test the LuaTEX extensions. In MkIV we go quite far in replacing TEX code by Lua, and how far one goes is a matter of taste and ambition. An example of a recent replacement is graphic inclusion. This is one of the oldest mechanisms in ConTEXt and it has been extended many times, for instance by plugins that deal with figure databases (selective filtering from PDF files made for this purpose), efficient runtime conversion, color conversion, downsampling and product dependent alternatives.

One can question if a properly working mechanism should be replaced. Not only is there hardly any speed to gain (after all, not that many graphics are included in documents), a Lua–TEX mix may even look more complex. However, when an opened-up TEX keeps evolving at the current pace, this last argument becomes invalid because we can no longer give that TEXie code to Lua. Also, because most of the graphic inclusion code deals with locating files and figuring out the best quality variant, we can benefit much from Lua: file handling is more robust, the code looks cleaner, complex searches are faster, and eventually we can provide way more clever lookup schemes. So, after all, switching to Lua here makes sense. A nice side effect is that some of the mentioned plugins now take a few lines of extra code instead of many lines of TEX. At the time of writing this, the beta version of MkIV has Lua based graphic inclusion.

A disputable area for Luafication is multipass data. Most of that has already been moved to Lua files instead of TEX files, and the rest will follow: only tables of contents still use a TEX auxiliary file. Because at some point we will reimplement the whole section numbering and cross referencing, we postponed that till later. The move is disputable because in the end, most data ends up in TEX again, which involves some conversion. However, in Lua we can store and manipulate information much more easily and so we decided to follow that route. As a start, index information is now kept in Lua tables, sorted on demand, depending on language needs and such. Positional information used to take up much hash space which could deplete the memory pool, but now we can have millions of tracking points at hardly any cost.

Because it is a quite independent task, we could rewrite the MetaPost conversion code in Lua quite

early in the development. We got smaller and cleaner code, more flexibility, and also gained some speed. The code involved in this may change as soon as we start experimenting with MPlib. Our expectations are high because in a bit more modern designs a graphic engine cannot be missed. For instance, in educational material, backgrounds and special shapes are all over the place, and we're talking about many MetaPost runs then. We expect to bring down the processing time of such documents considerably, if only because the MetaPost runtime will be close to zero (as experiments have shown us).

While writing the code involved in the MetaPost conversion a new feature showed up in Lua: `lpeg`, a parsing library. From that moment on `lpeg` was being used all over the place, most noticeably in the code that deals with processing XML. Right from the start I had the feeling that Lua could provide a more convenient way to deal with this input format. Some experiments with rewriting the MkII mechanisms did not show the expected speedup and were abandoned quickly.

Challenged by `lpeg` I then wrote a parser and started playing with a mixture of a tree based and stream approach to XML (MkII is mostly stream based). Not only is loading XML code extremely fast (we used 40 megabyte files for testing), dealing with the tree is also convenient. The additional MkIV methods are currently being tested in real projects and so far they result in an acceptable and pleasant mix of TEX and XML. For instance, we can now selectively process parts of the tree using path expressions, hook in code, manipulate data, etc.

The biggest impact of LuaTEX on the ConTEXt code base is not the previously mentioned mechanisms but one not yet mentioned: fonts. Contrary to XETEX, which uses third party libraries, LuaTEX does not implement dealing with font specific issues at all. It can load several font formats and accepts font data in a well-defined table format. It only processes character nodes into glyph nodes and it's up to the user to provide more by manipulating the node lists. Of course there is still basic ligature building and kerning available but one can bypass that with other code.

In MkIV, when we deal with Type 1 fonts, we try to get away from traditional TFM files and use AFM files instead (indeed, we parse them using `lpeg`). The fonts are mapped onto Unicode. Awaiting extensions of math we only use TFM files for math fonts. Of course OpenType fonts are dealt with and this is where we find most Lua code in MkIV: implementing features. Much of that is a grey area but as part of the Oriental TEX project we're forced to deal with complex feature support, so that provides a good test bed as well as some pressure for getting it done. Of course there is always the question to what extent we should follow the (maybe faulty) other programs that deal with font features. We're lucky that the Latin Modern and TEX Gyre projects provide real fonts as well as room for discussion and exploring these grey areas.

In parallel to writing this, I made a tracing feature for Oriental TEXer Idris so that he could trace what happened with the Arabic fonts that he is making. This was relatively easy because already in an early stage of MkIV some debugging mechanisms were built. One of its nice features is that on an error, or when one traces something, the results will be shown in a web browser. Unfortunately I have not enough time to explore such aspects in more detail, but at least it demonstrates that we can change some aspects of the traditional interaction with TEX in more radical ways.

Many users may be aware of the existence of so-called virtual fonts, if only because it can be a cause of problems (related to map files and such). Virtual fonts have a lot of potential but because they were related to TEX's own font data format they never got very popular. In LuaTEX we can make virtual fonts at runtime. In MkIV for instance we have a feature (we provide features beyond what OpenType does) that completes a font by composing missing glyphs on the fly. More of this trickery can be expected as soon as we have time and reason to implement it.

In pdfTEX we have a couple of font related goodies, like character expansion (inspired by Hermann Zapf) and character protruding. There are a few more but these had limitations and were suboptimal and therefore have been removed from LuaTEX. After all, they can be implemented more robustly in Lua. The two mentioned extensions have been (of course) kept and have been partially reimplemented so that they are now uniquely bound to fonts (instead of being common to fonts that traditional TEX shares in memory). The character related tables can be filled with Lua and this is what MkIV now does. As a result much TEX code could go away. We still use shape related vectors to set up the values, but we also use information stored in our main character database.

A likely area of change is math and not only as a result of the TEX Gyre math project which will result in a bunch of Unicode compliant math fonts. Currently in MkIV the initialization already partly takes place using the character database, and so again we will end up with less TEX code. A side effect of removing encoding constraints (i.e. moving

to Unicode) is that things get faster. Later this year math will be opened up.

One of the biggest impacts of opening up is the arrival of attributes. In traditional TEX only glyph nodes have an attribute, namely the font id. Now all nodes can have attributes, many of them. We use them to implement a variety of features that already were present in MkII, but used marks instead: color (of course including color spaces and transparency), inter-character spacing, character case manipulation, language dependent pre and post character spacing (for instance after colons in French), special font rendering such as outlines, and much more. An experimental application is a more advanced glue/penalty model with look-back and look-ahead as well as relative weights. This is inspired by the one good thing that XML formatting objects provide: a spacing and pagebreak model.

It does not take much imagination to see that features demanding processing of node lists come with a price: many of the callbacks that LuaTEX provides are indeed used and as a result quite some time is spent in Lua. You can add to that the time needed for handling font features, which also boils down to processing node lists. The second half of 2007 Taco and I spent much time on benchmarking and by now the interface between TEX and Lua (passing information and manipulating nodes) has been optimized quite well. Of course there's always a price for flexibility and LuaTEX will never be as fast as pdfTEX, but then, pdfTEX does not deal with OpenType and such.

We can safely conclude that the impact of LuaTEX on ConTEXt is huge and that fundamental changes take place in all key components: files, fonts, languages, graphics, MetaPost XML, verbatim and color to start with, but more will follow. Of course there are also less prominent areas where we use Lua based approaches: handling url's, conversions, alternative math input to mention a few. Sometime in 2009 we expect to start working on more fundamental typesetting related issues.

5 Roadmap

On the LuaTEX website `http://www.luatex.org` you can find a roadmap. This roadmap is just an indication of what has happened and will happen and it will be updated when we feel the need. Here is a summary:

- merging engines

  Merge some of the Aleph codebase into pdfTEX (which already has ε-TEX) so that LuaTEX in DVI mode behaves like Aleph, and in PDF mode like pdfTEX. There will be Lua callbacks for file searching. This stage is mostly finished.

- OpenType fonts

  Provide PDF output for Aleph bidirectional functionality and add support for OpenType fonts. Allow Lua scripts to control all aspects of font loading, font definition and manipulation. Most of this is finished.

- tokenizing and node lists

  Use Lua callbacks for various internals, complete access to tokenizer and provide access to node lists at moments that make sense. This stage is completed.

- paragraph building

  Provide control over various aspects of paragraph building (hyphenation, kerning, ligature building), dynamic loading of hyphenation patterns. Apart from some small details these objectives are met.

- MetaPost (MPlib)

  Incorporate a MetaPost library and investigate options for runtime font generation and manipulation. This activity is on schedule and integration will take place before summer 2008.

- image handling

  Image identification and loading in Lua including scaling and object management. This is nicely on schedule, the first version of the image library showed up in the 0.22 beta and some more features are planned.

- special features

  Cleaning up of Hz optimization and protruding and getting rid of remaining global font properties. This includes some cleanup of the backend. Most of this stage is finished.

- page building

  Control over page building and access to internals that matter. Access to inserts. This is on the agenda for late 2008.

- TEX primitives

  Access to and control over most TEX primitives (and related mechanisms) as well as all registers. Especially box handling has to be reinvented. This is an ongoing effort.

- PDF backend

  Open up most backend related features, like annotations and object management. The first code will show up at the end of 2008.

- math

  Open up the math engine parallel to the development of the TEX Gyre math fonts. Work

on this will start during 2008 and we hope that it will be finished by early 2009.

- CWEB

  Convert the TEX Pascal source into CWEB and start using Lua as glue language for components. This will be tested on MPlib first. This is on the long term agenda, so maybe around 2010 you will see the first signs.

In addition to the mentioned functionality we have a couple of ideas that we will implement along the road. The first formal beta was released at TUG 2007 in San Diego (USA). The first formal release will be at TUG 2008 in Cork (Ireland). The production version will be released at EuroTEX in the Netherlands (2009).

Eventually LuaTEX will be the successor to pdfTEX (informally we talk of pdfTEX version 2). It can already be used as a drop-in for Aleph (the stable variant of Omega). It provides a scripting engine without the need to install a specific scripting environment. These factors are among the reasons why distributors have added the binaries to the collections. Norbert Preining maintains the Linux packages, Akira Kakuto provides Windows binaries as part of his distribution, Arthur Reutenauer takes care of Mac OS X and Christian Schenk recently added LuaTEX to MiKTEX. The LuaTEX and MPlib projects are hosted at Supelec by Fabrice Popineau (one of our technical consultants). And with Karl Berry being one of our motivating supporters, you can be sure that the binaries will end up someplace in TEX Live this year.

◇ Hans Hagen
PRAGMA ADE
http://pragma-ade.com

Porting TEX Live to OpenBSD

Edward Barrett

Abstract

The history, creation, and fruition of porting TEX Live to OpenBSD.

1 Why TEX?

At the time that I became a student, I had been using *NIX systems for years, in particular an operating system called OpenBSD, which is one of the BSD[1] derived open-source Unix-a-likes. OpenBSD aims to be "Free, Functional and Secure", but it also has other qualities which made it appealing to me: it was small, it did not demand a fast computer and is developed with "correctness" in mind.

The existing WYSIWYG[2] document preparation software packages for OpenBSD either did not suit the nature of the document, or did not feel natural to me as a computing student. In particular:

- Most of my assignments required source code listings. Standard procedure generally consisted of copy and pasting large chunks of code into a word processor (screen by screen if you were in a terminal appplication). It did work, but it was tedious to repeat when the sources changed.
- During my two years in industry, Vim[3] had become second nature to me when editing text. In comparison a word processor seemed very limited and inefficient.

It was only when I saw what some of the members of staff at University had done with TEX that I became fully aware of its purpose. The concept of logical markup in a document felt right, and my previous gripes with word processors could be solved, not to mention that documents typeset with TEX looked a much more professional than anything I could make with a word processor. I started learning how to make documents like this myself using the existing teTEX package on my OpenBSD systems.

2 The retirement of the teTEX distribution

Oddly enough, the porting process of TEX Live started with the simple need for the rcs package in a document. I had one day decided that a verbatim CVS tag in the preface of my documents looked ugly and that I should find a solution.

A CTAN search revealed the rcs package, but it was not included in teTEX. I visited the teTEX web page to see if there was a possibility of including it in future versions. It was then that I realized that teTEX development had sadly been retired for almost a year. The web page suggested that I pursue a project called "TEX Live", so I did. I was mostly pleased with what I saw, but found the ISO disk image distribution format inconvenient, and the DVD only supplied i386 binaries for OpenBSD, which needless to say would not work on my Sun Sparc systems.

This was when I started the porting process, after an email to the `ports@openbsd.org` mailing list in March 2007 to see if anyone else had been working on a port. At the time I remember thinking that this was a week's work at most. How wrong I was ...

3 Porting

My initial build confirmed that TEX Live was quite happy to build on OpenBSD. Now I had to integrate the build with the OpenBSD "ports" build system. The "ports" build system is basically a set of instructions (in the form of Makefiles) for building third party software, whose concept was originally devised by the FreeBSD project. NetBSD also has a similar system, as does the Gentoo Linux project. The advantage of these systems is that packages need not be manually built upon each new release of the operating system, as it can be powerfully automated.

Getting TEX Live to play nice with the ports infrastructure was not easy. A lot of the TEX Live build system did not honour the GNU standard *DESTDIR* environment variable, which is required to fool software into installing into a fake root filesystem ready for packaging. Luckily a tool called systrace allowed me to detect these bugs and I was able to "patch them away" before the build commenced. I applied just under 40 patches to the TEX Live build system, resulting in a package named *texlive_base-2007*. Following the conventions of the existing teTEX port, I made the *texmf* tree a separate package named *texlive_texmf-2007*.

On May 8th 2007 I posted my hopefully "completed" work onto the porting mailing list and waited.

Responses to my work were positive, but there was concern with the size of the texmf package which had exceeded 512 MB. I was asked to "trim it down", if at all possible. The next few months proved to be the most difficult stage of the port, not only because I was studying for exams *as well as* porting, but also because I had uncovered some nasty bus errors in XETEX when run on the sparc64 CPU architecture. I think my housemates must have thought I was chasing a lost cause, but I was not prepared to give up.

[1] Berkeley Software Distribution
[2] What you see is what you get
[3] Vi Improved, `http://www.vim.org`

I decided to first concentrate on splitting the *texmf* tree. This involved learning how the (now deprecated) "TPM"[4] hierarchy worked as well as the various configuration files, so that they matched the relevant *texmf* subset that was installed. Norbert Preining had done a similar task for Debian Linux and generously shared what he knew. Also, a lecturer at University, Peter Knaggs, who had been using TEX for a while was able to explain some of the mechanisms of a TEX system. A Python script I called *MFSplit* sprang into existence. This creation split the large *texmf* tree into three smaller counterparts (minimal, full and documentation) and helped to generate configuration files. I must admit I found this quite challenging, even frustrating at times, and the TEX Live mailing list was a invaluable resource during times of confusion.

A major cause of the aforementioned frustration was due to the slowness of my dated Sun hardware. In an attempt to find something more practical, I placed a banner on my web page asking for hardware donations. I had expected to hear nothing. Within a few days one individual donated some money for a beer, which pleasantly shocked me, but not as much as when a company contacted me saying that they wished to donate a machine to the port! The company was *yellowshift LLC*, a consultancy company in America that believed in giving back to open source (`https://www.yellowshift.com/giving-back`). Gratefully I accepted and now I am a proud owner of a Sun Blade 1000 for development of TEX Live/OpenBSD for sparc64 systems.

As for the XeTEX issues, the XeTEX author and a very dedicated member of the TEX community, Jonathan Kew, helped me out greatly. It proved very difficult to debug over email, so I ended up granting him remote access to my build box in order for him to look more closely at the problem. Very quickly he put together some patches which fixed the issue (and a few others too), which were committed upstream and to the port.

4 Integrating TEX Live into OpenBSD

On July 17th 2007 the TEX Live port was committed on my behalf to the head branch of the ports tree, but not yet linked to the build as a replacement for teTEX, since further community testing was needed. Small bug fixes and cleanups were applied. Eventually, on October 11th 2007, TEX Live replaced teTEX entirely for OpenBSD developer snapshots, and will certainly be included in upcoming OpenBSD-4.3. I consider this is a great personal achievement, as I had always wished I could give something back to the open source community. I have been in contact with several developers interested in developing ports of TEX Live for their operating systems, including a developer from the Macports project who succeeded in translating the OpenBSD port to their ports system. I certainly do enjoy the freedom of open source software development.

If you are running the developer branch of OpenBSD (or *-current* as it is known), then you can get TEX Live now from the OpenBSD FTP servers. OpenBSD 4.3 pre-orders are available now; this is a great way to help the project.

5 Thanks

I would like to take the opportunity to thank all of the people who have answered my questions on mailing lists, donated resources, helped me bug-fix or otherwise contributed to porting TEX Live to OpenBSD. It is greatly appreciated and I hope to meet some of you at the conference in Cork.

6 About the author

Allow me to introduce myself, as I am new to the TEX community. I am a student at Bournemouth University in England. As far as I can remember, I have always been fascinated by computers, starting with the Sinclair Spectrum 48K. After messing about on computers most of my childhood (and advancing through almost all of the Sinclair Spectrum models), I studied computing at college and entered the *real* world as a programmer for a kitchen manufacture company. During this time I learned a web scripting language inside out, programmed some really interesting industrial machinery and pulled out a lot of hair trying to interpret 20-year-old BASIC code. Having done this for two years, I felt a change was in order and enrolled for a BSc computing degree. At the time of writing I am on the 3rd year of the degree, which is a "sandwich year" in which students work in industry for a year before returning to University to complete the final year. It turned out that I really liked the academic environment and wished to look into the possibility of an academic future. After enquiring if the University accepted placement students, I became a Unix systems administrator for the school of design, engineering and computing.

◇ Edward Barrett
`eddbarrett (at) googlemail dot com`
`http://students.dec.bournemouth.ac.uk/ebarrett`

[4] TEX Package Manager

LaTeX

Good things come in little packages: An introduction to writing .ins and .dtx files

Scott Pakin

Abstract

LaTeX packages made available from CTAN are commonly distributed as a pair of files: ⟨*something*⟩`.ins` and ⟨*something*⟩`.dtx`. The user is then instructed to run the `.ins` file through `latex` to produce the actual package files. What are these `.ins` and `.dtx` files? How do you, as a class or style-file writer, create your own? And, why would you want to? This article answers those questions and elucidates the mysterious techniques underlying LaTeX package distribution.

1 Introduction

A typical CTAN package comprises a `README` file, some PDF documentation, an `.ins` file, and a `.dtx` file. Running the `.ins` file through `latex` creates one or more `.sty`, `.cls`, `.def`, or other files that the user can install. Few LaTeX users and developers understand the reasoning behind that extra step or the purpose of the seemingly unnecessary `.dtx` file.

Before we examine `.ins` and `.dtx` files in depth, let us consider a coding example from outside the TeX world. Figure 1 presents a function in the C programming language for solving a quadratic equation. Comments at the top of the function are used to explain what the function does. Although comments are intended to be human-readable, only simple text can be used to format comments. Wouldn't it be nice if comments and the code they describe could be typeset using a tool such as LaTeX, as in Figure 2? Even for short programs, including mathematics, figures, and tables in comments can assist readability. For longer programs, sectioning commands, cross references (maybe even with hyperlinks), indexes, and a table of contents can be quite beneficial for explaining the program's purpose and usage to readers. However, having to maintain two versions of a program — a nicely formatted version with typeset documentation for human readers and a text-only version for the compiler — is an approach doomed to failure as the two versions will inevitably drift apart.

The idea behind a `.dtx` file is to maintain a single version of a program yet be able to process it either as a typeset document or as compilable code. As far as the `latex` compiler is concerned, a `.dtx` file is an ordinary document; it just happpens to describe a program. However, when the corresponding `.ins` file is processed, the (text-only) program is extracted from the `.dtx` file to one or more separate files.

Placing emphasis on providing thorough, typeset code documentation intertwined with the code itself is commonly known as *literate programming* [1]. Literate programming is particularly apropos for documenting LaTeX packages because of the esotericism of the TeX language and the consequent need for copious explanation. The mechanisms needed to implement literate programming in `.ins` and `.dtx` files are provided by two packages that come standard with all LaTeX 2_ε distributions: Doc [3, 4] for typesetting, formatting, and indexing LaTeX macro and environment definitions, and, DocStrip [5] for extracting the code from a literate program while stripping away all of the commentary.

2 Installer (.ins) files

The first step in preparing a package for distribution is to write an *installer* (`.ins`) file. An installer file extracts the code from a `.dtx` file, uses DocStrip to strip off the comments and documentation, and outputs a `.sty` file. The good news is that a `.ins` file is typically fairly short and doesn't change significantly from one package to another.

Figure 3 presents a typical `.ins` file. An `.ins` file usually begins with a comment block that states the package's copyright notice and license agreement (lines 1–13). Most of the commands that appear in an `.ins` file are provided by the DocStrip package so that is loaded in line 15. DocStrip is normally excessively verbose about its operation so Figure 3 includes an invocation of `\keepsilent` (line 16) to instruct DocStrip to output only the most important information.

A package can invoke the `\usedir` macro (as on line 18) to specify a preferred installation directory relative to the root of the TeX directory tree. In practice, the `\usedir` call serves primarily as a comment and is seldom used to automatically place files in their final destination.

Lines 20–36 of Figure 3 specify a set of comments to include at the top of every file that the `.ins` file generates. Typically, these comments include a remark that the file is generated plus a repetition of the package's copyright notice and license agreement.

The most important line in an `.ins` file is the call to `\generate`. The `\generate` macro is the mechanism by which an `.ins` file instructs DocStrip how to extract the various package files from an accompanying `.dtx` file. Line 38 of Figure 3 should be interpreted as the instruction, "Generate a file called `mypackage.sty` by extracting all text marked with

```
/* Use the quadratic formula (x=(-b +/- sqrt(b^2-4ac))/2a) to store the two
 * roots of ax^2+bx+c=0 in x1 and x2.  Return 1 on success, 0 on failure
 * (if a=0 or the roots are complex). */
int solve_quadratic (double a, double b, double c, double *x1, double *x2)
{
  double discrim = b*b - 4*a*c;

  if (a == 0.0 || discrim < 0.0)
    return 0;
  *x1 = (-b + sqrt(discrim)) / (2*a);
  *x2 = (-b - sqrt(discrim)) / (2*a);
  return 1;
}
```

Figure 1: Compiler-readable C code for solving a quadratic equation

solve_quadratic() Use the quadratic formula ($x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$) to store the two roots of $ax^2+bx+c = 0$ in x_1 and x_2. Return `1` on success, `0` on failure (if $a = 0$ or the roots are complex).

```
1  int solve_quadratic (double a, double b, double c, double *x1, double *x2)
2  {
3      double discrim = b*b - 4*a*c;
4
5      if (a == 0.0 || discrim < 0.0)
6        return 0;
7      *x1 = (-b + sqrt(discrim)) / (2*a);
8      *x2 = (-b - sqrt(discrim)) / (2*a);
9      return 1;
10 }
```

Figure 2: Typeset C code for solving a quadratic equation

the tag `package` in the file called `mypackage.dtx`."

The `\generate` command is fairly flexible in that a file can be generated from multiple tags spread across multiple files. In fact, blocks of code can be shared by multiple generated files. As a fairly complex example, consider the `\generate` command used by LaTeX 2ε's `classes.ins` file to generate all of the standard LaTeX 2ε class files and their per-size helper files. As the excerpt from `classes.ins` shown in Figure 4 indicates, both `size10.clo` and `bk10.clo` are produced by extracting all text from `classes.dtx` that is marked with the `10pt` tag. The `bk10.clo` file additionally includes all text marked with the `bk` tag. The same `bk`-tagged text is copied into `bk11.clo` and `bk12.clo` as well.

Returning to our complete example of an `.ins` file in Figure 3, DocStrip provides a `\Msg` macro that outputs a message to the standard output device. It is helpful to use `\Msg` to inform the user what files were extracted and need to be installed. Lines 40–53 in Figure 3 output a typical end-of-installation message. Note the use of `\obeyspaces` in line 40 to prevent TeX from collapsing multiple spaces into a single space in the subsequent `\Msg` invocations.

An `.ins` file ends with a call to `\endbatchfile`, as shown in line 55.

3 Documented LaTeX (`.dtx`) files

A *documented LaTeX* (`.dtx`) file contains both the commented source code and the user documentation for the package. Running a `.dtx` file through `latex` typesets the user documentation, which usually also includes a nicely typeset version of the commented source code.

Due to some Doc trickery, `latex` actually evaluates a `.dtx` file *twice* when generating documentation. On the first pass, only a small piece of `latex` driver code is evaluated. The second time, *comments* in the `.dtx` file are evaluated, as if there were no "`%`" preceding them. This can lead to a great deal of confusion when writing `.dtx` files and occasionally leads to some awkward constructions. Fortunately, once the basic structure of a `.dtx` file is in place, filling in the code is fairly straightforward.

```
%%
%% Copyright (C) 2008 by Your Name Here <you@yournamehere.org>
%%
%% This file may be distributed and/or modified under the conditions of
%% the LaTeX Project Public License, either version 1.3c of this license
%% or (at your option) any later version.  The latest version of this
%% license is in:
%%
%%    http://www.latex-project.org/lppl.txt
%%
%% and version 1.3c or later is part of all distributions of LaTeX
%% version 2006/05/20 or later.
%%

\input docstrip.tex
\keepsilent

\usedir{tex/latex/mypackage}

\preamble

This is a generated file.

Copyright (C) 2008 by Your Name Here <you@yournamehere.org>

This file may be distributed and/or modified under the conditions of
the LaTeX Project Public License, either version 1.3c of this license
or (at your option) any later version.  The latest version of this
license is in:

   http://www.latex-project.org/lppl.txt

and version 1.3c or later is part of all distributions of LaTeX
version 2006/05/20 or later.

\endpreamble

\generate{\file{mypackage.sty}{\from{mypackage.dtx}{package}}}

\obeyspaces
\Msg{*************************************************************}
\Msg{*                                                           *}
\Msg{* To finish the installation you have to move the following *}
\Msg{* file into a directory searched by TeX:                    *}
\Msg{*                                                           *}
\Msg{*     mypackage.sty                                         *}
\Msg{*                                                           *}
\Msg{* To produce the documentation run the file mypackage.dtx   *}
\Msg{* through LaTeX.                                           *}
\Msg{*                                                           *}
\Msg{* Happy TeXing!                                            *}
\Msg{*                                                           *}
\Msg{*************************************************************}

\endbatchfile
```

Figure 3: A typical `.ins` file

```
\generate{\file{article.cls}{\from{classes.dtx}{article}}
          \file{report.cls}{\from{classes.dtx}{report}}
          \file{book.cls}{\from{classes.dtx}{book}}
          \file{size10.clo}{\from{classes.dtx}{10pt}}
          \file{size11.clo}{\from{classes.dtx}{11pt}}
          \file{size12.clo}{\from{classes.dtx}{12pt}}
          \file{bk10.clo}{\from{classes.dtx}{10pt,bk}}
          \file{bk11.clo}{\from{classes.dtx}{11pt,bk}}
          \file{bk12.clo}{\from{classes.dtx}{12pt,bk}}
          }
```

Figure 4: Excerpt from LaTeX 2$_\varepsilon$'s `classes.ins` file

```
1  % \iffalse meta-comment
2  %
3  % Copyright (C) 2008 by Your Name Here <you@yournamehere.org>
4  % ------------------------------------------------------------
5  %
6  % This file may be distributed and/or modified under the conditions of
7  % the LaTeX Project Public License, either version 1.3c of this license
8  % or (at your option) any later version.  The latest version of this
9  % license is in:
10 %
11 %    http://www.latex-project.org/lppl.txt
12 %
13 % and version 1.3c or later is part of all distributions of LaTeX
14 % version 2006/05/20 or later.
15 %
16 % \fi
17 %
```

Figure 5: `.dtx` header comments

3.1 Package identification

A `.dtx` file traditionally begins with a copyright and license notice, which are formatted as in Figure 5. The significance of the `\iffalse` ... `\fi` construct is that on `latex`'s second pass through the `.dtx` file, commented lines are processed as if they were uncommented. To prevent the copyright and license statement from appearing at the beginning of the typeset document we wrap them within a conditional that will never be true. The `meta-comment` after `\iffalse` is nothing more than a convention for indicating that the comment is intended to be read by a human, not by Doc, DocStrip, or `latex`.

The next block of `.dtx` code (Figure 6) identifies the package. On `latex`'s first pass through the `.dtx` file, "`%`" introduces a comment line, as normal. Hence, `latex` sees only the `\ProvidesFile` command (line 20) and its optional argument (line 25). The optional argument must be in the format shown: package date (YYYY/MM/DD), package version, and package description. The Doc package parses the optional argument into three macros — `\filedate`, `\fileversion`, and `\fileinfo` — that can be used to automatically date-stamp and version-stamp the documentation. On `latex`'s second pass through the file, the `\iffalse`, which is now executed, tells `latex` to disregard the entire block of code shown in Figure 6.

```
18 % \iffalse
19 %<*driver>
20 \ProvidesFile{mypackage.dtx}
21 %</driver>
22 %<package>\NeedsTeXFormat{LaTeX2e}[2003/12/01]
23 %<package>\ProvidesPackage{mypackage}
24 %<*package>
25     [2008/02/18 v1.0 My sample package]
26 %</package>
27 %
```

Figure 6: `.dtx` package identification

The remaining lines of Figure 6 are ignored on both the first and second pass through the file. However, they still have an important purpose. In addition to the two `latex` passes over the `.dtx` file

for producing documentation, `latex` is also run on the `.ins` file to extract the various package files from the `.dtx` file. The `\generate` call on line 38 of Figure 3 associated the tag `package` with the derived file `mypackage.sty`. Consequently, all lines either beginning with `%<package>` or bracketed between `%<*package>` and `%</package>` are written to `mypackage.sty`. Thus, the code in Figure 6 writes to `mypackage.sty` the `\NeedsTeXFormat` line (line 22), the `\ProvidesPackage` line (line 23), and the optional argument to `\ProvidesPackage` (line 25) — which, as we saw, cleverly also serves as the optional argument to `\ProvidesFile` when generating the package documentation.

`\NeedsTeXFormat` and `\ProvidesPackage` are part of the standard LaTeX 2_ε package-identification mechanism [6]. (Classes use `\ProvidesClass` instead of `\ProvidesPackage`, while other file types use `\ProvidesFile`.) `\NeedsTeXFormat` specifies the earliest date of the LaTeX format itself with which the package is compatible. (From LaTeX, `\show\fmtversion` displays the current format date.) The argument to `\ProvidesPackage` is written to the `.log` file associated with any document that uses the corresponding package.

3.2 Driver code

When producing documentation from a `.dtx` file, the *driver code* is the first block of code that `latex` sees. Figure 7 lists typical driver code. Because `mypackage.ins` does not supply a `\generate` rule for `driver`, placing the driver between `%<*driver>` and `%</driver>` ensures that it will not be processed when generating package files from the `.ins` file. `ltxdoc` is a class designed for typesetting LaTeX documentation; it derives from article but additionally includes the Doc package and defines a few useful commands for documenting classes and packages. One of those commands, `\EnableCrossrefs` (line 30), specifies that the document's index should automatically cross-reference the use of every control sequence (macro or primitive) in the package code. `\CodelineIndex` (line 31) indicates that references to code in the index should point to the corresponding line number instead of to the corresponding page number. `\RecordChanges` (line 32) says to create a file of package changes that can then be incorporated automatically into the documentation in a "Change History" section.

Within the document's body, the `\DocInput` call (line 34 of Figure 7) is the critical line. `\DocInput` tells the Doc package to input the `.dtx` file from within itself. In this second pass through the `.dtx` file, percent characters are not treated as comment characters but are instead ignored. (The sequence "`^^A`" can be used instead of "`%`" to introduce a comment.) After the code is typeset, the `\PrintChanges` call (line 35) typesets a Change History section that informs the reader about the changes that were made to the source code in each revision. `\PrintIndex` (line 36) typesets an index. Finally, the `\fi` in line 39 matches the `\iffalse` in line 18 of Figure 6.

```
28 %<*driver>
29 \documentclass{ltxdoc}
30 \EnableCrossrefs
31 \CodelineIndex
32 \RecordChanges
33 \begin{document}
34   \DocInput{mypackage.dtx}
35   \PrintChanges
36   \PrintIndex
37 \end{document}
38 %</driver>
39 % \fi
```

Figure 7: The `.dtx` driver code

3.3 Code verification

The remainder of this section discusses the part of the `.dtx` file that is processed recursively by `\DocInput`: the documentation proper. In this part of the document, lines beginning with a percent sign are treated as documentation (i.e., the "`%`" is stripped and the result is processed as ordinary LaTeX code). Lines not beginning with a percent sign are both processed as documentation and written to the `.sty` file. This rigmarole is the key to using the same code in both a typeset document and a LaTeX package.

The documentation traditionally begins with a block of code that may be considered slightly anachronistic: a document checksum and a test for unexpected variations in character encoding. The `\CheckSum` call in line 40 of Figure 8 takes an argument representing the total number of backslash characters in the package code (i.e., in lines not beginning with a percent sign). If the tally is correct, Doc outputs

```
**********
* Checksum passed *
**********
```

If the tally is incorrect, Doc issues an error message:

```
! Package doc Error: Checksum not passed
(⟨incorrect⟩<>⟨correct⟩).
```

If the tally is `0`, Doc outputs the correct tally but does not issue an error message:

```
40 % \CheckSum{0}
41 %
42 % \CharacterTable
43 %  {Upper-case    \A\B\C\D\E\F\G\H\I\J\K\L\M\N\O\P\Q\R\S\T\U\V\W\X\Y\Z
44 %   Lower-case    \a\b\c\d\e\f\g\h\i\j\k\l\m\n\o\p\q\r\s\t\u\v\w\x\y\z
45 %   Digits       \0\1\2\3\4\5\6\7\8\9
46 %   Exclamation  \!     Double quote  \"     Hash (number) \#
47 %   Dollar       \$     Percent       \%     Ampersand     \&
48 %   Acute accent \'     Left paren    \(     Right paren   \)
49 %   Asterisk     \*     Plus          \+     Comma         \,
50 %   Minus        \-     Point         \.     Solidus       \/
51 %   Colon        \:     Semicolon     \;     Less than     \<
52 %   Equals       \=     Greater than  \>     Question mark \?
53 %   Commercial at \@     Left bracket  \[     Backslash     \\
54 %   Right bracket \]     Circumflex    \^     Underscore    \_
55 %   Grave accent \`     Left brace    \{     Vertical bar  \|
56 %   Right brace  \}     Tilde         \~}
```

Figure 8: `.dtx` verification code

```
**********************************
* This macro file has no checksum!
* The checksum should be ⟨number⟩!
**********************************
```

It is convenient to specify `\CheckSum{0}` when developing a package and to replace `0` with the correct checksum only when the package is ready to be released.

The character table must appear exactly as shown in Figure 8, lines 42–56. Doc verifies that the character table has not been corrupted and outputs the following success message:

```
****************************
* Character table correct *
****************************
```

If any character differs from that which was expected, Doc issues the following error message:

```
! Package doc Error: Character table
corrupted.
```

3.4 Miscellaneous initialization

Doc can automatically typeset a list of changes made in each version of the package code. It is customary to include an entry for the first version of the code, as shown in line 57 of Figure 9. The first argument is the version number in which the change was made; the second argument is the date the change was made; and, the third argument is a description of the change. If `\changes` is called from within the description of a macro or environment, the change is associated with that macro or environment. Otherwise, the change is categorized as "General".

The `\GetFileInfo` macro (line 59) reads the given file and parses its invocation of `\ProvidesFile` (lines 20 and 25 of Figure 6). `\GetFileInfo` makes

```
57 % \changes{v1.0}{2008/02/18}{Initial version}
58 %
59 % \GetFileInfo{mypackage.dtx}
60 %
61 % \DoNotIndex{\newcommand,\newenvironment}
```

Figure 9: Miscellaneous initialization commands

the date part of `\ProvidesFile`'s argument available as `\filedate`, the version as `\fileversion`, and the package description as `\fileinfo`. The documentation can then use those macros when referring to the package.

One of Doc's most useful features is the automatic production of a code index. Every control sequence defined or used by the package is automatically indexed. However, particularly common control sequences can be distracting and should be omitted from the index. The `\DoNotIndex` macro takes a comma-separated list of control sequences that should not be indexed. (`\DoNotIndex` can be — and usually is — invoked repeatedly, with one line's worth of control sequences at a time.) Typically, TEX primitives such as `\if` / `\else` / `\fi`, `\begingroup` / `\endgroup`, and `\def` / `\edef` / `\gdef` / `\xdef` appear as arguments to `\DoNotIndex`, as do common macros from the LATEX kernel such as `\newcommand` / `\renewcommand` and `\newcounter` / `\newsavebox` / `\newlength`. However, a package that redefines `\newcounter`, for example, probably *would* want to index that control sequence. Producing a good index takes a lot of judgment; think carefully about what someone reading the code might be interested in locating.

```
62 % \title{The \textsf{mypackage} package\thanks{This document
63 %   corresponds to \textsf{mypackage}~\fileversion, dated \filedate.}}
64 % \author{Your Name Here \\ \texttt{you@yournamehere.org}}
65 %
66 % \maketitle
67 %
68 % \section{Introduction}
69 %
70                                                    ⋮
71 %
72 % \section{Usage}
73 %
74                                                    ⋮
75 %
76 % \DescribeMacro{\myMacro}
77 % This macro does nothing.\index{doing nothing|usage}  It is merely an example.  If this were a
78 % real macro, you would put a paragraph here describing what the macro is supposed to do, what
79 % its mandatory and optional arguments are, and so forth.
80 %
81 % \DescribeEnv{myEnv}
82 % This environment does nothing.  It is merely an example.  If this were a real environment, you
83 % would put a paragraph here describing what the environment is supposed to do, what its
84 % mandatory and optional arguments are, and so forth.
```

Figure 10: Prose description of the package

\myMacro	This macro does nothing. It is merely an example. If this were a real macro, you would put a paragraph here describing what the macro is supposed to do, what its mandatory and optional arguments are, and so forth.
myEnv	This environment does nothing. It is merely an example. If this were a real environment, you would put a paragraph here describing what the environment is supposed to do, what its mandatory and optional arguments are, and so forth.

Figure 11: Typeset output of `\DescribeMacro` and `\DescribeEnv`

3.5 User documentation

Package documentation usually begins with a few sections of documentation for the user of the package, as shown in Figure 10. The `\title` specification in lines 62 and 63 is fairly typical in that it sets the package name with `\textsf` and uses `\thanks` to include a footnote with the package's version number and release date. `\date` is often omitted from the title block to distinguish the date the document was printed (`\today`) from the date the package was last modified (`\filedate`).

There is no `\begin{document}` in Figure 10 because the `\begin{document}` already appeared in the `.dtx` driver code (Figure 7); the code in Figure 10 is included through the driver code's invocation of `\DocInput`.

It is common to begin the package documentation with an introductory section that describes what the package does and a usage section that explains how to use the package. The Doc package provides two macros that help give a uniform look to usage sections in package documentation: `\DescribeMacro` and `\DescribeEnv`. Figure 11 displays how Doc typesets lines 76–84 of Figure 10. Notice that the macro or environment name is placed in the margin, where it is easy for a reader to find. Furthermore, the macro/environment name is automatically indexed, with the corresponding page number appearing in the so-called `usage` style (normally italics) in the index. Line 77 of Figure 10 shows how to index arbitrary text in the same style, using `\index{`⟨*term*⟩`|usage}`.

3.6 Package source code

The documented package source code follows the user documentation. Because the average user is not interested in the package's implementation, Doc enables a user to avoid including the package's source code when building the documentation by inserting a call

```
85 % \StopEventually{}
86 %
87 % \section{Implementation}
88                          ⋮
89 % \begin{macro}{\myMacro}
90 % The |\myMacro| macro takes a person's name
91 % and returns the string ``Hello,
92 % \meta{name}''.
93 %    \begin{macrocode}
94 \newcommand{\myMacro}[1]{%
95   Hello, #1\relax
96 }
97 %    \end{macrocode}
98 % \end{macro}
99                          ⋮
100 % \Finale
```

Figure 12: Sample Implementation section

to `\OnlyDescription` into the `.dtx` driver code (between the `\documentclass` and `\begin{document}` lines in Figure 7).

Figure 12 shows how to document the package's source code. The entire code should be bracketed between a call to `\StopEventually` (line 85) and a call to `\Finale` (line 100). `\StopEventually` takes an argument, which is the text for all of the sections that follow the package source code, for example the list of references or the package's copyright and license information. Because the text appears as an argument to a command, certain LaTeX constructs such as `\verb` cannot be used within `\StopEventually`. Unfortunately, ordinary document sections cannot simply be placed after the call to `\Finale` because `\OnlyDescription` would still discard them.

It is good practice to use the standard LaTeX sectioning commands within the implementation section to organize the code and clarify its structure; for example, `\subsection{Initialization macros}`, `\subsection{Helper macros}`, `\subsection{User-callable macros and environments}`, ... One of the beauties of literate programming is that any LaTeX code can be used to document a package: tables, figures, mathematics — whatever is appropriate for explaining how the package works.

Lines 89–98 of Figure 12 give a sample macro definition. A macro definition starts with `\begin{macro}` and the macro name and ends with `\end{macro}`. The Doc package puts the macro name in the margin and includes an index entry with the source-code line number set in the `main` style (normally underlined).

Following the `\begin{macro}` comes the description of what the macro does. The sample description in Figure 12 uses two convenient features of the Doc package. First, "`|`" toggles verbatim mode, which is convenient for macro documentation that would otherwise be cluttered with `\verb` invocations. (This shortcut is in fact provided by the shortvrb package, which is included by Doc.) One caveat is that "`|`" cannot then be used in a `tabular` (or other) environment without first disabling its verbatim properties using `\DeleteShortVerb` and reenabling them afterwards with `\MakeShortVerb`. See the Doc documentation [4] for more information. The second useful Doc feature that appears in Figure 12 is `\meta`, which typesets its argument in italics and within angle brackets, as in "⟨*name*⟩". This is useful for typesetting metasyntactic variables such as ⟨*number*⟩ or ⟨*length*⟩.

The macro source code appears, uncommented, within a `macrocode` environment. Because of some behind-the-scenes trickery in how `macrocode` is handled, there must be *exactly* four spaces between the "`%`" and the `\begin{macrocode}` (as shown in line 93) and between the "`%`" and the `\end{macrocode}` (as shown in line 97). When the documentation is typeset, the lines between `\begin{macrocode}` and `\end{macrocode}` are automatically numbered, and all control sequences encountered are automatically indexed in an unadorned style.

While Figure 12 shows only how to define a macro, environments are defined analogously, using `\begin{environment}` / `\end{environment}` instead of `\begin{macro}` / `\end{macro}` but still using `\begin{macrocode}` / `\end{macrocode}` to delineate blocks of LaTeX code. Definitions of things other than macros and environments — lengths, counters, boxes, etc. — should be placed within a `macro` environment.

The sample macro definition given in Figure 12 is typical of short, simple macros. Longer, more complex macros may benefit from additional commentary within the macro body. In addition, it is common in LaTeX for macros to define other macros. A `.dtx` file can handle both of these cases: Figure 13 shows how. It may be easier to follow Figure 13 by comparing it to the typeset output, shown in Figure 14. Notice how the `\begin{macro}{\othermacro}` is nested within the `\begin{macro}{\complexdef}`. Packages that include a number of short, related definitions (e.g., a set of `\newlength` calls) commonly specify a sequence of `\begin{macro}` calls followed by a description of all the definitions as a whole (e.g., "These lengths represent the jabberwock's width, height, and depth"), followed by a single `macrocode` environment that includes all of the related declarations back-to-back.

```
% \begin{macro}{\complexdef}
% This is a more sophisticated use of the |macro| and |macrocode| environments than was used in
% Figure 12.  Notice the nested |macro| environments and the repeated |macrocode| environments.
% \changes{v1.1}{2008/02/18}{Changed ``Goodbye'' to ``Hello''}
%    \begin{macrocode}
\DeclareRobustCommand{\complexdef}[1]{%
  Hello, #1.
%    \end{macrocode}
% You can insert comments anywhere.  Just call |\end{macrocode}|, enter your text, and start a
% new |\begin{macrocode}|.
%    \begin{macrocode}
  How do you like my macro?%
%    \end{macrocode}
% \begin{macro}{\othermacro}
% Here we have the |\othermacro| macro defined within the |\complexdef| macro.  |macro|
% environments are allowed to nest.
%    \begin{macrocode}
  \gdef\othermacro{#1}%
}
%    \end{macrocode}
% \end{macro}
% \end{macro}
```

Figure 13: A more complex macro definition

`\complexdef` This is a more sophisticated use of the `macro` and `macrocode` environments than was used in Figure 12. Notice the nested `macro` environments and the repeated `macrocode` environments.

```
1 \DeclareRobustCommand{\complexdef[1]{%
2   Hello, #1.
```

You can insert comments anywhere. Just call `\end{macrocode}`, enter your text, and start a new `\begin{macrocode}`.

```
3   How do you like my macro?%
```

`\othermacro` Here we have the `\othermacro` macro defined within the `\complexdef` macro. `macro` environments are allowed to nest.

```
4   \gdef\othermacro{#1}%
5 }
```

Figure 14: Typeset version of Figure 13

3.7 The change history and index sections

The `\changes` call in Figure 13 is not typeset in place but rather schedules a line to be added to the document's Change History section. Because Figure 13's `\changes` call appears within a `macro` environment it is assumed to apply to the surrounding macro instead of to the document as a whole. Figure 15 illustrates how the Change History section may appear in the typeset documentation. If `\changes` appears outside of a `macro` or `environment` environment, the corresponding line in the Change History section lists "General" in place of a macro/environment name.

Running the `.dtx` file through `latex` produces a corresponding `.idx` file if `\CodelineIndex` appears in the driver code and a corresponding `.glo` file if `\RecordChanges` appears in the driver code. The `makeindex` program [2] can be used as shown in

Change History

v1.1

`\complexdef`: Changed "Goodbye" to "Hello" 1

Figure 15: Sample Change History section

```
makeindex -s gind.ist -o ⟨package⟩.ind \
  ⟨package⟩.idx
makeindex -s gglo.ist -o ⟨package⟩.gls \
  ⟨package⟩.glo
```

Figure 16: Commands for producing an index and a change history

Figure 16 to convert the `.idx` file to a typeset index (`.ind`) and the `.glo` file to a typeset change history (`.gls`).

3.8 Additional notes about comments

Program comments should not be written between `\begin{macrocode}` and `\end{macrocode}` because everything within a `macrocode` environment is typeset as code, not as formatted text. (Figure 13 shows the proper way to include inline code comments.) However, it is possible to write comments that are not typeset at all (e.g., for documenting a macro definition that is part of the user documentation, not of the package itself), In fact, all combinations of "visible in the user documentation" and "visible in the `.sty` file" are possible. Table 1 summarizes the techniques for achieving each of these combinations.

Table 1: Comment visibility

Appears in docs	Appears in `.sty`	Mechanism
N	N	`% ^^A` ⟨*comment*⟩

N	Y	`% \iffalse`
		`%%` ⟨*comment*⟩
		`% \fi`

Y	N	`%` ⟨*comment*⟩
Y	Y	`%%` ⟨*comment*⟩

4 Concluding remarks

The advantage of using `.ins` and `.dtx` files is that they encapsulate not only the LaTeX-readable package code but also a human-readable description of the code. Unlike typical, text-only program comments, documentation produced from `.ins` and `.dtx` files can take advantage of all of LaTeX's typesetting power — sectioning, cross-references, figures, tables, mathematics, etc. — coupled with automatic indexing of all macro and environment definitions and uses and automatically pretty-printed code listings. Because of their ability to facilitate the production of immensely readable package documentation, `.ins` and `.dtx` files are the most popular way to distribute LaTeX packages and represent a technique that all LaTeX package writers should strongly consider using for their own packages.

References

[1] Donald E. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, May 1984. Available from `http://www.literateprogramming.com/knuthweb.pdf`.

[2] Leslie Lamport. *MakeIndex: An Index Processor for LaTeX*, February 17, 1987. Available from `http://www.ctan.org/get/indexing/makeindex/doc/makeindex.pdf`.

[3] Frank Mittelbach. The `doc`—option. *TUGboat*, 10(2):245–273, July 1989. Available from `http://www.tug.org/TUGboat/Articles/tb10-2/tb24mitt-doc.pdf`.

[4] Frank Mittelbach. The `doc` and `shortvrb` packages. Distributed as part of LaTeX 2_ε, February 9, 2004. Document source is available from `http://www.ctan.org/get/macros/latex/base/doc.dtx`.

[5] Frank Mittelbach, Denys Duchier, Johannes Braams, Marcin Woliński, and Mark Wooding. The DocStrip program. Distributed as part of LaTeX 2_ε, July 29, 2005. Available from `http://www.ctan.org/get/macros/latex/base/docstrip.dtx`.

[6] The LaTeX3 Project. LaTeX 2_ε for class and package writers. Distributed with LaTeX 2_ε as `clsguide.dvi`, February 15, 2006. Also available from `http://www.ctan.org/get/macros/latex/doc/clsguide.pdf`.

◇ Scott Pakin
4975 S. Sol
Los Alamos, NM 87544-3794
USA
`scott+tb (at) pakin dot org`
`http://www.pakin.org/~scott`

ConTEXt basics for users: Indentations

Aditya Mahajan

Abstract

ConTEXt's indentation mechanism can be a bit confusing. This article explains why ConTEXt indentation works the way it does and how to set up indentation to achieve desired behaviour.

1 Introduction

In plain TEX, controlling indentation is simple: The user sets a value for `\parindent`, and each new paragraph is indented by that value, unless explicitly begun with `\noindent`. Environments can provide a `\noindent` at the end of their definitions, and if the user wants to overrule that, he can add an explicit `\indent` at the beginning of the next paragraph. For the most part, LATEX follows the same convention.

So, understanding indentation in plain TEX and LATEX boils down to this: set a value for `\parindent`, and start a new paragraph (i.e., leave an empty line) whenever you want indentation. For example, LATEX usually does not indent the first line after a sectioning command. If you want to indent the first line after a sectioning command you use the `indentfirst` package (which is part of the required LATEX bundle). If you want to indent the paragraph after an environment, you leave a blank line after the end of the environment; if you do not want to indent after the end of an environment, you do not leave a blank line. It takes a while to get used to, but the rules are easy to remember and eventually you do not need to even think about indentation; it becomes a matter of habit.

Indentations in ConTEXt are a bit different; and sometimes difficult to understand. In this article I hope to explain how ConTEXt does indentations. First, let's understand why ConTEXt does indentations differently; why does it not simply follow the time-tested approach of plain TEX and LATEX? The way I understand it, the reason is that Hans Hagen, the author of ConTEXt, prefers *spaced out* markup — surrounding each environment by empty lines — which makes it easy to see where an environment starts and ends while reading the source file. However, this style means that the "indentation after empty lines" paradigm of plain TEX and LATEX cannot be used for indentations. So, ConTEXt provides an alternative. As with other things in ConTEXt, this alternative is consistent and easy to configure; but if you are used to other TEX formats it takes some time to get comfortable with it.

2 The basics

Indentation involves two things: when to indent, and how much to indent. In ConTEXt, these can be specified using `\setupindenting[...]`. There are two types of keys for this command:

1. To specify *when to indent*: `never` or `always`, (equivalently, `no` or `yes`), `odd` or `even`, and `first` or `next`.
2. To specify *how much to indent*: `none`, `small`, `medium`, `big`, or a specific dimension. `small` corresponds to 1em, `medium` to 1.5em, and `big` to 2em.

Suppose we want to indent all paragraphs by 20pt, which is the convention followed by this journal: we can use `\setupindenting[20pt, yes]`. This is what one typically uses in a document. The other keys are needed only for special cases (like typesetting quotations and verses), and we will not talk about them in this article.

`\setupindenting` does not take care of indenting after environments, such as ConTEXt's itemizes (approximately LATEX `itemize` and `enumerate`), enumerations (≈ LATEX `theorem`), definitions (≈ LATEX `subparagraph`), formulas and floats. It also does not take care of indenting after heads such as chapters, sections, and subsections.

The setup command of these environments provides an `indentnext` key to configure the indentation behaviour after the environment. The `indentnext` key can take one of three values: `yes`, `no`, and `auto`. If `indentnext=yes` then the paragraph after the end of the environment is always indented; if `indentnext=no` then the next paragraph is never indented; if `indentnext=auto` then the next paragraph is indented only if there is a blank space after the environment. Setting `indentnext=auto` is equivalent to the default plain TEX and LATEX behaviour.

Let us provide a couple of examples. Suppose we do not want paragraphs after itemize to be indented; then we can say

```
\setupitemgroup[itemize][indentnext=no]
```

If we want paragraphs after section heads to be indented, we can say

```
\setuphead[section][indentnext=yes]
```

If we want the paragraphs after all sectioning heads to be indented, we can say

```
\setupheads[indentnext=yes]
```

If we want paragraphs after formulas to be indented only if we leave a blank space after them, we can say

```
\setupformulas[indentnext=auto]
```

There is one case that is not taken care of by `\setupindenting` and the `indentnext` key: indentation of paragraphs inside multi-paragraph environments such as itemizes, descriptions, and enumerations. By default ConTEXt does not indent such paragraphs. The setup commands of these environments provides an `indenting` key to configure the indentation behaviour of paragraphs inside these environments. This key takes the same values as the arguments of `\setupindenting` command. For example, if we set

```
\setupitemgroup[itemize][indenting={40pt,yes}]
```

then the following (`|-|` in ConTEXt indicates a hyphen after which further hyphenation is allowed; see `http://www.logosrl.it/context/modules/current/singles/lang-mis_ebook.pdf` for more):

```
\startitemize
\item This is an example of a multi|-|paragraph
      item inside an itemize environment.

      This second paragraph is indented by 40pt
      (double the normal indentation).
\stopitemize
```

gives

- This is an example of a multi-paragraph item inside an itemize environment.

  This second paragraph is indented by 40pt (double the normal indentation).

---

3 Manual indentation

No matter how careful we are with all the settings, there are some cases which cannot be taken care of by automatic indentation, and we have to tell ConTEXt how to indent. Plain TEX (and LATEX) provide the `\indent` and `\noindent` commands for explicitly indenting and preventing indenting of a paragraph. These commands are defined in ConTEXt but are not hooked into the ConTEXt indentation mechanism. Instead, ConTEXt provides `\indentation` and `\noindentation` which achieve the same effect.

Notice that in this article, paragraphs after itemize are not indented. This is because the itemize environment has been set up (according to the general style of the journal) as

```
\setupitemgroup [itemize] [indentnext=no]
```

Now, if we want to indent after an itemize, starting the next paragraph with `\indent` does not work; to get indentation we have to say `\indentation`. For example,

```
\startitemize[n]
\item A dummy list
\item To check \tex{indent}
\stopitemize

\indent This paragraph is not indented. The
\tex{indent} command does not work here.

\startitemize[n]
\item Another dummy list
\item To check \tex{indentation}
\stopitemize

\indentation This paragraph will be indented
because we used \tex{indentation} instead.
```

gives

---

1. A dummy list
2. To check `\indent`

This paragraph is not indented. The `\indent` command does not work here.

1. Another dummy list
2. To check `\indentation`

This paragraph will be indented because we used `\indentation` instead.

---

4 Beware of typos

ConTEXt defines two more commands not commonly used: `\indenting` and `\noindenting`. `\indenting` is similar to `\setupindenting` and is provided for backward compatibility; `\noindenting` is equivalent to `\setupindenting[no]`. Unfortunately, these commands sound very similar to `\indentation` and `\noindentation`, thus can be easily used by mistake. If you happen to write `\noindenting` instead of `\noindentation` in a document, it can take a while to debug. I have been bitten by this mistake once too often so I have invented a mnemonic to avoid it:

shun (-tion) the -ing—use indentation

I admit, this is not a clever mnemonic, but it has saved me quite a few trips to the manual.

5 Conclusion

This article explained how indentation works in ConTEXt. By default, ConTEXt does not enable *any* indentation; the user is expected to set up indentation as he wants. I hope this article has helped to illustrate how to set up indentation for different environments.

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

MetaPost

Kanji-Sudokus: Integrating Chinese and graphics

Denis Roegel

1 Introduction

Recently, I had the need to get my hands on Werner Lemberg's excellent CJK package, for a talk on the Chinese calendar in which I wanted to use METAPOST figures with Chinese labels. This worked almost seamlessly.

Actually, this isn't quite true, but CJK is better and better integrated into TEX Live these days, and writing in Chinese, Japanese or Korean has become pretty much mundane with an up-to-date TEX environment. This wasn't so even a year ago. Nowadays, you still need to install various Linux (say) packages, and one is likely to run into trouble because some crucial element is missing. For instance, on my latest Ubuntu, the TEX Live setup wasn't complete and I was missing some Korean fonts I needed. By the time you read this, the problem may have been solved already.

To sum up, with the latest TEX Live 2007 setup, and perhaps a few additional Linux packages, as well as the latest Emacs, you are all set for typesetting beautiful CJK documents! Typesetting CJK has even become easier, because one can now write almost everything in UTF-8, without a need to postprocess the input file with Emacs macros (this procedure used to output a `.cjk` file which could only then be processed by LATEX). Now, the file you write is the file you process, and processing has become faster.

For METAPOST figures, the matter was also made easier. Up until recently, when including Chinese in METAPOST, one had first to produce a `.cjk` file, which could unfortunately not be processed by METAPOST. The `.cjk` file had to be slightly altered first, because the Emacs macros were not aware of the METAPOST format. This could have been corrected within the Emacs macros, but in fact, since the conversion to the `.cjk` file is now mostly an old story, the processing problems have also vanished. So, my advice is not only to switch to the latest TEX Live and Linux, but also to write CJK in UTF-8. It works!

2 A small example

I will illustrate the integration of Chinese and METAPOST with a small example. I will draw a Sudoku grid, not with Hindu-Arabic numerals, but with Chinese numerals. These numerals are 一 (1), 二 (2), 三 (3), 四 (4), 五 (5), 六 (6), 七 (7), 八 (8), and 九 (9).

A typical Sudoku problem reads as follows (this example from Wikipedia):

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

2.1 The grid

The whole Sudoku problem can be drawn as follows in METAPOST:

```
beginfig(1);
  string sol[];
  drawgrid(1.5pt,.5pt);
  % first row at the bottom
  % last row at the top
  sol1="000080079";sol2="000419005";
  sol3="060000280";sol4="700020006";
  sol5="400803001";sol6="800060003";
  sol7="098000060";sol8="600195000";
  sol9="530070000";
  fillgrid(sol)(false);
endfig;
```

The `drawgrid` macro is straightforward; it produces the horizontal and vertical lines (with `u` being for instance equal to 1 cm):

```
def drawgrid(expr tha,thb)=
  pickup pencircle scaled thb;
  for i=0 upto 9:
    draw (i*u,0)--(i*u,9u);
    draw (0,i*u)--(9u,i*u);
  endfor;
  pickup pencircle scaled tha;
  for i=0 upto 3:
    draw (3i*u,0)--(3i*u,9u);
    draw (0,3i*u)--(9u,3i*u);
  endfor;
enddef;
```

2.2 Filling the grid

In order to fill the grid, we need to access the position (i, j), where i is the column and j is the row, all numbered from 1 at the bottom-left cell. We therefore define the following macro, which takes i, j and a label that it centers in the middle of the cell. In our case, the label is scaled 200%, but how much you scale depends on the dimensions of the frame and on the base size of the font.

```
def pos(expr i,j,l)=
  label(l scaled 2,((i-.5)*u,(j-.5)*u));
enddef;
```

2.3 Cell entries

In order to put, say, the value 3 at position (2,9), we could write

```
pos(2,9,btex 3 etex);
```

However, we want to be more general and draw our figures from the string array `sol`. That way, some program can produce a problem and/or a solution, and the problem can easily be plugged into our macros. So, instead, we could write

```
pos(2,9,TEX(s));
```

where `s` is a string provided to the `TEX` macro. The latter is defined by loading the package `TEX`:

```
input TEX;
```

This, however, is not very efficient, because it will call TEX up to 81 times. And besides, it won't take care of Chinese numerals when we need them. So, we are looking for something more flexible. The `latexmp` METAPOST package will suit our needs. The previous label is now obtained with

```
pos(2,9,textext(s));
```

One of the advantages of the `latexmp` package is that it will require only two runs of LaTeX, and not one for every label. This package also makes it easy to load LaTeX packages, in particular for Chinese. So, our METAPOST file will begin as follows:

```
input latexmp;

setupLaTeXMP(class="article",
  packages="CJKutf8",
  preamble=(
     "\let\N\newcommand"
    &"\N\0{}\N\1{一}\N\2{二}\N\3{三}"
    &"\N\4{四}\N\5{五}\N\6{六}\N\7{七}"
    &"\N\8{八}\N\9{九}"
    &"\AtBeginDocument{"
    &   "\begin{CJK}{UTF8}{bsmi}}"
    &"\AtEndDocument{\end{CJK}}"));
```

This preamble will load the `CJKutf8` package, which is what we need for UTF-8 input. It then defines the commands `\0` (for void), `\1` (Chinese numeral 1), `\2` (Chinese numeral 2), etc., up to `\9`.

Then, we start an appropriate (for these characters) CJK environment at the `\begin{document}` hook:

```
\AtBeginDocument{
  \begin{CJK}{UTF8}{bsmi}}
```

and we close the environment at the end of the document:

```
\AtEndDocument{\end{CJK}}
```

2.4 Putting all the pieces together

We now have a definition of cell values, we can draw a grid, and we have macros for Chinese numerals. What's next? Well, we want to be able to do two kinds of things: draw problems, and draw solutions. For our purposes, a problem is merely an array of cell values with some cells being equal to 0. These 0s will be displayed as empty cells. In addition to this switch, we want to display the non-void values either with Hindu-Arabic numerals or with Chinese numerals.

Our coding of cell values makes this rather easy, because we will use (for instance) `4` for the Hindu-Arabic numeral, and `\4` for the Chinese numeral. So, care must be taken of this additional `\` when needed. The special case of 0 must also be considered, because the Hindu-Arabic numeral 0 must not be displayed, whereas the Chinese `\0` can be displayed, since it is void.

The '0' switch is handled with the `zerospace` macro. This macro takes a character `s` and replaces this character by a space only when it is 0 and when the output uses Hindu-Arabic numerals.

```
def zerospace(expr chinese,s)=
  if not chinese and (s="0"): " "
  else: s fi
enddef;
```

Finally, filling the grid is done with `fillgrid`. The first parameter is the name of the string array and the second parameter is a switch for Chinese or Hindu-Arabic numerals. `substring` is used to isolate the character of interest.

```
def fillgrid(text grid)(expr chinese)=
  for i=1 upto 9:for j=1 upto 9:
    pos(j,i,textext(if chinese: "\" & fi
         zerospace(chinese,
           substring(j-1,j) of grid[i])));
  endfor;endfor;
enddef;
```

The result is then as follows for the problem and the solution, with Chinese numerals:

五	三			七				
六			一	九	五			
	九	八					六	
八				六				三
四			八		三			一
七				二				六
	六					二	八	
			四	一	九			五
				八			七	九

五	三	四	六	七	八	九	一	二
六	七	二	一	九	五	三	四	八
一	九	八	三	四	二	五	六	七
八	五	九	七	六	一	四	二	三
四	二	六	八	五	三	七	九	一
七	一	三	九	二	四	八	五	六
九	六	一	五	三	七	二	八	四
二	八	七	四	一	九	六	三	五
三	四	五	二	八	六	一	七	九

3 Conclusion

This example demonstrates how straightforward the integration of Chinese and METAPOST has become. What remains to be done is to link these macros with a general problem solving algorithm for Sudokus.

◇ Denis Roegel
LORIA, BP 239
54506 Vandœuvre-lès-Nancy
FRANCE
roegel (at) loria dot fr
http://www.loria.fr/~roegel

Hints & Tricks

Interesting loops and iterations — second helping

Paweł Jackowski

Abstract

Where on earth does a programmer have to implement a loop construct himself? In TEX! TEX as a programming language is akin only to itself. Its interesting feature, rarely to be found among programming languages, is the lack of a built-in loop construct. However, thanks to TEX dealing perfectly well with recursive definitions and its ability to check conditions there are no obstacles to defining DIY loops. It has been done by Donald Knuth in plain TEX, extended by Alois Kabelschacht, Kees van der Laan, Marcin Woliński and many others and used by every practicing TEXie. This article sums up what every TEXie should know about loops. We will not shy away from dirty tricks which users need not know about.

Taking on `\loop`

Let's yet again review the traditional `plain`-ish loop definition ([1], p. 352):

```
\def\loop#1\repeat{%
 \def\body{#1}\iterate}
\def\iterate{%
 \body \let\next\iterate
 \else \let\next\relax\fi \next}
\let\repeat=\fi
```

The definition is pretty readable and understandable thanks to the supporting macros suggestively named `\next` and `\body`. However, an unnecessary assignment is performed at every iteration. This assignment gives a meaning to the `\next` instruction as well as the `\body` instruction which, in principle, should not be used anywhere else. As the instructions are hidden from the user a name conflict can easily arise.

There are several well-known enhancements of this traditional definition which use `\expandafter` instead of the scratch `\next` macro. For example, from [2]:

```
\def\loop#1\repeat{%
 \def\body{#1}\iterate}
\def\iterate{%
 \body\expandafter\iterate\fi}
```

or even simpler, from [6]:

```
\def\loop#1\repeat{%
 \def\iterate{%
  #1\expandafter\iterate\fi}%
 \iterate}
```

In the first case we are getting rid of the superfluous definition of `\next` and in the second also of the definition of `\body`. In yet another construction (for an extended description see [3]) the whole contents of the loop is executed outside of a conditional block `\if...\fi`:

```
\def\loop#1\repeat{%
 \def\body{#1}\iterate}
\def\iterate{%
 \body\else\etareti\fi\iterate}
\def\etareti\fi\iterate{\fi}
```

A summary of these and other solutions may be found in [4].

A loop in a loop

The above constructs, though correct and elegant, do not allow loop nesting. In each of them the first operation remembers the content of the loop in an instruction. Embedding it would cause a conflict for the inner and outer loops.

Is there a way out? Yes. At the cost of slightly slowing the loop one may use a macro parameter instead of a definition. For example, instead of repeating the `\body` at every `\iterate`, we can set the repeated code fragment as an argument of the `\iterate` instruction. For convenience the `\long` prefix is used, to enable the use of `\par` within the loop. We also define the `\gobbleone` macro, which is called just before processing leaves the loop and gobbles the superfluous argument just after the `\fi` ending the conditional.

```
\long\def\loop#1\repeat{%
 \iterate\gobbleone{#1}}
\long\def\iterate\gobbleone#1{%
 #1\expandafter\iterate\fi
 \gobbleone{#1}}
\long\def\gobbleone#1{}
```

The `\gobbleone` definition plays a second role — it delimits the `\iterate` macro (i.e., is a *macro delimiter*). When the `\iterate` instruction is being

---

This is a translation of the article "Ciekawe pętle i iteracje na drugą nóżę", which first appeared in *Biuletyn GUST* nr 22 (2005), 3–6. Reprinted by permission. Translation by Jerzy Ludwichowski.

executed, the immediately following `\gobbleone` is swallowed as an unused fragment of the parameter. At the end of the loop `\iterate` is skipped, but `\gobbleone` swallows the loop content argument.

This loop might be used like the traditional form, the difference being that it can be nested, as shown in the following example:

```
\count100=9
\loop{\count101=65 % ASCII ‘A’
 \advance\count100 by-1
 \ifnum\count100>0
 \leavevmode\loop
  \char\count101 \the\count100
  \advance\count101 by1
  \ifnum\count101<73 \space
\repeat\par
}\repeat
```

The code produces something akin to a chess field. The row elements are typeset by the inner loop and the rows are produced by the outer loop.

```
A8 B8 C8 D8 E8 F8 G8 H8
A7 B7 C7 D7 E7 F7 G7 H7
A6 B6 C6 D6 E6 F6 G6 H6
A5 B5 C5 D5 E5 F5 G5 H5
A4 B4 C4 D4 E4 F4 G4 H4
A3 B3 C3 D3 E3 F3 G3 H3
A2 B2 C2 D2 E2 F2 G2 H2
A1 B1 C1 D1 E1 F1 G1 H1
```

One should note the use of grouping in the outer loop block:

```
\loop{...\loop...\repeat...}\repeat
```

This group affects only the scope of the argument reading. The content of the outer loop is not executed within the group. Thanks to this, the outer loop can use the assignments done in the inner loop. Grouping is necessary — without it TEX would cease reading the outer loop just after seeing the first `\repeat`.

Let it resolve

The capabilities of TEX do not end in incrementing and checking the counter value. Moreover, TEX iterations are not restricted to `\loop...\repeat` constructions. Often there is a need to execute some procedure for each token of a group, in a context where assignments cannot be used (e.g., when creating definitions with `\edef`, `\xdef`, inside `\write`-s, `\special`-s and `\mark`-s). Here it is worth citing the beautiful-in-its-simplicity macro `\fifo`, described in more detail in [3]:

```
\def\fifo#1{\ifx\ofif#1\ofif\fi
 \process#1\fifo}
\def\ofif#1\fifo{\fi}
```

In the example below `\fifo` is used to create a crib sheet of codes of some diacritics:

```
\def\process#1{(#1 -> \number‘#1)}
\immediate\message
 {\fifo áéó\ofif}
```

Counting of iterations is replaced here by executing the `\process` instruction for consecutive arguments. At the start of each iteration `\ifx` checks if the just-found argument is the `\ofif` token. The latter both delimits the token list and is a macro ending the condition executed after the last iteration.

Number games

No one needs convincing that expandable macros (without assignments) are more convenient. But how can assignments be avoided in loops operating on numbers? The most typical use of loops is repeating code some defined number of times. The previously shown `\loop...\repeat` constructs achieve this by iteratively incrementing or decrementing a counter, but this requires assignments.

The task is not hopeless, however. As the preceding example shows, the `\number` instruction expands “on the fly” any TEX representation of a number into its decimal form. In the basic version of TEX every arithmetic operation requires an assignment. To the rescue comes ε-TEX, which offers several convenient operations that allow dodging inconvenient assignments. The `\numexpr` instruction will serve as an example. It executes, in an expandable way, the basic operations on numbers (addition, subtraction, multiplication and division).

Let us use `\numexpr` to build a `\replicate` macro which repeats an arbitrary piece of code a given number of times. The first parameter is the number of repetitions, the second is the content of the loop.

```
\long\def\replicate#1#2{%
 \ifnum\numexpr#1>0
  #2\replicate{#1-1}{#2}\fi}
```

The loop starts with the check for the counter being positive, i.e., if the repetition should be executed. If so, then the contents of the loop, given as the second parameter, is executed and then a recursive call is being made to the `\replicate` procedure with the counter subtracted by 1 and the second parameter unchanged.

This construct suffers from two serious drawbacks. First, each repetition is executed within accumulating \ifnum...\fi blocks, which threatens catastrophe if a large number of iterations is required. Second, the length of the first parameter of the macro is increased by two at each turn of the loop, hence during the check of the counter value TEX must each time evaluate an ever longer expression of the form \numexpr100-1-1-1....

Therefore let us try to modify the \replicate macro so as to execute each repetition outside of the \ifnum...\fi condition and give the parameter representing the counter a more elegant form.

```
\long\def\replicate#1#2{%
 \ifnum\numexpr#1>0
  #2\expandafter\replicate\expandafter
  {\number\numexpr#1-1\expandafter}%
 \else
  \expandafter\gobbleone
 \fi{#2}}
```

Again we start by checking if the loop counter is positive, i.e., if the repetition should be executed. If so, the content of the loop (the second parameter) is processed, after which \expandafter in connection with \number\numexpr decrements the counter by one and enters the \replicate procedure with the new value of the counter. The second parameter to the \replicate procedure is passed on unchanged and immediately follows the \fi instruction ending the conditional. When the counter reaches 0 (or if we mischievously start the loop with the parameter being not greater than 0), \expandafter kills the remaining \fi after which the already described \gobbleone procedure swallows the superfluous parameter.

We use here the previously mentioned beneficial feature of the \number instruction which causes the macros following it to be expanded completely, i.e., until the decimal representation of the number is produced. During the expansion of \numexpr, \expandafter is executed which as if in passing (during the number expansion!) causes the loop condition block to disappear. TEX then "notices" that the expression cannot be expanded further and returns to the \replicate instruction. The latter is executed with the numerical argument in decimal representation and the second argument being the immutable loop content. This happens outside of the conditional block.

Here is an example of \replicate in a context in which the traditional TEX loop with assignment would fail. The \replicate macro is expandable and can be nested.

```
\immediate\message
 {\replicate{100+1}
  {I will be using eTeX%
    \replicate{3}{!} }}
```

Let's move on to a more complicated example. We will try to define a \fixed macro which puts the digit 0 in front of all other digits in such a way as to complement the number to a set length. For example,

```
\fixed{4}{12}
```

should expand to 0012. We begin by defining a helper macro to "measure" the length of a sequence.

```
\long\def\abacus#1{\addabacus#10}
\long\def\addabacus#1#2#3{%
 \ifx#3#1#2\else
  \expandafter\addabacus
  \expandafter#1\expandafter
  {\number\numexpr#2+1\expandafter}%
 \fi}
```

The \abacus macro (from Latin: a calculating tool) counts tokens appearing between a pair of two other tokens.

```
\count100=\abacus|Llanfairpwllgwyngyll%
gogerychwyrndrobwllllantysiliogogogoch|
\edef\numofletters{%
 \abacus\relax Antidisestablish%
  mentarianism\relax}
```

At each turn of the loop the macro tests if the upcoming token is the delimiting token of the measured sequence. If not, the macro in the already described manner increments the counter by 1 and moves on to the next iteration. If yes, it simply returns its counter which is the number of tokens between the freely chosen delimiters.

Now, we can use \replicate and \abacus to define a macro to pad the sequence with a chosen character to a given length.

```
\def\fixedprefix#1#2#3{%
 \expandafter\replicate\expandafter
 {\number
   \numexpr#1-\abacus\relax#2\relax}
 {#3}#2}
```

If we now write

```
\edef\test{\fixedprefix{4}{ab}{*}}
```

the \test instruction will be assigned the value of **ab. It remains to construct a specialized version of the \fixedprefix macro which will format numbers in such a way that they will have the specified number of digits by prepending with zeros if needed. Because the \fixed macro should operate on numbers, the first operation to be performed is

to expand the argument to a sequence of digits only. We know this trick already.

```
\def\fixed#1#2{%
 \expandafter\fixedzero\expandafter
  {\number\numexpr#1\expandafter}%
 \expandafter{\number\numexpr#2}}
\def\fixedzero#1#2{%
 \fixedprefix{#1}{#2}{0}}
```

We also know that TEX expands numbers tirelessly until the end. We also know that it has no problems with long sequences of tokens swallowed as arguments. The `\rnum` (*read number*) macro presented below exploits both TEXniques of iteration to read numbers in different notations, from binary to hexadecimal.

```
\def\rnum#1#2{\dornum{#1}{0}#2\relax}
\def\dornum#1#2#3{\ifx#3\relax#2\else
  \expandafter\dornum\expandafter
  {\number
    \numexpr#1\expandafter}\expandafter
  {\number
    \numexpr#1*#2+"#3\expandafter}%
 \fi}
```

We thus taught TEX to understand what, e.g., 1 000 000 000 000 means in binary notation:

```
\count100=\rnum{2}{1000000000000}
```

The reader may have noticed the character '`"`' which was used in the second-to-last line of the `\dornum` macro. As is known, for TEX this means: "read the digits as hexadecimal". Without it, TEX would not properly understand the digits `A` through `F`.

For dessert we propose the `\xnum` macro, which does the opposite of `\rnum`. It rewrites decimal numbers into other notations, from binary to hexadecimal and, of course does this in a fully expandable way. If the reader made it to this point, he should have no problems in understanding the following code. However, those who do not use ε-TEX deserve two explanations.

1. If during calculating `\numexpr` ε-TEX encounters the `\relax` token, it immediately stops reading the expression and `\relax` disappears without a trace.
2. If `\numexpr` contains non-integer division, the result will be rounded, unlike in TEX, where it will be truncated to the integer part.

More about ε-TEX constructions is in [5].

```
\def\hexdigit#1{%
 \expandafter\hexdigits
 \number\numexpr#1\relax\relax}
\def\hexdigits#1\relax
{\ifcase#1
  0\or1\or2\or3\or4\or5\or
  6\or7\or8\or9\or A\or
  B\or C\or D\or E\or F\fi}

\def\xnum#1#2{%
 \expandafter\doxnum\expandafter
 {\number
  \numexpr#1\expandafter}\expandafter
 {\number\numexpr#2}}
\def\doxnum#1#2{%
 \ifcase
  \ifnum#2<\numexpr#2/#1*#1\relax
   0 \else1 \fi
 \expandafter\doxnumdown\or
 \expandafter\doxnumup\fi
 {#1}{#2}}
\def\doxnumdown#1#2{%
 \ifnum#1>#2 \else
  \expandafter\doxnum\expandafter
  {\number#1\expandafter}\expandafter
  {\number\numexpr#2/#1-1\expandafter}\fi
 \hexdigit{#2-(#2/#1-1)*#1}}
\def\doxnumup#1#2{%
 \ifnum#1>#2 \else
  \expandafter\doxnum\expandafter
  {\number#1\expandafter}\expandafter
  {\number\numexpr#2/#1\expandafter}\fi
 \hexdigit{#2-#2/#1*#1}}
% test
\count100=\rnum{2}{1000000000}
\immediate\message
 {\xnum{16}{\count100}}
```

Bibliography

[1] Donald E. Knuth: *The TEXbook* (1990), Addison-Wesley.

[2] Alois Kabelschacht: `\expandafter` vs. `\let` and `\def` in conditionals and a generalization of plain's `\loop`. *TUGboat*, Volume 8 (1987), No. 2, 184–185.

[3] Kees van der Laan: FIFO and LIFO sing the BLUes. *Biuletyn* GUST, nr 4 (1992), 20–26.

[4] Marcin Woliński: O pewnych konstrukcjach warunkowych i iteracyjnych [On some conditional and iterative constructs]. *Biuletyn* GUST, nr 7 (1996), 5–9.

[5] Peter Breitenlohner: The ε-TEX Manual, Version 2, February 1998, 9.

[6] Victor Eijkhout: The bag of tricks. *TUGboat*, Volume 21 (2000), No. 1, 91.

◇ Paweł Jackowski
GUST
P dot Jackowski (at) gust dot org dot pl

Glisterings

Peter Wilson

> The raging waves doth belching upwardcast
> The wretched wrackes that round about doe fleete,
> The silken sayles and glistering golden Mast,
> Lies all to torne and trodden under feete.
>
> *The Ship of safegarde*, Barnabe Googe

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.

The main topic this time is macro definition. Questions about this, particularly with respect to LaTeX, are fairly regular on the `comp.text.tex` newsgroup. But first...

> The lines are fallen unto me in pleasant places; yea I have a goodly heritage.
>
> *Psalm 16, verse 6*

1 More on paragraphs

Donald Knuth sent me a version of the following code saying

> 'I've found this macro to be useful for checking out a `\parshape` specification before cluttering it up with actual text.'

```
% parshape.tex, featuring a possibly useful
%    macro by Don Knuth, April 2007
% \parshapetest{n} will typeset n lines of
%    horizontal rules using the current
%    paragraph shape (as specified by
%    \hangindent, \hangafter, \parshape, or
%    none of the above)
\def\parshapetest#1{%
  \leavevmode%% DEK originally had \indent here
  \count255=1 \loop
    \ifnum\count255<#1
    \null\leaders\hrule\hfil\null\break
    \advance\count255 by 1   \repeat
  \null\leaders\hrule\hfil\hskip-\parfillskip
  \null\par}
```

Unfortunately it was too late to incorporate it into the last column [8] which was about how to typeset variously shaped paragraphs. It was doubly unfortunate because when I tried using `\parshapetest` on some of the examples I found that I had misunderstood some aspects of paragraph setting.

`\parshapetest{`⟨*num*⟩`}` draws ⟨*num*⟩ lines according to the current paragraph shape specification, which doesn't sound very exciting but can save a lot of fiddling trying to get the right number of words for a more realistic trial layout.

For instance, I tried this example from [8]

```
\begingroup
\hangindent=3pc \hangafter=-2
\parshapetest{4}
\endgroup
```

which, to my surprise, resulted in:

What I hadn't realised was that even with specifying `\hangindent` and `\hangafter`, `\parindent` was applied to the first line of the shaped paragraph. The effect that I had expected is obtained as below.

```
\begingroup
\parindent=0pt
\hangindent=3pc \hangafter=-2
\parshapetest{4}
\endgroup
```

which results in:

Following this I tried the `\hangfrom` example from the same column which demonstrated a hanging paragraph. The macro was defined as:

```
\newcommand*{\hangfrom}[1]{%
  \setbox\@tempboxa\hbox{{#1}}%
  \hangindent \wd\@tempboxa
  \noindent\box\@tempboxa}
```

And a demonstration is:

```
\hangfrom{$\Rightarrow$\space}
\parshapetest{3}
```

⇒

Here's a more interesting paragraph shape, and the result of testing it:

```
\newdimen\zide
  \zide=\baselineskip
\newcommand*{\aparshape}{%
\parshape=10 0pt 10\zide % 1
             0pt 10\zide % 2
             9\zide \zide % 3
             8\zide \zide % 4
             6\zide \zide % 5
             4\zide \zide % 6
             2\zide \zide % 7
```

```
            \zide \zide % 8
            0pt 10\zide % 9
            0pt 10\zide % 10
}
\aparshape
\noindent\parshapetest{10}
```

Try using this paragraph shape with a text of 76 'z' characters with spaces between each, like this:

```
\aparshape
\noindent
z z z z z z z z z z z z z z z z z z z z
etc
```

Replying to a request on the `comp.text.tex` newsgroup by Stephen Moye, Paul Vojta [7] posted the following code[1] for setting the first line of a paragraph flushleft, the next centered and the final line flushright.

```
\newcommand*{\leftcenterright}{%
  \leftskip=0pt plus 1fil
  \rightskip=0pt plus 1fil
  \parfillskip=0pt plus -1fil
  \parindent=0pt
  \everypar={\hskip0pt plus -1fil}}
```

This should either be used in a group, or you can use the following macro to return to the regular paragraph style, where you have previously specified `\myparindent` as the normal value of `\parindent`.

```
\newcommand*{\regularpar}{%
  \leftskip=0pt plus 0pt minus 0pt
  \rightskip=\leftskip
  \parfillskip=0pt plus 1fil
  \parindent=\myparindent
  \everypar{}}
```

Following is an example of a `\leftcenterright` paragraph, typeset from:

```
\leftcenterright
First line \\ Second line \break
Third line \break Last line \par
\regularpar
```

First line
Second line
Third line
Last line

[1] For convenience I have put Paul's code into a macro.

> Who will change old lamps for new? . . . new lamps for old ones?
>
> *Arabian Nights: The History of Aladdin*

2 LaTeX's defining triumvirate

The macro provided by LaTeX for defining new commands is somewhat simpler than the TeX macro upon which it is based. This is the LaTeX one:
`\newcommand`{⟨*cmd*⟩}`[`⟨*num*⟩`][`⟨*arg1*⟩`]`{⟨*defn*⟩}
where ⟨*cmd*⟩ is the name, including the backslash (e.g., `\amacro`), of the new macro being defined and ⟨*defn*⟩ is the definition of the new macro, which may be simply some text to be typeset or something very complex. The optional ⟨*num*⟩ argument specifies the number of arguments that the new macro will take; if given this must be at least one and at most nine. The new macro will take an optional argument if ⟨*arg1*⟩ is given, where ⟨*arg1*⟩ is the default value of the first argument. The macro resulting from `\newcommand` is, in TeX terms, a *long* macro, meaning that an argument may consist of more than one paragraph or, equivalently, include a `\par`. There is also a star form of the command (`\newcommand*`) which creates a macro where paragraph(s) are not allowed in an argument to the new macro. If ⟨*cmd*⟩ has been defined previously LaTeX will give an error message.

The LaTeX macro
`\renewcommand`{⟨*cmd*⟩}`[`⟨*num*⟩`][`⟨*arg1*⟩`]`{⟨*defn*⟩}
and its companion `\renewcommand*`, are similar to `\newcommand` except that they change the definition of ⟨*cmd*⟩, which *must* have been defined earlier, otherwise LaTeX will complain.

The third member of LaTeX's macro definition macros is:
`\providecommand`{⟨*cmd*⟩}`[`⟨*num*⟩`][`⟨*arg1*⟩`]`{⟨*defn*⟩}
which acts like `\newcommand` if ⟨*cmd*⟩ has not been defined, otherwise it silently does nothing. Again, there is a starred version of the command.

If you want to make sure that your definition for ⟨*cmd*⟩ is used regardless of whether or not it has been defined before you can do this:

```
% ensure \amacro is defined
\providecommand{\amacro}{}
% change the definition
\renewcommand{\amacro}...
```

Within the ⟨*defn*⟩ argument to the macros the use of the first argument, if any, is denoted by `#1`, the second by `#2`, and so on up to the ninth which is denoted by `#9`. The arguments can be used as many times as needed and in any order.

A question that pops up now and then on the `comp.text.tex` newsgroup is how to define a macro that takes more than nine arguments. The answer is

to split it up into two or more macros each of which handles a portion of the required number. For, say, 11 arguments:

```
\newcommand{\xiargs}[9]{%
  % 9 args used here then
  \xtrargs}
\newcommand{\xtrargs}[2]{%
  % use last 2 args here
  % #1 and #2 are the apparent 10th & 11th args
}
```

The user calls `\xiargs` with the 11 arguments, and `\xiargs` processes the first 9 of these. It then calls `\xtrargs`, which is effectively hidden from view, to process the remaining 2 arguments. If you need to use, say, the 4th argument within `\xtrargs` this can be easily accomplished:

```
\newcommand{\xiargs}[9]{%
  % 9 args used here then
  \xtrargs{#4}}
\newcommand{\xtrargs}[3]{%
  % #1 here is #4 from \xiargs and
  % #2 and #3 are the apparent 10th & 11th args
}
```

As a lead in to the next section, here is another way of getting the 4th argument into `\xtrargs`:

```
\newcommand{\xiargs}[9]{%
  % 9 args used here including
  \def\ivarg{#4}%
  % then
  \xtrargs}
\newcommand{\xtrargs}[2]{%
  % #1 and #2 are the apparent 10th & 11th args
  % call \ivarg for original 4th arg
}
```

where `\def` is the TEX command for defining a command.

This kind of code can obviously be extended to handle as many arguments as you wish, but after a while it might be easier to use the keyval package [3], or the later extension called xkeyval [2], which provide a very different approach. You name each argument and the user can use as many or as few of these as he deems necessary.

> He who can properly define and divide is to be considered a god.
>
> *Novum Organum*, Francis Bacon quoting Plato

3 TEX's dictator

TEX has an all-purpose command for defining new macros, namely `\def`. There are too many aspects to this to cover them all in a short article; Knuth [5, ch. 20] provides the definitive explanation, but you may find that Eijkhout [4, ch. 11] or Abrahams *et al.* [1, chs. 4 and 9] are more accessible or helpful.

The syntax of the `\def` command is unlike anything you see in an author's view of LATEX.

`\def`⟨*cmd*⟩⟨*paramspec*⟩{⟨*defn*⟩}

As in the LATEX formulation, ⟨*cmd*⟩ is the name, including the backslash (e.g., `\amacro`), of the new macro being defined and ⟨*defn*⟩ is the definition of the new macro, just as with LATEX. Note that there are no braces around ⟨*cmd*⟩.

The ⟨*paramspec*⟩ is where you specify the appearance of any arguments to ⟨*cmd*⟩. Each argument is denoted by `#1`, `#2`, etc., in ⟨*paramspec*⟩; these must be in numerical order, and spaces within ⟨*paramspec*⟩ are significant. Below are two equivalent pieces of (LA)TEX code:

```
\newcommand*{\amacro}[2]{....} % LaTeX
\def\amacro#1#2{....}          % TeX
... \amacro{foo}{bar} ...      % (La)TeX
```

That finishes the simple bit, except to say that if you need an argument to consist of one or more paragraphs, by including a blank line or a `\par`, then the macro must be *long*. Also TEX gives no warning if you `\def` a macro that has already been defined — it just throws the old definition away. Be careful of this as it is not a good idea to inadvertently redefine some vital macro that you did not know existed. Anyway, here are two more equivalent pieces of code:

```
\renewcommand{\amacro}[2]{....}       % LaTeX
\long\def\amacro#1#2{....}            % TeX
... \amacro{A paragraph\par}{bar} ... % (La)TeX
```

When the ⟨*paramspec*⟩ consists only of parameters (the `#1` etc.) they are said to be *undelimited*; simplistically these correspond to LATEX's mandatory arguments. On the other hand, if any non-parameter tokens (that is, anything except a `#n` or the opening `{` of the {⟨*defn*⟩}) occur after a `#n` then that parameter is said to be *delimited*. When the new macro is called, the argument for a delimited parameter does not end until TEX encounters the delimiting character(s). Internally, LATEX uses delimited parameters to implement optional arguments.

Suppose we need a macro that looks like this:

`\where{foo}(x,y)`

where `foo`, `x` and `y` are the arguments to `\where`. The LATEX commands described above can't handle this, but TEX can:

```
\def\where#1(#2,#3){#1 in #2 #3}
```

and calling

```
\textit{%
\where{A nightingale sang}(Berkely,Square)}
```

results in

A nightingale sang in Berkely Square

Perhaps you need a command that comes in two versions, as `\newcommand` does. The LaTeX kernel includes a macro called `\@ifnextchar`, whose syntax is like this:
`\@ifnextchar`⟨*char*⟩{⟨*yes*⟩}{⟨*no*⟩}
It looks to see if the next non-space character in the input text is ⟨*char*⟩. If it is it executes the ⟨*yes*⟩ argument, otherwise it executes the ⟨*no*⟩ argument. The kernel also provides the next command:
`\@ifstar`{⟨*yes*⟩}{⟨*no*⟩}
which looks to see if the next character is a `*` and if it is it gobbles up the `*` and executes the ⟨*yes*⟩ argument, otherwise it executes the ⟨*no*⟩ argument. It is defined as follows:

```
\long\def\@firstoftwo#1#2{#1}
\def\@ifstar#1{%
  \@ifnextchar *{\@firstoftwo{#1}}}
```

Now you can define your own (un)starred command pair, like this:

```
\makeatletter  % if not in a .cls or .sty file
\def\maybestar{%
  \@ifstar{\@maybestarS}{\@maybestar}}
  % handle starred version
\def\@maybestarS#1#2{Star (#1) and (#2).}
  % handle plain version
\def\@maybestar#1#2{(#1) and (#2).}
\makeatother   % if not in a .cls or .sty file
```

The end result is a macro with a starred and unstarred version that takes two arguments. A pair of example results are:
`\maybestar*{1st}{2nd}` → Star (1st) and (2nd).
`\maybestar{1st}{2nd}` → (1st) and (2nd).

If you would like to use another character, say a `?`, in place of the `*`, here's a way of doing it.

```
\def\maybeQ{%
  \@ifnextchar ?{\@maybeQ}{\@maybe}}
\def\@maybeQ#1#2#3{Query (#2) and (#3).}
\def\@maybe#1#2{(#1) and (#2).}
```

Unlike the starring code where `\@ifstar` got rid of the `*` the `\@maybeQ` macro has to discard the `?` which is the first character it will see; TeX treats a single character[2] as an argument so `\@maybeQ` is defined such that it throws away its first argument.

A pair of example results are:
`\maybeQ?{1st}{2nd}` → Query (1st) and (2nd).
`\maybeQ{1st}{2nd}` → (1st) and (2nd).

Maybe you would like a LaTeX command that takes two optional arguments and one required one. Heiko Oberdiek has produced a comprehensive package for creating such macros [6] but as another TeX example here is a simple method that might be useful for the odd occasion. The result will be a LaTeX macro, `\twoopt`, that takes one required and two optional arguments. The defaults for the two optional arguments are to be 'opt1' and 'opt2', respectively and unimaginatively.

```
\def\twoopt{%
  \@ifnextchar [{\@twoopt}{\@twoopt[opt1]}}
\def\@twoopt[#1]{%
  \@ifnextchar [%
      {\@@twoopt{#1}}{\@@twoopt{#1}[opt2]}}
\def\@@twoopt#1[#2]#3{%
  1 (#1) 2 (#2) 3 (#3)}
```

Don't forget that this has to be defined when LaTeX thinks that `@` is a letter. Trying this out we get:
`\twoopt{no opts}` → 1 (opt1) 2 (opt2) 3 (no opts)
`\twoopt[foo]{one opt}` → 1 (foo) 2 (opt2) 3 (one opt)
`\twoopt[bar][baz]{two opts}` → 1 (bar) 2 (baz) 3 (two opts)

[2] More precisely, a token, but now is not the time to get into all that.

References

[1] Paul W. Abrahams, Karl Berry, and Kathryn A. Hargreaves. *TeX for the Impatient*. Addison-Wesley, 1990. Available on CTAN in `info/impatient`.

[2] Hendri Adriaens. The xkeyval package, 2005. Available on CTAN in `latex/macros/contrib/xkeyval`.

[3] David Carlisle. The keyval package, 1999. Available on CTAN in `latex/macros/required/graphics`.

[4] Victor Eijkhout. *TeX by Topic, A TeXnician's Reference*. Addison-Wesley, 1991. ISBN 0-201-56882-9. Available at `http://www.eijkhout.net/tbt/`.

[5] Donald E. Knuth. *The TeXbook*. Addison-Wesley, 1984. ISBN 0–201–13448–9.

[6] Heiko Oberdiek. The twoopt package: Definitions with two optional arguments, 1999. Available on CTAN in `latex/macros/contrib/oberdiek`.

[7] Paul Vojta. Re: New York Times headline style. Post to `comp.text.tex` newsgroup, 10 July 2007.

[8] Peter Wilson. Glisterings. *TUGboat*, 28(2):229–232, 2007.

◇ 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://www.ctan.org`) from July 2007–May 2008, 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.

biblio

`aichej` in `biblio/bibtex/contrib/misc`
: BibTeX style file for the *American Institute of Chemical Engineers Journal*.

`bib2ml` in `biblio/bibtex/utils`
: Perl script converting BibTeX databases to HTML, XML, or SQL.

`fbs` in `biblio/bibtex/contrib/misc`
: BibTeX style file for *Frontiers in Bioscience*.

`text2bib` in `biblio/bibtex/utils`
: PHP script to references to BibTeX.

fonts

* `Asana-Math` in `fonts`
: An OpenType math font with almost all mathematical Unicode symbols and the MATH OpenType table supported by XeTeX and Word 2007.

`cyklop` in `fonts`
: OpenType, Type 1, and support files for the Cyklop typeface, a very high contrast display design.

`fge` in `fonts`
: A font for Frege's *Grundgesetze der Arithmetik*.

`gfsbaskerville` in `fonts/greek/GFS`
: The venerable Baskerville design with complete support for Greek (only).

`hwkatakana` in `fonts`
: Half-width Katakana Type 1 fonts in the SJIS encoding, based on the Wadalab fonts.

* `lxfonts` in `fonts`
: Revived LaTeX slide fonts for both text and math.

`umtypewriter` in `fonts`
: Four OpenType fonts supporting Latin, Greek, and Cyrillic scripts.

graphics

`automata` in `graphics/metapost/contrib/macros`
: Draw finite state machines, graphs, trees, etc.

`bpolynomial` in `graphics/metapost/contrib/macros`
: Draw polynomial functions up to degree three.

`dot2tex` in `graphics`
: Convert Graphviz-format graphs to LaTeX-friendly formats.

`graphicxsp` in `graphics`
: Embed graphics in PDF output once, and support the Adobe transparency imaging model.

`pgfplots` in `graphics/pgf/contrib`
: Normal and logarithmic plots using Ti*k*Z/PGF.

`pgf-soroban` in `graphics/pgf/contrib`
: Draw a Japanese abacus using Ti*k*Z/PGF.

`pst-cox` in `graphics/pstricks/contrib`
: Draw two-dimensional projections of complex regular polytopes (after Coxeter).

`pst-diffraction` in `graphics/pstricks/contrib`
: Draw diffraction patterns for different geometric forms of apertures for monochromatic light.

`pst-optexp` in `graphics/pstricks/contrib`
: Facilitate sketching of optical experimental setups.

`pst-pad` in `graphics/pstricks/contrib`
: Draw attachment surfaces for friction and adhesion models.

`pst-solides3d` in `graphics/pstricks/contrib`
: Draw solids in three-dimensional perspective.

`pst-soroban` in `graphics/pstricks/contrib`
: Draw a Japanese abacus using PSTricks.

`tableau-variations` in `graphics/metapost/contrib/macros`
: Create variation tables in MetaPost.

info

`compact` in `info/symbols/compact`
: Beamer-style list of common symbols, including the variations in `txfonts`.

`dtxgallery` in `info`
: Collection of minimal `.dtx` examples.

`l2tabues` in `info/l2tabu/spanish`
: Spanish translation of `l2tabu`.

* `latex-brochure` in `info`
: Publicity flyer for LaTeX; please print and post wherever appropriate.

`lshort-slovenian` in `info/lshort/slovenian`
: Slovenian translation of `lshort`.

`mpman-ru` in `info/metapost/doc/russian`
: Russian translation of the MetaPost user's guide.

* `texbytopic` in `macros/latex/contrib`
: Victor Eijkhout's excellent TeX reference book *TeX by Topic*, now available under the GNU Free Documentation License.

`xetexref` in `info`
: Unofficial reference documentation for XeTeX.

language

mx in language/spanish/nonstandard
: Babel support for typesetting Spanish with Mexican conventions.

ptex in language/japanese
: A high-quality Japanese TEX system by ASCII Co., compatible with teTEX-based systems. Released under the modified BSD license.

macros/latex/contrib

addlines in macros/latex/contrib
: Enhancements of \enlargethispage.

ametsoc in macros/latex/contrib
: Official American Meteorological Society LATEX template files.

boldtensors in macros/latex/contrib
: Provide bold Latin and Greek characters within \mathversion{normal}, by using ~ and " as prefix characters.

catechis in macros/latex/contrib
: Support for catechisms: question-and-answer, comments on answers, etc.

changepage in macros/latex/contrib/misc
: Locally changing text width or other page layout based on odd/even pages; compatible with memoir, unlike chngpage.

chemscheme in macros/latex/contrib
: Support for a chemical scheme float type with automatic numbering.

chemstyle in macros/latex/contrib
: Formatting chemistry documents (successor to the rsc bundle).

confproc in macros/latex/contrib
: Class for conference proceedings.

constants in macros/latex/contrib
: Label/reference system for (families of) constants, e.g., in a proof.

* datatool in macros/latex/contrib
: Database operations and charts within LATEX, possibly created from external CSV, TSV, etc. files.

dotarrow in macros/latex/contrib
: Extendable dotted arrows (like \xrightarrow).

dprogress in macros/latex/contrib
: Log LATEX's progress through the document, outputting section headers, equation numbers, etc.

ecclesiastic in macros/latex/contrib
: Typesetting Latin in an ecclesiastical style.

ed in macros/latex/contrib
: Editorial notes for LATEX documents.

edmargin in macros/latex/contrib
: Support for endnote sections in critical editions.

environ in macros/latex/contrib
: Collect body text inside an environment.

epspdfconversion in macros/latex/contrib
: Do EPS-to-PDF conversion on the fly from within LATEX.

errata in macros/latex/contrib
: Recording errata in LATEX documents and then highlighting changes.

etoolbox in macros/latex/contrib
: Programming facilities for LATEX class and package authors, including but not limited to support for new ε-TEX primitives.

euproposal in macros/latex/contrib
: Support for FP7 proposals.

fouridx in macros/latex/contrib
: Enable left subscripts and supersets; alternative to \sideset and \leftidx.

frletter in macros/latex/contrib
: Typeset French-style letters (in any language).

frontespizio in macros/latex/contrib
: Typeset thesis frontispiece in Italian style.

gaceta in macros/latex/contrib
: Support for the journal *La Gaceta de la Real Sociedad Matemática Española*.

hexgame in macros/latex/contrib
: Draw a hexagonal game board.

ifplatform in macros/latex/contrib
: Conditionals for testing the current operating system (using -shell-escape).

ijmart in macros/latex/contrib
: LATEX class for the *Israel Journal of Mathematics*.

image-gallery in macros/latex/contrib
: Generate overview page of a photo gallery, possibly adjusting photo sizes.

isonums in macros/latex/contrib
: Display numbers in math mode according to ISO-31-0, regardless of input format.

jsclasses in macros/latex/contrib
: Classes supporting Japanese, including jsarticle and jsbook.

lcyw in macros/latex/contrib
: LATEX support for the CM Cyrillic fonts.

mathexam in macros/latex/contrib
: LATEX package for typesetting exams, including space for students to show work.

mciteplus in macros/latex/contrib
: Reimplementation of mcite with support for physics-style grouping of multiple citations; compatible with REVTEX4.

mfpic4ode in macros/latex/contrib
: Draw direction fields and other differential equation support.

newspaper in macros/latex/contrib
: Typesetting newsletters to resemble newspapers.

nostarch in macros/latex/contrib
: Official LATEX for No Starch Press.

notes2bib in macros/latex/contrib
: Integrate footnotes, endnotes, and other notes in a bibliography.

`numname` in `macros/latex/contrib`
Convert a numerical number to the English spelled-out name of the number.

`perltex` in `macros/latex/contrib`
Define LaTeX macros in terms of Perl code.

`philex` in `macros/latex/contrib`
Dynamically cross-reference to names and contents of environments.

`sagetex` in `macros/latex/contrib`
Embed Sage code and plots into LaTeX.

`sfmath` in `macros/latex/contrib`
Switch all mathematics typesetting to sans serif.

`show2e` in `macros/latex/contrib`
Support for `\show` variants suited to LaTeX 2ε's commands and environments, and other debugging help.

`shipunov` in `macros/latex/contrib`
Bundle with support for field biologists, Russian-language authors, and much more.

`siunitx` in `macros/latex/contrib`
Comprehensive package for typesetting values with units.

`stex` in `macros/latex/contrib`
(La)TeX support for mathematical knowledge management, via allowing semantic markup without leaving the document format.

`stringstrings` in `macros/latex/contrib`
Large collection of expandable routines for string processing.

`subdepth` in `macros/latex/contrib`
Unify math subscript height.

`tablor` in `macros/latex/contrib`
Create tables for signs and variations using XCAS for calculus and MetaPost for drawing.

`ted` in `macros/latex/contrib`
Token list editor: substitutions and display of tokens.

`theoremref` in `macros/latex/contrib`
Variants of `\label` and `\ref` for theorem-like environments.

`thinsp` in `macros/latex/contrib`
A stretchable `\thinspace` for LaTeX.

`*translator` in `macros/latex/contrib`
Open translation platform, compatible with `babel`.

`turnstile` in `macros/latex/contrib`
Support for logicians' turnstile sign.

`upmethodology` in `macros/latex/contrib`
Support for Unified Process methodologies (UP or RUP).

`varsfromjobname` in `macros/latex/contrib`
Select hyphen-separated pieces of `\jobname`.

`vxu` in `macros/latex/contrib`
Classes for Växjö University, Sweden.

`xargs` in `macros/latex/contrib`
Define macros with many optional arguments.

`xskak` in `macros/latex/contrib`
Extensions to `skak` for chess typesetting, including saving information for later use.

macros/latex/exptl

`biblatex-dw` in `macros/latex/exptl/biblatex-contrib`
Support for humanities-style citations and more, for `biblatex`.

`biblatex-mla` in `macros/latex/exptl/biblatex-contrib`
Support for MLA-style parenthetical citations and bibliographies in `biblatex`.

`gcite` in `macros/latex/exptl`
German-style citations, providing a small amount of bibliographic information in a footnote on the page where each citation is made.

`thmtools` in `macros/latex/exptl`
Keyval interface to `\newtheorem`, a `\listoftheorems` command, `hyperref` compatibility, and more theorem extensions.

macros/omega

`tamil-omega` in `macros/omega/latex/contrib`
Typeset the Tamil script using Omega.

macros/xetex

`xecyr` in `macros/xetex/latex`
Support for Cyrillic languages in XeLaTeX + `babel`.

support

`acroweb` in `support`
Scripts for creating interactive tests in AcroTeX from web databases.

`autolatex` in `support`
Automate generation of large-scale LaTeX projects.

`ctantools` in `support`
Search LaTeX packages on CTAN.

`delig` in `support`
Java program to disable misplaced ligatures in LaTeX documents, for example "Auflage" in German.

`fastpictex` in `support`
Preprocessor for PiCTeX to ease creation of line graphs, bar graphs, x-y graphs, etc.

`graphconv` in `support`
Windows program to convert Graph (`http://www.padowan.dk/graph/`) files to PSTricks.

`jlm` in `support/jed`
LaTeX mode for the JED editor.

`miktex_update` in `support`
Automate updating and installing of new packages of an existing MiKTeX installation.

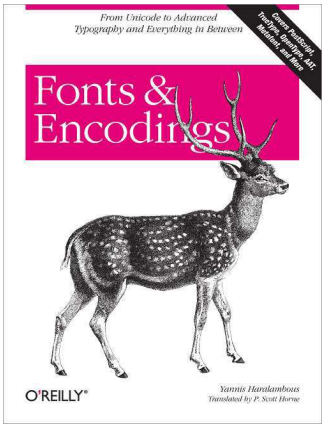
`texcount` in `support`
Count number of words in LaTeX text, with color output. Also available as a web service: `http://folk.uio.no/einarro/Comp/texwordcount.html`.

`vc` in `support`
Version control for (La)TeX, supporting Subversion and Git.

Reviews

Book review: *Fonts & Encodings* by Yannis Haralambous

Ulrik Vieth



Yannis Haralambous, translated by P. Scott Horne, *Fonts & Encodings*, O'Reilly Media, 1016+xx pp., first edition: September 2007, ISBN 0-596-10242-5, US$59.99. `http://www.oreilly.com/catalog/9780596102425`

Yannis Haralambous is well-known in the international TEX community, not only as a co-founder of the Omega project, but also for his numerous contributions as a developer of fonts for various languages. It only seems fitting that Yannis has undertaken the job of writing a comprehensive book on the topic of fonts and encodings.

The original edition, entitled *Fontes & Codages*, appeared in 2004, but only in French. Now, a long-awaited English translation as *Fonts & Encodings*, prepared by P. Scott Horne, has recently become available, published by O'Reilly Media Inc.

Contents

Amounting to a bit more than 1000 pages, the book matches the size of *The LATEX Companion, Second Edition*. It appears quite impressive, not only regarding its sheer size, but also regarding the broad range of topics covered as well as the depth of the coverage and the level of detail. In some cases the author has spent dozens of pages documenting some arcane details of font formats, which have so far been lacking a comprehensive or accessible documentation from other sources.

The book consists of a main body of 14 chapters of some 600 pages, followed by an appendix of 7 chapters amounting to another 400 pages.

As the author makes clear in the introduction, various groups of readers may benefit from different parts of the book without having to read all of it: Some chapters are mostly encoding-specific, dealing with characters on the input side, some are mostly font-specific dealing with glyphs on the output side, while other chapters find themselves in the middle ground, having to deal with font and encoding topics simultaneously. Some chapters are accessible to end users interested in installing and using fonts, while others are of interest only to font designers or developers of font-related software.

The first part of the book starts with encoding-specific topics. Chapter 1 provides an overview of the history of encodings before Unicode, ranging from 7-bit ASCII and various 8-bit ISO encodings to 16-bit East-Asian encodings. Chapters 2–4 cover the Unicode standard, starting with an overview of the symbols and scripts included in the standard and moving on towards more and more complex implementation details. Chapter 5 completes this part with a presentation of some useful tools for using Unicode input on various system platforms.

The second part covers the topic of font management on various system platforms and operates somewhere in the gray area between fonts and encodings. Chapters 6–8 each cover similar topics for the Macintosh, Windows and Unix/X11 platforms. While the description of the Macintosh platform is rather detailed in discussing the differences of font handling between MacOS 9 and Mac OS X, the description of the Unix/X11 platform only covers some very basic and old-fashioned X11 tools. Here, one could have wished for some more extensive coverage of font management in modern Linux desktop environments such as KDE or Gnome.

The following two chapters discuss platform-independent usage of fonts in TEX/Omega systems and on the Web. Chapter 9 starts with an overview of high-level font selection in LATEX/NFSS2, followed by a detailed description of low-level font installation for *dvips*. The remainder of the chapter then discusses numerous examples of creating virtual fonts using *fontinst* to implement specific effects needed in various scripts. Chapter 10 concludes this part with a coverage of fonts on the Web using either (X)HTML/CSS or alternatively SVG.

The final part of the book covers font-specific topics. Chapter 11 covers various classifications of Latin typefaces and simultaneously provides a nicely-illustrated overview of the history of the most important typeface designs. Chapters 12–13 then discuss creating, editing and optimizing PostScript, TrueType, and OpenType fonts using tools such as *FontLab* and *FontForge*. Chapter 14 finally introduces the concepts of advanced typographic features provided in OpenType or AAT fonts and discusses ways of enriching fonts using these facilities.

The appendix of the book mostly consists of

the detailed descriptions of font formats. Starting from bitmap fonts and TEX-related font formats and moving on to PostScript, TrueType, OpenType and AAT fonts, practically all relevant font formats are covered in detail in Appendices A–E.

Finally, Appendix F discusses the principles of font design in METAFONT and derived systems such as METAPOST, MetaFog and MetaType 1.

Commentary

Considering the size of the book, it is understandable that several years have passed from the time of writing the manuscript to the publication of the English translation. Unfortunately, because of this, some chapters of the book are in risk of becoming out-of-date rather quickly. For most of the material, we have to assume that the English edition of 2007 only represents the state of the art of 2003.

For many chapters serving as a reference, such a delay is not much of a problem, as the descriptions of encodings or font file formats remain unchanged and permanently valid. On the other hand, it is regrettable that especially the chapter about TEX has completely missed or overlooked some very important developments of the last few years.

As one example, when describing the details of font installation, the author only covers TEX/Omega with *dvips*, while PDFTEX isn't mentioned in this context, even though most of the description would be applicable to both systems in TEX Live systems. (In fact, PDFTEX isn't mentioned anywhere at all in the book, perhaps because it didn't support Omega at that time.)

As another example, Hàn Thế Thành, the principal author of PDFTEX, is mentioned only once in the context of Vietnamese fonts, while his significant achievements regarding the implementation of micro-typographic features of PDFTEX have been neglected completely. This is even more surprising as the author spends some pages discussing an example where the effect of *margin kerning*, which would have been accessible in PDFTEX, is simulated in a rather cumbersome way using virtual fonts created by *fontinst*.

Regarding examples of extensions of Computer Modern fonts, the author suggests the *CM-Super* fonts, while the (by now) much more popular Latin Modern fonts are mentioned only in passing as an example of a MetaType 1 application.

Finally, when it comes to discussing ways of using advanced typographic features of OpenType or AAT fonts in TEX, the author only offers some hints about his own research work in the Omega 2 project (which hasn't progressed beyond the prototype stage), while the (by now) readily-available newcomer XETEX remains unmentioned.

To be fair, one has to admit that the success and importance of these recent developments in the TEX world could not have been foreseen at the time of writing in 2003. Nevertheless, it could have been possible to include some additions and/or revisions by the time of the English translation of 2007.

It is rather unfortunate that the opportunity for updates was missed here; this would have made the book much more useful and valuable for TEX users interested in making use of the very latest of developments in font technology.

Despite these shortcomings, the book remains a valuable resource for TEX users and software developers, who are deeply interested in font technology and encodings. There is no other book providing a similar coverage in the broad range of topics and the deep level of detail in a single volume. To get anywhere near it, one would have to collect dozens of references from a variety of sources — and one would still be left with some gaps to fill.

In summary, this book is certainly recommendable. Nevertheless, some additions and/or revisions would be very desirable for future editions.

As a final remark, the reviewer would like to mention a little curiosity: like some other modern textbooks, this book also features a separate index of persons besides the usual general-purpose index. However, unlike other books, the author seems to have been rather liberal as to which kinds of persons are referenced in the index.

This way, authors of font software and tools not only find themselves in the glorious company of some the most famous font designers of history, but also in the vicinity of rather questionable political figures (e. g. Lenin, Hitler, Mao) as well as some fictional or literary characters (e. g. Sherlock Holmes, James Bond, James T. Kirk), which are only mentioned in passing in some light-hearted comments.

While the reviewer (who at one time was a contributor to *fontinst*) wishes to express thanks for the opportunity of being included in this unique selection of people, he also wishes to express serious doubts as to whether it is really helpful to the reader to include fictional characters in the person index in the same way as technical people.

◇ Ulrik Vieth
Vaihinger Straße 69
70567 Stuttgart
Germany
ulrik dot vieth (at) arcor dot de

Software review: TEXCAD for Windows

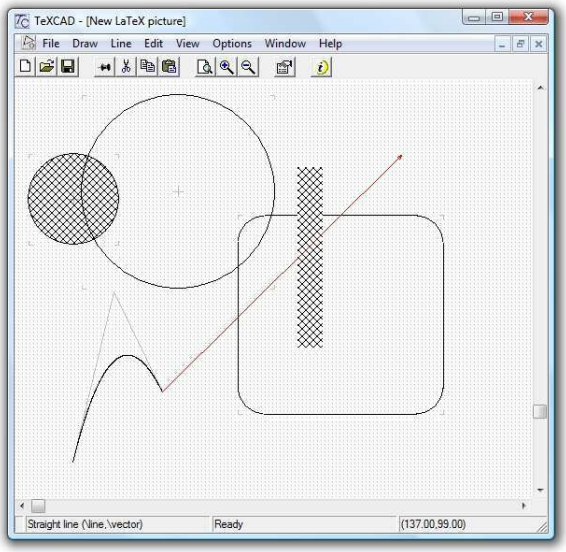
Bernd S. W. Schroeder

Overall Rating: 5.0 (highest rating possible)

1 Description

This program is a well-done adaptation of the classic DOS TEXCAD to the Windows platform. TEXCAD for Windows is a drawing program that provides a visual way to produce pictures in LATEX picture environments. It virtually eliminates the need to remember the way LATEX encodes pictures as most types of pictures can be generated quickly with TEXCAD. TEXCAD for Windows is freely available for download.

The review below considers three categories and assigns a rating in each.



2 Category: Content quality

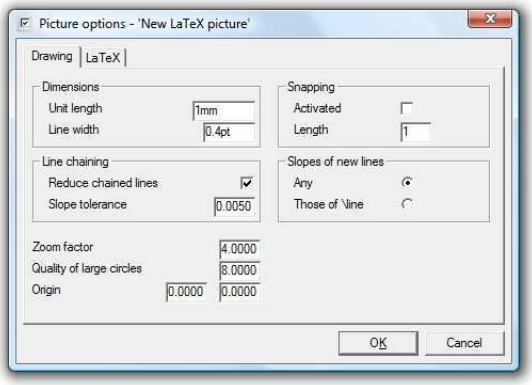
Rating: 5.0

2.1 Strength(s):

- Grid-based drawing program with a default grid in millimeters. Step length for the mouse can be adjusted under Options (Zoom Factor). A zoom factor of 10 provides steps of length 1/10 mm.
- Supports all functions that are available via the LATEX picture environment. Functions are selected via self-explanatory pull-down menus.
- Except for some text, TEXCAD for Windows shows a picture that is what LATEX will produce.
- Graphical editing features such as mirror copy and rotated copies.
- For Bezier curves the two tangent lines that determine the curve are displayed.
- Circles of any size can be drawn with TEXCAD (this feature of TEXCAD32 was not accessible in recent Windows versions because TEXCAD32 would not run).

2.2 Concern(s):

- A feature for connecting objects so that when one is dragged the others adjust would be nice. This can clash with the limited number of slopes of lines that LATEX allows though, so it is understandably not available.
- For fine tuning, it would be nice if one could move the cursor one step at a time with the cursor keys. (This was helpful in the original TEXCAD.)



3 Category: Potential effectiveness as a teaching tool

Rating: 5.0

3.1 Strength(s):

- Generation of LATEX picture environments that are sufficient for many mathematical graphics and very compact.
- Several functions (lines with any slope, bezier vectors) are geared towards eradicating some of the LATEX picture environment's limitations.
- TEXCAD for Windows interfaces with MiKTEX to allow preview of the actual LATEX picture in a very efficient edit-view-edit cycle.

3.2 Concern(s):

- The tips of Bezier vectors sometimes completely cover the end of the Bezier curve, and a little

Editor's note: This is a peer review by the MERLOT Mathematics Review Panel, published online January 16, 2004 (`www.merlot.org`). Reprinted by permission. TEXCAD is written by Gautier de Montmollin, who maintains a site at `www.mysunrise.ch/users/gdm/texcad.htm`. Adapted for *TUGboat* and screenshots provided by Steve Peter.

stub sticks out of the triangle that is the tip of the arrow. This occurs only for high curvature near the tip of the vector and going into the code for the picture environment can fix this problem easily. (*Note*: Gautier de Montmollin says this has been fixed.)

4 Category: Ease of use for both students and faculty

Rating: 5.0

4.1 Strength(s):

- Simple, matter-of-fact documentation included in English (dvi format).
- Once it is started the interface is highly intuitive. Mouse movements plus the left button (select) and the right button (escape/cancel) are all that is needed to run the whole application.
- TEXCAD is an executable that can be started through Windows Explorer, My Computer or on the command line. Shortcuts can be created.

4.2 Concern(s):

None.

5 Other issues and comments

The original TEXCAD is a program back from the days when DOS was the operating system on PCs. TEXCAD was firmly ahead of its time. The limitations of TEXCAD are mostly the limitations of the LATEX picture environment, so they cannot be held against this program. The new Windows version eradicates just about all problems that the reviewers saw in the original program.

Gautier de Montmollin is planning to further improve the program and add some of the features mentioned in this review.

◇ Bernd S. W. Schroeder
Edmondson/Crump Professor and Program Chair
Program of Mathematics and Statistics
Louisiana Tech University
Ruston, LA 71272

Warnings

`\looseness` on the loose

Frank Mittelbach

Question:

This paragraph was set twice with a small value for `\emergencystretch`. The first time it was set without any other special adjustments, the second time we also used `-1` as the value for the `\looseness` parameter. Can you explain why the two paragraphs are broken differently into lines even though the use of `\looseness` couldn't shorten the paragraph?

This paragraph was set twice with a small value for `\emergencystretch`. The first time it was set without any other special adjustments, the second time we also used `-1` as the value for the `\looseness` parameter. Can you explain why the two paragraphs are broken differently into lines even though the use of `\looseness` couldn't shorten the paragraph?

Answer: When `\looseness` has a non-zero value, TEX always runs through all line breaking passes: breaking without hyphenation, breaking with hyphenation, and (if `\emergencystretch` is non-zero as it is, for example, inside multicols) with the emergency stretch. But adding `\emergencystretch` to every line means that the line breaks chosen in the first paragraph may fall in different fitting classes so that at different places `\adjdemerits` are charged, thus making the original solution less attractive.

In fact the situation could even be worse: if a long paragraph can be broken into lines by just using `\pretolerance`, then a setting of `\looseness` to `+1` might in fact result in a paragraph with one line less — all that would be required is that breaking using `\tolerance` to get a default line count two lines less than with `\pretolerance` (a real life example is left to the reader).

◇ Frank Mittelbach
http://www.latex-project.org

First published in Die TEXnische Komödie 2007/4, p. 41.

Abstracts

MAPS 35 (Spring 2007)

MAPS is the publication of NTG, the Dutch language TEX user group. Their web site is `http://www.ntg.nl`.

Hans Hagen, Taco Hoekwater, The MPlib Project; p. 4

MetaPost as a reusable component.

Hans Hagen, Tokens in LuaTEX; pp. 5–8

Discussion of token parsing in LuaTEX.

Taco Hoekwater, Integrating the pool file; pp. 9–10

This short article discusses the method that is used in MetaPost and LuaTEX (since adopted in all other TEX family programs) to integrate the string pool file into the program.

This method allows the redistribution of a single updated executable in place of both a program and a data file, and this makes updating those programs easier on both the user and the developer.

Willi Egger, PGF/Ti*k*Z; pp. 11–17

For those who are looking for an alternative to external graphic drawing tools, PGF/Ti*k*Z offers a wealth of possibilities. PGF is a macro package that, together with its user interface Ti*k*Z, comprises a kind of 'graphics language' to build graphics inside the text as inline graphics or as pictures of larger size. PGF was originally written for LATEX, but it is now also available for use within ConTEXt. The package comes with a large set of libraries for different kinds of graphics. There is extensive documentation and a tutorial. For support a mailing list and web site are available. Users of the package with ConTEXt have to install the xkeyval package version 1.8. PGF and Ti*k*Z are distributed under the GNU Public License version 2.

External graphics for LATEX, Siep Kroonenberg; pp. 18–26

In this article, we discuss graphics file formats, software to create graphics, and procedures to convert them to LATEX and pdfLATEX-compatible formats.

Hans Hagen, Taco Hoekwater, Review: Alphabetgeschichten; pp. 27–29

[Printed in *TUGboat* 28:2.]

Richard Hirsch, Folding Sheets for a Modular Origami Dodecalendar; pp. 30–36

Twelve square sheets of paper can be folded in such a way that they can be assembled to a pentagon dodecahedron (origami). The single units are called modules, hence the name modular. If the sheets bear calendrical information at the right places, the dodecahedron shows the calendar for each month on its faces: the dodecalendar.

In this article we let MetaPost calculate piece by piece the information that needs to be printed on the module paper to enable us to fold the modules and assemble the dodecahedron.

Mojca Miklavec, ConTEXt user meeting 2007; pp. 37–42

[Printed in *TUGboat* 28:2.]

Michael Guravage, EuroBachoTeX 2007; pp. 43–50

[Printed in *TUGboat* 28:2.]

Frans Goddijn, MiKTEX installeren valt erg mee [Installing MiKTEX easier than expected]; pp. 51–54

The author's personal experiences installing MiKTEX.

[Received from Taco Hoekwater]

Die TEXnische Komödie Contents of issues 2007/2–2008/1

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/`.

2007/2

André Miede, Academic final theses with classicthesis; pp. 39–44

There are dozens of college-specific templates for a final thesis out there, each describing a special set of formal requirements. This article describes a solution, which emphasizes typographic beauty and offers a formal base for a thesis.

Joachim Schlosser, Large documents with LATEX — versioning, variants, automation and change tracking; pp. 45–67

LATEX has always supported large documents, but work-flows can be simplified and automated to a higher degree regarding modularisation, change tracking and preprint processing. This article aims to inspire without claiming to be the only feasible solution.

Rolf Niepraschk, Tips and Tricks: pdfLATEX, bitmap graphics and dimension `px`; pp. 68–69

pdfTEX supports a new length unit called `px`. A `px` is the distance between two screen dots of the resolution presumed by pdfTEX.

2007/3

Frank Mittelbach, Robin Fairbairns, Werner Lemberg, LATEX font encoding; pp. 22–59

The article describes the font encoding support in LATEX and the resulting imitations for new encodings. Furthermore it includes a detailed description of all the currently registered encodings.

An English version (`encguide.tex`) is part of the LATEX distribution.

Jürgen Fenn, Joachim Schlosser: Writing scientific papers with LATEX; pp. 65–68

Review of *Wissenschaftliche Arbeiten mit LATEX schreiben. Leitfaden für Anfänger*, by Joachim Schlosser (Verlag, Readline: Heidelberg 2007).

2007/4

Jürgen Fenn, Bibliography management with Firefox, Zotero and BibTEX; pp. 20–26

The article presents the new bibliography management tool Zotero, which can be integrated as a Firefox plugin. Zotero can handle BibTEX data.

Stephan Hennig and Herbert Voss, The font package `initials`; pp. 27–40

This article shows how the font package initials can be used in LATEX documents.

Frank Mittelbach, `\looseness` on the loose; p. 41

(Published in this issue of *TUGboat*.)

Herbert Voss, The `listings` package and UTF-8 encoding; pp. 45–56

The listings package is the de facto standard for the output of program code. The package has excellent possibilities to format code sequences, but it does not support UTF-8 encoding. Using a trick, however, it is possible to get around this problem for single tokens.

Markus Kohm, How to process the contents of a KOMA-Script variable as macro; pp. 48–59

The KOMA-Script class `scrlttr2` offers a new LATEX element: variables. The defined interfaces to the variables are the commands `\newkomavar`, `\setkomavar`, `\usekomavar` and `\ifkomavarempty`. But what to do when, e.g., you would like to test if a variable has a certain non-empty content?

Jürgen Fenn, Anselm Lingnau: LATEX Hacks; pp. 56–58

Review of the new publication *LATEX Hacks. Tipps & Techniken für professionellen Textsatz* by Anselm Lingnau (O'Reilly: Köln u. a. 2007).

Jürgen Fenn, M. Goossens, F. Mittelbach, S. Rahtz, D. Roegel, H. Voß: The LATEX Graphics Companion; pp. 59–61

Review of the *LATEX Graphics Companion*, second edition.

2008/1

Stephan Hennig, Some remarks on the article 'Hyphenation Exception Log for German hyphenation patterns, version 1'; pp. 7–17

This article raises some questions not answered by Werner Lemberg's articles (in DTK 2/2003 and 2/2005) on hyphenation patterns and exceptions.

Volker RW Schaa, Herman Zapf received the Goethe medallion; pp. 18–21

On December 11, 2007, in the Wiesbadener Staatskanzlei, our honorary member Prof. Dr. h.c. Hermann Zapf was awarded the Goethe-Plakette ('badge of honor'), the highest award of the Hessian Ministry of Science and the Arts.

Ulrik Vieth, Yannis Haralambous: *Fonts & Encodings*; pp. 22–25

(Published in this issue of *TUGboat*.)

Eutypon 16–20

Editor's note: *Eutypon* is the journal of the Greek TEX Friends, the Greek TEX user group. The journal's web site is `http://www.eutypon.gr/eutypon`.

Eutypon 16–19 triple issue, October 2007

Erik Meijer, Citations, reference list, and author index with `apacite`; pp. 1–31

The `apacite` package can be used with LATEX and BibTEX to generate citations and a reference list, formatted according to the rules of the American Psychological Association. Furthermore, `apacite` contains an option to (almost) automatically generate an author index as well. A recent addition is the support of different languages. The package can be customized in many ways. This paper describes the `apacite` package, paying special attention to its idiosyncrasies and how the problems associated with these have been solved. (*Article in English.*)

Athanassios Protopapas, `apa.cls`: A genuine LATEX solution for psychological research articles; pp. 33–52

Psychological research manuscripts must usually conform to the guidelines of the American Psychological Association (APA) publication manual. The LATEX document class `apa.cls` implements the structural requirements of the manual, so that authors have to concern themselves only with manuscript content. By separating appearance from content, in LATEX fashion, `apa.cls` can provide visually distinct outputs from the same manuscript file, thus producing manuscript-format or journal-style documents by switching a processing option. This article presents a bit of history and context for the development of `apa.cls`, noting the critical importance of an active online community of developers and users. There are several technical issues involved in handling the requirements of the APA manual, and these are discussed here along with their solution provided in `apa.cls`. The special macros and options of `apa.cls` are presented, with examples, on the topics of titles/headers, sectioning, lists, floats, typefaces, appendices, internationalization, and conditionals. (*Article in English.*)

Thomas A. Schmitz, Greek support for the ConTEXt macro package; pp. 53–67

This paper describes the implementation of support for typesetting ancient (polytonic) Greek in ConTEXt. ConTEXt is a macro package for TEX. It allows for a great deal of flexibility and customizability. Support for typesetting polytonic Greek was lacking. The article describes in detail what was needed to typeset Greek with ConTEXt. It discusses the most frequently used input methods (Unicode and transliterated ASCII babel input), fonts and encodings and some of the problems that had to solved. It also describes some of the challenges and new possibilities which luaTEX, the designated successor to pdfTEX, is bringing. (*Article in English.*)

Apostolos Syropoulos, The XETEX typesetting machine; pp. 69–74

XETEX is a new typesetting machine with many elements borrowed from ε-TEX and Ω. XETEX allows the direct use of TrueType and OpenType fonts. This paper is a short presentation of XETEX and its capabilities as well as a presentation of the `xgreek` package. (*Article in Greek.*)

Dimitrios Filippou, Half a century of **Helvetica** and one century of ***grotesque***; pp. 75–84

This year [2007] marks the 50th anniversary since the Helvetica fonts were put into circulation, and the event has received considerable attention from media of all kinds: the printed media, the electronic media, even from cinematographers. However, behind the media noise, one easily discovers that Helvetica did not fall from the heavens half a century ago. The roots of Helvetica lie in the German realistic *grotesque* typefaces of the end of the 19th century, which in turn have their roots in the first British jobbing typefaces of 1816–1834. This article is a short presentation of Helvetica, from its ancestors to its imitators. (*Article in Greek.*)

Eutypon 20, April 2008

Vassilios Tsagkalos, The Greek Font Society and Georgios D. Matthiopoulos; pp. 1–9 (Published in this issue of *TUGboat*.)

Dimitrios Filippou, Jean Kefalinos and Emmanuel Ch. Kasdaglis: two fighters for quality typography; pp. 11–21

Jean Kefalinos (b. Alexandria, Egypt, in 1894; d. Athens, Greece, in 1957) is considered as one of the best engravers of Modern Greece. But, beyond being a unique engraver, Kefalinos was also an exceptional book designer and decorator. Emmanuel Ch. Kasdaglis (b. Piraeus, Greece, in 1924; d. Athens, Greece, in 1998) was the man who brought into light the work and contribution of Jean Kefalinos in Greek typography. Kasdaglis started as a print shop corrector to help his meager family income. In 1966, he became the first director of the National Bank of Greece Cultural Foundation, now an institution renowned for its outstanding publications. As a book editor for nearly five decades, Kasdaglis contributed

enormously in the preservation and institutionalisation of book æsthetics in Greece, in a period when the low-cost sloppy print became the norm of Greek typographers. (*Article in Greek.*)

Werner Lemberg, Unicode support for the Greek `LGR` encoding; pp. 23–36

Up to now, only the `ucs` package has provided Unicode support for Greek. This article describes new support files for the `LGR` encoding which does the same (and even more) for LaTeX's default `inputenc` mechanism. The files described in this article can be downloaded from `http://www.latex-project.org/cgi-bin/ltxbugs2html?pr=babel/4015`. (*Article in English.*)

Ioannis K. Dimakos, TeX in the field of statistics: The power of free software; pp. 37–47

The statistical programming environment named `R`, a free software package, is presented in this article. This environment, in conjuction with the flexibility, ease of use and capabilities of TeX and its associated programs, offers every author the ability to create high fidelity mathematical and statistical texts, along with the necessary graphics. (*Article in Greek.*)

Georgios Georgiou, Experiences with XeTeX; pp. 49–51

First experiences of using XeLaTeX for typesetting Greek texts. (*Article in Greek.*)

Apostolos Syropoulos, TeXniques; pp. 53–55

Practical solutions for the common TeXxie: detecting the appropriate compiler for a given `.tex` file; putting diagonal lines in table cells; fixing overfull paragraphs. (*Article in Greek.*)

ArsTEXnica
Contents of issue #4 (October 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`.

Massimiliano Dominici, Editoriale [From the editor]; p. 3

A short overview of the present issue.

Lapo F. Mori and Maurizio W. Himmelmann, Scrivere il curriculum vitæ con LaTeX [How to write a curriculum vitæ with LaTeX]; pp. 5–15

This paper presents the tools that are currently available to prepare a curriculum vitæ with LaTeX, with a critical analysis of packages and classes.

Claudio Beccari and Andrea Guadagni, La progettazione di un'opera di consultazione: l'edizione del *Prontuario dell'ingegnere* con LaTeX [The design of a reference book: The production of the *Engineer's Quick Reference Book* with LaTeX]; pp. 16–24

(Published in this issue of *TUGboat*.)

Massimo Caschili, Introduzione a PSTricks [An introduction to PSTricks]; pp. 25–44

PSTricks is a powerful graphic system established by a large number of extensions; it offers many tools to produce pictures, graphical representations and figures with high-quality effects and an high-quality typographical performance.

Luciano Battaia, LaTeX nella Scuola Media Superiore: applicazioni didattiche con PSTricks [LaTeX in secondary high school: didactic use of PSTricks]; pp. 45–50

The aim of this paper is to promote the use of LaTeX, and in particular PSTricks with its extensions, in secondary high school and to debate the positive influences of this fact on math teaching.

The paper is not a technical introduction to PSTricks: rather it reflects the author's ideas concerning the real possibility of reconciling both "old fashioned" school programs and LaTeX strategies. The techniques discussed here have been used in a course at Liceo Scientifico Grigoletti, Pordenone.

Agostino De Marco, Illustrazioni tridimensionali con Sketch/LaTeX/PSTricks/Tik*Z* nella didattica della Dinamica del Volo [Three-dimensional illustrations with Sketch/LaTeX/PSTricks/Tik*Z* in the teaching of flight dynamics]; pp. 51–68

This article shows how combining LaTeX with the package PSTricks or with Tik*Z* can be used to produce advanced, nice-looking illustrations. As a matter of fact, the creation of drawings representing three-dimensional scenes with scientific or non-trivial annotations is possible with LaTeX. One of the goals of the article is introducing the program Sketch, by Eugene Ressler, and how one can manipulate and put in place objects in a three-dimensional scene by means of its intuitive scripting language. The output of Sketch is a set of PSTricks or Tik*Z* commands that can be included by a master LaTeX document to produce the final picture. The technique proposed here enables overcoming the limitations encountered by PSTricks or Tik*Z* users when it comes to representing non-trivial three-dimensional scenes.

As a teacher of engineering subjects related to flight dynamics, I report some concrete examples

that may help to better understand the potential of Sketch and of the workflow proposed in the article.

Norbert Preining, TEX Live's new infrastructure; pp. 69–73

Since the release of TEX Live 2007 a new infrastructure for TEX Live distribution and management has been developed. This article presents the reasons for this switch, the ideas behind the new infrastructure, software developed, and ways to incorporate this new infrastructure. We will close with a look at what new features this new infrastructure could bring to the TEX (Live) world.

Klaus Höppner, Typesetting tables with LATEX; pp. 74–77

From a LATEXoligist's point of view, LATEX is a perfect tool to typeset nearly everything in a beautiful manner. Without any doubt, LATEX can typeset tables, but it is easy to produce bad tables with ugly lines and text touching the lines. This talk is intended to introduce how to typeset tables with LATEX on a beginners' level, mentioning some typographic aspects, showing some packages that help the author in formatting tables and concluding with how to typeset tables with page breaks. [This article was published in *TUGboat* 28:3, `http://tug.org/TUGboat/Articles/tb28-3/tb90hoeppner.pdf`.]

Gianluca Gorni and Stphane Matiz, Inserire equazioni LATEX in grafici di *Mathematica* [Including LATEX equations in graphics generated through *Mathematica*]; pp. 78–81

In this article we introduce a solution for creating graphics with the LATEX fonts. This solution is meant for users who create pictures in Mathematica to be included in LATEX documents. With a single command within the Mathematica front end

`TeXClipping[`*LATEX syntax*, *options*`]`

we get a graphical object `Graphics` for Mathematica: a simple set of polygons that gives the same impression as its corresponding font and that integrates perfectly in the *Mathematica* context.

Claudio Beccari, I font per le slide LATEX resuscitati [LATEX slide fonts revived]; pp. 82–87

(Published in this issue of *TUGboat*.)

Massimiliano Dominici, Utilizzo di caratteri TrueType con LATEX. Un esempio pratico: i *Fell Types* [TrueType fonts in LATEX, a concrete example: the *Fell Types*]; pp. 88–102

This paper explains how TEX can make the best use of the features of a TrueType font. For this purpose, the paper shows the installation of a collection of fonts, the *Fell Types*, full of nonstandard features that TEX can be taught to manage in a transparent way for the user.

Jean-Michel Hufflen, Guidelines for Bibliographical Citations in LATEX; pp. 103–110

After a short overview of the schemes used for bibliographical citations, we give some guidelines for using some packages of LATEX 2ε and bibliography styles of BibTEX in order to write *adaptable* citations, i.e., texts where switching one citation scheme to another is easy.

***The PracTEX Journal* 2007-3–2008-1**

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* 2007-3, August 2007**

Issue theme: Tools for TEX and LATEX users.

Francisco Reinaldo, From the Editor

The Editors, News from Around: Upcoming conferences in Pisa and Cluj (Romania); LATEX workshop in Berkeley

From the Readers, Feedback

Theresa Song Loong, The beginner's forest of LATEX

Images kept floating away, and keeping the style within one project both beautiful and consistent took up a lot of time. I have never been really content with a word processor. For a large project containing lots of math formulas, I assumed that learning to use LATEX would take as much time as trying to input the maths in a word processor. The final result was beautiful, but I was very wrong about how much time it would consume: using Greek fonts, making tables, using some packages, and trying to solve issues by reading incomprehensible package documentation (some of which didn't even explain how to use the package, only how it was coded) was very time-consuming indeed.

Antero Neves, A minha experiência em LATEX

[Article is in Portuguese.]

Arne Henningsen, Tools for collaborative writing of scientific LaTeX documents

Collaborative writing of documents requires a strong synchronisation among authors. This paper describes a possible way to organise the collaborative preparation of scientific LaTeX documents. The presented solution is primarily based on the version control system Subversion. The paper describes how Subversion can be used together with several other software tools and LaTeX packages to organise the collaborative preparation of LaTeX documents.

Arthur Buchsbaum and Francisco Reinaldo, A tool for logicians

turnstile is a LaTeX package that allows typesetting of the mathematical logic symbol, "turnstile", in all of the various ways it is used. This package was developed because there was no easy way in LaTeX to typeset this symbol in its various forms, and place expressions above and below the crossbar.

Charilaos Skiadas and Thomas Kjosmoen, LaTeXing with TextMate

This article discusses the TextMate text editor and its many capabilities that make working with LaTeX documents a lot easier. Some of its features include syntax highlighting, various methods for automatic insertion of text (such as the begin-end blocks in environments and automatic labels for section commands), lookup of labels and cite keys based on partial matches, as well as tools for dealing with large projects.

TextMate is designed with the user in mind, so it is easy to customize it to your needs. During its short lifetime (about two and a half years) it has gained many supporters and has become a very popular text editor for the Mac OS X platform, and especially among LaTeX users, as can be seen from the large number of LaTeX related questions on the TextMate mailing list.

In addition to this article, the first author's weblog can be used as a starting point for learning more about using TextMate for LaTeX:
`http://skiadas.dcostanet.net/afterthought`

Vinicius Provenzano, A LaTeX 2$_\varepsilon$ "Linux-like" environment on Mac OS X

Free and commercial LaTeX 2$_\varepsilon$ implementations for Mac OS X are available on the Internet. If you have always used a Mac, the best starting point is to download and install one of these systems. However if you have always used Linux and now find yourself in front of a brand new Mac OS X machine and have no time to learn new tools from scratch, your best option would be to use your familiar Linux applications on Mac OS X. This paper aims to show you how to install and configure a Linux-like LaTeX 2$_\varepsilon$ environment in Mac OS X, using Fink, teTeX and Kile.

Jérôme Laurens, Will TeX ever be WYSIWYG or the pdf synchronization story

Why can editing Plain, LaTeX or ConTeXt documents be such a pain? Of course, we can use dedicated text editors and environments that are very handy, but we are far from the efficient graphical user interface of a modern word processor. In this article, we will point out some of the reasons why TeX has no WYSIWYG (What You See Is What You Get) user interface and we will discuss the possible remedies. One of them is the pdf synchronization implemented in the pdfsync package. This technology will be explained, and we will see its benefits and its limitations. Finally, we consider routes towards a better user experience. Here, we are mainly concerned with LaTeX and pdf output.

Alexander Tsyplakov, TpX — a drawing tool for LaTeX

This article describes TpX, a lightweight, easy-to-use graphical editor for the Windows platform, presents guidelines for its use, and discusses some features and limitations.

Duvvuri Venugopal, Tools for creating bibliographic databases for use with BibTeX

By using BibTeX we can easily change the style of bibliography/references according to the style of the journal. But creating bibliographic databases for use with BibTeX is very cumbersome. This article describes the various software tools available for creating bibliographic databases easily, particularly for the Windows platform.

Uwe Ziegenhagen, LaTeX document management with Subversion

From the single-author composition of a bachelor's thesis to the creation of a book by a team, there are many occasions where version management of a document may be helpful. With the aim of overcoming the shortcomings of CVS (Concurrent Version System) the Subversion version control system was implemented.

In this article I will describe the Subversion setup on Windows and Linux systems, the elementary steps of document management and various LaTeX packages working hand in hand with Subversion.

Martin Scharrer, Version control of LaTeX documents with svn-multi

This paper describes how to use the Subversion software to version control your LaTeX files while also placing the current revision information in your document using the package `svn-multi` (v1.3 or later).

It covers all steps needed to set up and use Subversion, and to manage multi-file documents. Usage examples are provided for both basic and advanced features, to allow the reader to get the most out of the package.

David Walden, Travels in TEX Land: Fonts, self-publishing and another reason I like TEX

In this column in each issue I muse on my wanderings around the TEX world. Section 1 of this column describes some work I did organizing my experiences of using different fonts within TEX. Section 2 describes use of an external processor in combination with TEX. Section 3 gives an update on my self-publishing efforts using TEX.

The Editors, Ask Nelly: How do I create inline numbered lists the LATEX way?

The Editors, Distractions — Alea iacta est!

***The PracTEX Journal* 2007-4, December 2007**

Issue theme: Teaching LATEX and TEX.

Paul Blaga and Lance Carnes, From the Editor

The Editors, News from Around: Conferences in Pisa and Cluj (Romania); LATEX workshop in Berkeley; Helvetica — the movie

Paul Blaga, Teaching LATEX: Why and how?

We discuss some of the problems related to the process of learning LATEX and the opportunities presented by a LATEX course. We also propose a syllabus for such a course and briefly mention some of the LATEX books which, in our opinion, are suitable to be adopted as course material.

Vincent Verfaille, A new package for conference proceedings

The new confproc package is a simple and efficient solution to build conference proceedings. Built from scripts developed for the DAFx-06 proceedings, it deals with various aspects: layout issues, table of contents, index of authors, maybe a general bibliography, etc. It combines the pdfpages package (to include PDF papers), the hyperref package (to provide hyperlinks) and other packages; and it runs pdfLATEX.

Kumar M Senthil, LATEX tools for life scientists (BioTEXniques?)

LATEX has been a long time favorite of mathematicians and physicists. However, many packages are now available that have tremendously extended the capabilities of LATEX beyond routine typesetting and provide biologists new avenues to not only typeset documents, but also help in the visualization of membrane proteins and in the analysis of DNA or amino acid sequences by multiple sequence alignment. I will discuss with examples some of the LATEX packages and tools that are presently available for the biologists. Some scientific journals (for biological research) now accept (LA)TEX formatted manuscripts, although they are still a rarity. This article will provide references for those sources that might be helpful to prospective authors from life sciences who want to submit manuscripts in (LA)TEX format. This article is written from the perspective of a biologist who might be interested in creating better documents using LATEX & friends.

Rohit Vishal Kumar, Using LATEX for writing a thesis

LATEX has been successfully used for typesetting widely different document formats. However, the complexity of typesetting some commonly used documents acts as a deterrent for some people who would like to use LATEX for their work. Over the years, I have noticed students who come to LATEX eager in their anticipation of using LATEX, lose their enthusiasm midway, and revert back to using Word. In this article, I have tried to described my own experiences of typesetting a doctoral thesis using widely available packages, in the hope that students can see the ease with which LATEX can be used for complex work.

Wybo Dekker, The ctable package

The ctable package provides a `\ctable` command for typesetting table and figure floats. You will not need to type the usual nested begin...end sequences, as `\ctable` is a command, not an environment. `\ctable` takes only four arguments, but the optional first one may hold many *key=value* pairs and makes `ctable` very flexible and extensible. It uses Simon Fear's booktabs package for better vertical spacing around horizontal rules and it provides facilities for making table footnotes.

Nicola Talbot, Teaching LATEX for a staff development course

I taught LATEX at the University of East Anglia from 1997 to 2004 as part of the staff development course. In this article I will describe the headaches and lessons I learnt which helped me improve the course. This article is intended to assist those who are planning to teach LATEX in a practical environment.

S. Parthasarathy, Brevity is the soul of wit: How LATEX can help

This essay is about using "lists" in LATEX. Lists are very useful, in presenting material in a crisp

and compact form. This makes technical documents, less verbose, and easier to follow. The author hopes that this paper will make LaTeX enjoyable for more people.

Keith Jones, Writing your dissertation using LaTeX

The old adage, "You can lead a horse to water, but you can't make it drink," applies when trying to convince students to change to LaTeX for writing their thesis or dissertation. Students like to stay in their "comfort" zone and do not look favorably toward the work of learning a new software system. To date I have convinced one professor and two students to use LaTeX as their primary document formatting system.

Jonathan Fine, Interactive TeX training and support

It is today practical and helpful to provide TeX as a web service. This allows us a new approach to learning TeX.

Lapo Mori and Maurizio Himmelmann, Writing curriculum vitæ with LaTeX

This paper presents the tools that are currently available to prepare the curriculum vitæ with LaTeX with a critical analysis of packages and classes.

David Walden, Travels in TeX Land: Benefits of thinking a little bit like a programmer

In this column in each issue I muse on my wanderings around the TeX world. Section 1 of this column provides another illustration of the benefit of defining a few simple macros for a particular TeX project. Section 2 gives another example of using an external processor in combination with TeX. Section 3 gives another example.

The Editors, Ask Nelly: How do I write matrices in the text?

The Editors, Distractions — Music scores with LaTeX

The PracTEX Journal 2008-1, April 2008

Issue theme: LaTeXniques.

Yuri Robbers, From the Editor

The Editors, News from Around: TeX program updates; Bigelow introduces Grotesque; Day of LaTeX; Keming (?)

Lapo Mori, Writing a thesis with LaTeX

This article provides useful tools to write a thesis with LaTeX. It analyzes the typical problems that arise while writing a thesis with LaTeX and suggests improved solutions by handling easy packages. Many suggestions can be applied to book and article styles, as well.

Jim Hefferon, LaTeX goes with the flow

One advantage of TeX and friends is that they fit naturally into a work flow where there are many tools, each good at its own job. This paper gives an example involving a system for doing class evaluations online.

Ista Zahn, Learning to Sweave in APA style

Until recently I used Microsoft Word and clones such as OpenOffice to write academic manuscripts, as do most in my field. The standard software toolkit for many psychology professors and graduate students also includes SPSS for performing statistical analyses, and perhaps EndNote or similar reference manager software for generating bibliographies. These tools work, but my experience suggests that LaTeX-based solutions have significant advantages. This article describes how to use Sweave to write LaTeX documents in APA style, complete with results, tables, and figures generated by R.

Yogeshwarsing Calleecharan, Using BibTeX to produce customized layouts

Normal LaTeX and TeX usage does not require touching existing `.bst` files nor creating new ones. However, BibTeX offers several interesting commands which can be used to do many things apart from bibliography generation. In this article, it is demonstrated how customized layouts for a database can be created without much trouble.

David Walden, Travels in TeX Land: Another ornament for "thought breaks"

In this column in each issue I muse on my wanderings around the TeX world. I suggested at the end of section 2 of my last column that I would continue my investigation of colors and TeX in this issue. However, I was distracted for much of the period between issues by a health problem (now resolved). Thus, in this issue I only have time to touch *briefly* again on a topic from my column in TPJ 2005-4 — another ornament to use for "thought breaks".

The Editors, Ask Nelly: How do I find the files required to compile my document?

The Editors, Distractions — Spirograph with PSTricks

TUG Business

TUG financial statements for 2007

David Walden, TUG treasurer

This financial report for 2007 has been reviewed by the TUG board but has not been audited. It may change slightly when the final 2007 tax return is filed. As a tax-exempt organization, TUG's annual information returns are publicly available on our web site: `http://www.tug.org/tax-exempt`.

Revenue (income) highlights

Membership dues revenue was slightly up (at the end of December 2007 we had 1567 paid members, about 75 more than in 2006); conference income was substantially up; and interest income was up somewhat. However, product sales and contributions income were down more substantially (much of the decrease in contributions was because 2006 income included an atypical, one-time contribution of $5K). So altogether, revenue decreased 4 percent from 2006 to 2007.

Cost of Goods Sold and Expenses highlights

Payroll, office expenses, and *TUGboat* production and mailing continue to be the major expense items.

The shipping expense of the 2007 TEX Collection software was budgeted in 2006 (when, as it turned out, no software was shipped). However, we had significant postage/delivery expenses as we shipped *TUGboat* issues to many individuals who joined TUG after the normal mailings had been made. *TUGboat* expenses were nevertheless down in 2007 from 2006 because we produced three normal issues in 2007, and no extra-expensive EuroTEX proceedings as happened in 2006.

Overall, expenses are up about $3K in 2007 because of a modest cost-of-living increase in payroll and increased contributions made by TUG.

The bottom line

Subtracting 'Cost of Goods Sold' from 'Income', we are essentially flat from 2006 to 2007 — the savings in COGS mostly offsets the decrease in revenue. As expenses are up about $3K, the net income is down a little under $3K in 2007 — down $2,683, to be exact.

Typically we have a prior year adjustment that takes place early in the year to compensate for something that had to be estimated at the time the books were closed for the year on December 31. We had a small positive prior year adjustment in 2007 versus a larger negative adjustment in 2006, and year to year the difference is $2,244. Netting 'Prior Year Adjustments' with 'Net Ordinary Income', the overall net income for 2007 is a very positive gain of $15,408 (though down 2.8 percent from 2006).

Balance sheet highlights

Thus, for the second year in a row, TUG's end-of-year asset level is much higher than the previous year's level — up over $26K from 2006 to 2007.

The 'Committed Funds' come to TUG specifically for designated projects: the LATEX project, the TEX Development fund, and so forth. They have been allocated accordingly and are disbursed as the projects progress. TUG charges no overhead for administering these funds.

'Prepaid Member Income' is member dues that were paid in 2007 for 2008 and beyond. Most of this liability (the 2008 portion) was converted to 'Membership Dues' for 2008 on January 2008. The payroll liabilities are for 2006 state and federal taxes due January 15, 2007.

Summary

TUG was in essentially the same financial condition at the end of 2007 as at the end of 2006 and is in a relatively strong position overall. However, there is no fee increase for 2008 despite inflationary tendencies, and the TUG board is planning to increase direct TUG contributions (fund more TEX development) in 2008 as well as cover the cost of producing and shipping the software and another expensive EuroTEX proceedings in 2008 (already shipped as *TUGboat* 29:1). In other words, we will be using some of the 2006-to-2007 asset increase in 2008 to further our fundamental goals and benefits.

TUG continues to work closely with the other TEX user groups and ad hoc committees on many activities to benefit the TEX community.

TUG 12/31/2007 (versus 2006) Balance Sheet

	Dec 31, 07	Dec 31, 06
ASSETS		
Current Assets		
Total Checking/Savings	160,490	133,790
Accounts Receivable	254	395
Other Current Assets	1,327	
Total Current Assets	162,071	134,185
Fixed Assets	3,726	5,224
TOTAL ASSETS	**165,797**	**139,409**
LIABILITIES & EQUITY		
Liabilities		
Software Delay until 2007		6,500
Committed Funds	24,413	9,322
Prepaid member income	4,075	1,710
Payroll Liabilities	1,080	1,057
Total Current Liabilities	29,568	18,589
TOTAL LIABILITIES	**29,568**	**18,589**
Equity		
Unrestricted	120,820	104,972
Net Income	15,409	15,848
Total Equity	136,229	120,820
TOTAL LIABILITIES & EQUITY	**165,797**	**139,409**

TUG 2007 (versus 2006) Revenue and Expenses

	Jan - Dec 07	Jan - Dec 06
Ordinary Income/Expense		
Income		
Membership Dues	101,956	101,669
Product Sales	7,667	11,776
Contributions Income	5,423	11,376
Practical TeX Conference		2,909
Conference Classes		965
Annual Conference	6,827	-275
Interest Income	5,901	4,589
Advertising Income	230	370
Total Income	128,004	133,379
Cost of Goods Sold		
TUGboat Prod/Mailing	25,130	28,998
Software Production/Mailing	1,111	6,500
Postage/Delivery - Members	6,296	2,702
Conf Expense, office + overh	1,164	1,651
Member Renewal	335	
Copy/Printing for members	55	60
Total COGS	34,091	39,911
Gross Profit	93,913	93,468
Expense		
Contributions made by TUG	5,750	3,000
Office Overhead	11,653	12,229
Payroll Exp	59,863	58,622
Professional Fees	200	318
Depreciation Expense	1,498	1,667
Total Expense	78,964	75,836
Net Ordinary Income	14,949	17,632
Other Income/Expense		
Other Income		
Prior year adjust	459	-1,785
Total Other Income	459	-1,785
Net Other Income	459	-1,785
Net Income	15,408	15,847

TUG Institutional Members

Aalborg University, Department of Mathematical Sciences, *Aalborg*, *Denmark*

Aware Software, Inc., *Midland Park*, *New Jersey*

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

Banca d'Italia, *Roma*, *Italy*

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

Certicom Corp., *Mississauga*, *Ontario*, *Canada*

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*

MOSEK ApS, Copenhagen, Denmark

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*

TEX Users Group Membership Form 2008

Promoting the use of TEX throughout the world.

mailing address:
P. O. Box 2311
Portland, OR 97208-2311 USA

shipping address:
1466 NW Naito PKWY, Suite 3141
Portland, OR 97209-2820 USA

phone: +1 503-223-9994
fax: +1 206-203-3960
email: office@tug.org
web: http://www.tug.org

TUG membership rates are listed below. Please check the appropriate boxes and mail the completed form with payment (in US dollars) to the mailing address at left. If paying by credit/debit card, you may alternatively fax the form to the number at left or join online at `http://tug.org/join.html`. The web page also provides more information than we have room for here.

Status (check one) ☐ New member ☐ Renewing member

Automatic membership renewal in future years ☐
Using the same payment information; contact office to cancel.

	Rate	Amount
☐ **Regular membership** for 2008	$85	_____
☐ **Special membership** for 2008 You may join at this special rate if you are a senior (62+), student, new graduate, or from a country with a modest economy. Please circle accordingly.	$55	_____
☐ **Subscription** for 2008 (non-voting)	$95	_____
☐ **Institutional membership** for 2008 Includes up to eight individual memberships.	$500	_____
☐ **Don't ship any physical benefits** (*TUGboat*, software) *TUGboat* and software are available electronically.	deduct $20	_____
Purchase last year's materials:		
☐ *TUGboat* volume for 2007 (3 issues)	$20	_____
☐ TEX Collection 2007 DVD with proTEXt, MacTEX, TEX Live, CTAN.	$10	_____
☐ TEX Live CD 2007	$10	_____
☐ proTEXt CD 2007	$10	_____
☐ CTAN 2007 on CD	$15	_____
Voluntary donations (see `https://www.tug.org/donate.html`)		
☐ General TUG contribution		_____
☐ Bursary Fund contribution		_____
☐ TEX Development Fund contribution		_____
☐ CTAN contribution		_____
☐ TEX Gyre fonts contribution		_____
☐ LATEX 3 contribution		_____
☐ LuaTEX contribution		_____
☐ MacTEX contribution		_____
Total $		_____

Tax deduction: The membership fee less $35 is generally deductible, at least in the US.
Multi-year orders: To join for more than one year at this year's rate, just multiply.

Payment (check one) ☐ Payment enclosed ☐ Visa ☐ MasterCard ☐ AmEx

Account Number: _____ Exp. date: _____

Signature: _____

Privacy: TUG uses your personal information only to send products, publications, notices, and (for voting members) official ballots. TUG does not sell or otherwise provide its membership list to anyone.

Name _____

Department _____

Institution _____

Address _____

_____

City _____ State/Province _____

Postal code _____ Country _____

Email address _____

Phone _____ Fax _____

Position _____ Affiliation _____

Calendar

2008

May 15 – 16 Seventh annual Friends of St Bride Library Conference, "Seeking inspiration: Creative thinking around the design process", London, England. stbride.org/events_education/events

May 28 – 31 DANTE: Exhibitor at LinuxTag, Berlin, Germany. www.dante.de

Jun 24 – 28 SHARP 2008, "Teaching and Text", Society for the History of Authorship, Reading and Publishing, Oxford Brookes University, Oxford. ah.brookes.ac.uk/conference/sharp2008

Jun 25 – 29 Digital Humanities 2008, Association of Literary and Linguistic Computing / Association for Computers and the Humanities, Oulu, Finland. www.ekl.oulu.fi/dh2008

Jul 6 – Aug 2 Wells College Book Arts Center, Summer Institute, Aurora, New York. www.wells.edu/bkarts/info.html

Jul 15 – 20 TypeCon 2008: Punkt (10th anniversary), Buffalo, New York. www.typecon.com

---

TUG 2008 — TEX's 30th birthday
University College Cork, Ireland

Jul 20 Workshops

Jul 21 – 24 The 29th annual meeting of the TEX Users Group. www.tug.org/tug2008

Aug 11 – 15 SIGGRAPH 2008, "Evolve", Los Angeles, California. www.siggraph.org/events/s2008

Aug 12 – 15 Balisage: The Markup Conference, Montréal, Québec. www.balisage.net

Aug 20 – 25 Second International ConTEXt User Meeting, Bohinj, Slovenia. meeting.contextgarden.net

Sep 1 – 4 Book history workshop, École de l'institut d'histoire du livre, Lyon, France. ihl.enssib.fr

Sep 16 – 19 ACM Symposium on Document Engineering, São Paolo, Brazil. www.documentengineering.org

Sep 17 – 21 Association Typographique Internationale (ATypI) annual conference, "The Old · The New", St. Petersburg, Russia. www.atypi.org

Oct 4 – 5 Oak Knoll Fest XV, "Celebrating a 'Hot Metal Man'", honoring Henry Morris and his 50th anniversary in printing, New Castle, Delaware. www.oakknoll.com/fest

Oct 6 Journée GUTenberg & Assemblée générale, Paris, France. www.gutenberg.eu.org/manifestations

Oct 10 – 12 American Printing History Association 2008 annual conference, "Saving the History of Printing", Grolier Club and Columbia University, New York, New York. www.printinghistory.org

Oct 16 – 18 Guild of Book Workers, Standards of Excellence Annual Seminar, Toronto, Ontario. palimpsest.stanford.edu/byorg/gbw

Oct 25 – 27 The Sixth International Conference on the Book, "Save, Change or Discard: Tradition and Innovation in the World of Books", Catholic University of America, Washington, DC. b08.cgpublisher.com

Dec 8 – 10 XML 2008 Conference, Arlington, Virginia. xmlconference.org

---

2009

Jan 8 – 10 College Book Art Association Biennial Conference, "Art, Fact, and Artifact: The Book in Time and Place", University of Iowa Center for the Book, Iowa City, Iowa. www.uiowa.edu/~ctrbook/events/CBAA_conference.shtml

---

TUG 2009
University of Notre Dame, South Bend, Indiana

Jul 27 – 31 The 30th annual meeting of the TEX Users Group. www.tug.org/tug2009

Aug 31 – Sep 4 EuroTEX 2009, The Hague, The Netherlands. www.ntg.nl/EuroTeX2009

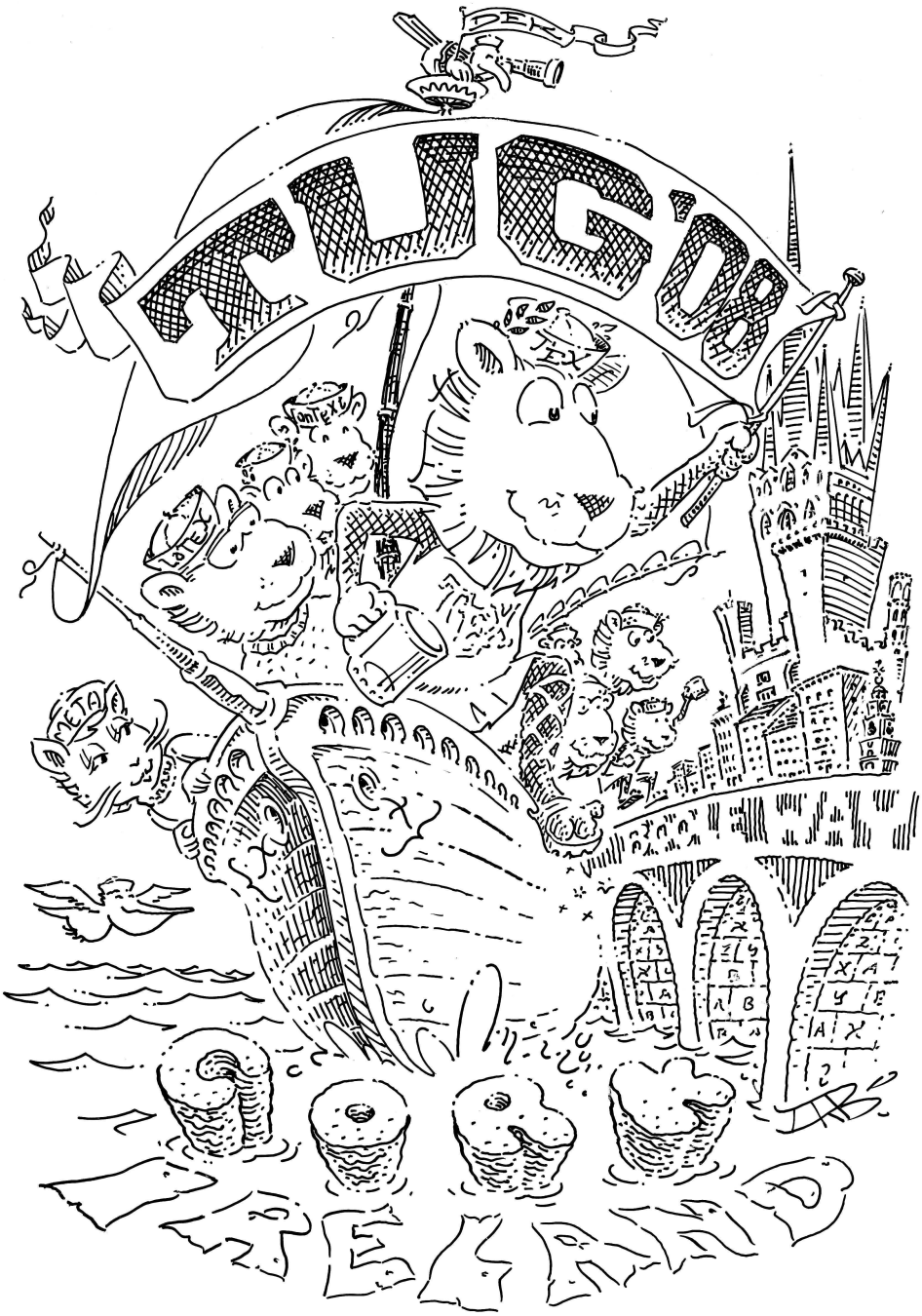
Status as of 15 May 2008

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.

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

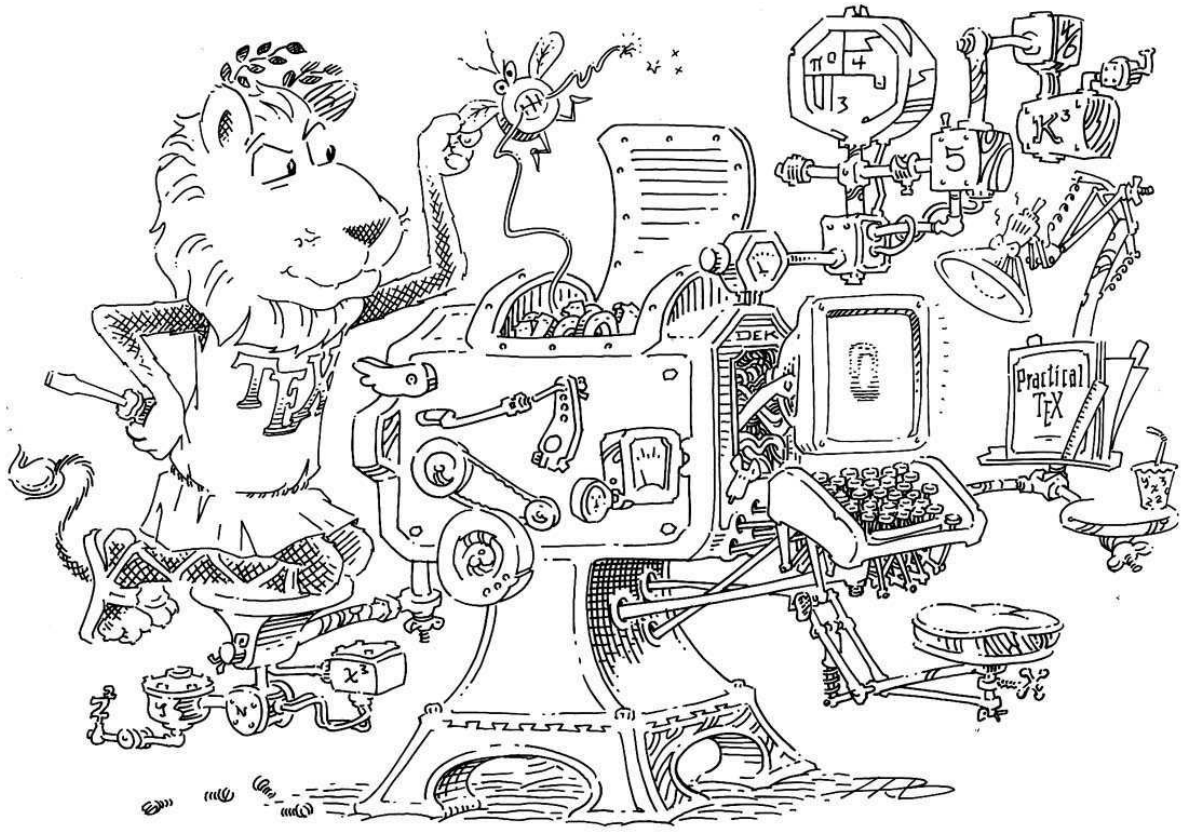
The information here comes from the consultants themselves. We do not include information we know to be false, but we cannot check out any of the information; we are transmitting it to you as it was given to us and do not promise it is correct. Also, this is not an official endorsement of the people 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`.

Martinez, Mercè Aicart
Tarragona 102 4^o 2^a
08015 Barcelona, Spain
+34 932267827
Email: `m.aicart (at) menta.net`
Web: `http://www.edilatex.com`
We provide, at reasonable low cost, TEX and LATEX typesetting services to authors or publishers worldwide. 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 29 (2008), No. 2

Table of Contents (ordered by difficulty)

Introductory

Intermediate

Intermediate Plus

Advanced

Contents of other TEX journals

Reports and notices

TUGboat

Volume 29, Number 2 / 2008

TUGBOAT Volume 29 (2008), No. 2

Table of Contents (ordered by difficulty)

Introductory

Intermediate

Intermediate Plus

Advanced

Contents of other TEX journals

Reports and notices