TUGboat

Volume 25, Number 2 / 2004

TEX Users Group

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

Memberships and Subscriptions

2004 dues for individual members are as follows:

- Ordinary members: $75.
- 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: `http://www.tug.org`.

TUGboat subscriptions are available to organizations and others wishing to receive *TUGboat* in a name other than that of an individual. Subscription rates: $85 a 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. For further information, contact the TUG office (`office@tug.org`) 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

+1 503 223-3960

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/`

Problems not resolved?

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

[printing date: June 2005]

The link between weaving and representations of letters on a computer screen can be seen very clearly by looking at how the weavers of Lyons wove words into their designs.

James Essinger
Jacquard's Web: How a hand-loom led to the birth of the information age (2004)

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP
EDITOR BARBARA BEETON

VOLUME 25, NUMBER 2 • 2004
PORTLAND • OREGON • U.S.A.

TUGboat

This issue (Vol. 25, No. 2) is the only regular issue of the 2004 volume year. Vol. 25, No. 1 was the Practical TEX 2004 conference proceedings, and the first publication for 2004 appeared in June 2004, the special non-*TUGboat* "preprints" of the TUG 2004 conference proceedings, subsequently published by Springer-Verlag. (For more information about the TUG'04 proceedings, see `http://tug.org/TUGboat/Articles/tb25-0`.)

TUGboat is distributed as a benefit of membership to all 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 may be 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

Suggestions and proposals for *TUGboat* articles are gratefully accepted and processed as received. We encourage submitting contributions by electronic mail to `TUGboat@tug.org`. Alternatively, please contact the TUG office.

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

As of the 2005 volume year, submission of a new manuscript will imply permission to publish the article, if accepted, on the *TUGboat* web site, as well as in print. So, if you have any reservations about posting online, please notify the editors at the time of submission. (Background: until now, it has been *TUGboat* policy to seek explicit permission for posting online, but we believe this has become unnecessary, leading primarily to articles never being posted, as well as being a time-consuming burden on *TUGboat* staff. For several years, no author has refused permission to post online, so it seems reasonable to now assume this permission by default.)

Other TUG Publications

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

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

TUGboat Advertising

For information about advertising rates and options, write or call the TUG office, or see our web page `http://tug.org/TUGboat/advertising.html`.

Trademarks

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

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

General Delivery

From the President

Karl Berry

TUG board and election

As I write this in May 2005, the postmark deadline for the first TUG election for president in many years is almost here, with Lance Carnes and myself as the candidates. We expect the election results to be known by the time of the Practical TEX 2005 conference in Chapel Hill, North Carolina this June.

No matter the outcome of the election, I am honored to have served as president these past two years, and am happy that TUG members actually have a choice in this election, instead of the president being selected by "default". Thanks to Lance for having the interest and inclination to get involved.

As with the last several election cycles, there were fewer candidates for the board than positions available. As president, I therefore appointed the incoming candidates immediately, as is traditional. I would like to welcome our two newest board members: Klaus Höppner and Dave Walden.

Klaus is also the vice-president of DANTE e.V. and so brings a wealth of cross-continental information and initiatives. We're very happy to have him join us, as the TEX user groups worldwide work more closely together than ever.

Dave is a long-time worker in computers, including a long stint at Bolt, Beranek, and Newman during the time of primary Internet development, as programmer, technical manager, and ending as general manager. He thus brings enormous management and organizational expertise to TUG, along with plenty of programming and writing skills.

More information about all the board members can be found in the election report elsewhere in this issue of *TUGboat*, and online at `http://tug.org/board.html`.

New TUG initiatives in 2004

First, TUG launched an online publication, *The PracTEX Journal*, with Lance Carnes as editor-in-chief. Two issues have been published to date and are available now on the TUG web site, at `http://tug.org/pracjourn`. As you might guess from the name, *TPJ* focuses on short, timely, and practical pieces, and thus complements *TUGboat*.

Many people contributed to making *TPJ* a reality; the web pages have the full list of the organizing board, as well as (of course) the authors, without whom there would be nothing to publish. Still, I'd like to especially thank Lance for his efforts in bringing this to fruition, and Dave Walden, for his extensive work writing the (Perl) program which generates the published web pages.

Second, another new feature on the TUG web site is the Interview Corner. This was conceived by Dave Walden (thanks again, Dave) as a way to record some of the community history, and get to know some of the individuals so important to TEX and TUG over the years. He's interviewed many notables already, including Robin Fairbairns, George Grätzer, and Christina Thiele. Dave welcomes feedback on the interviews, suggestions for future interviewees, and also other interviewers. Check out the web pages at `http://tug.org/interviews`.

Lastly, I'm happy to report that TUG and the WinEdt team have begun a program whereby TUG now offers WinEdt licenses at a substantial discount to TUG members, following a similar agreement between WinEdt and DANTE. My thanks to Steve Peter, Klaus Höppner, and the WinEdt folks for the idea and execution of this, and to Robin Laakso in the TUG office for taking on yet another task with enthuasism. More information and the order form are available at `http://tug.org/winedt`.

TUG futures

TUG's longstanding activities also continue: *TUGboat*, software, and conferences. With this issue, *TUGboat* will essentially be once again current, after several years of work catching up. The next issues, for the 2005 volume, will (barring disaster) be published in 2005. Barbara Beeton discusses this further in her editorial.

On the software front, work on TEX Live 2005 is proceeding. We will also distribute an update for proTEXt, the Windows distribution based on MiKTEX which we distributed in 2004 for the first time. Thanks to Thomas Feuerstack and Christian Schenk for making that possible.

And we've sponsored Practical TEX conferences in 2004 and 2005 in the United States, as well as the annual conferences (this year in historic Wuhan, China, `http://tug.org/tug2005`).

Despite these new and ongoing activities, our membership is down around 10% in 2004 (see the financial report in this issue for more details). If you are taking the trouble to read this, you are likely one of TUG's many long-time supporters — thanks. If TUG is to remain viable over the long term, clearly we must find ways to attract and retain more members; the overwhelming majority (around 80%) of

funding for TUG activities comes from membership dues.

In turn, this presumably means keeping TEX itself vibrant and growing. TUG is, after all, the TEX *users* group, an organization of, by, and for TEX users, not a big for-profit company or government institution handing down pronouncements about how things must be.

So if you have ideas for or interest in promoting or developing TEX and friends, or have thought of other projects useful to the community that TUG might undertake, please don't hesitate to contact the full board (board@tug.org) or myself. Thanks for your support, and happy TEXing.

◇ Karl Berry
president@tug.org

Editorial Comments

Barbara Beeton

New *TUGboat* submission and posting policies

Effective with the first issue of volume 26 (2005), there will be two changes in *TUGboat* policies for article submission and posting.

First, submissions to *TUGboat* will assume that the author agrees to posting of the submitted article on the TUG web site when the issue in which it appears is posted, subject to two restrictions:

- If the author specifically states that the article may not be posted, and provides the reason, this will be honored.
- If an article appeared previously in another publication, permission to post on the TUG web site will be requested from the editor(s) of that publication as well as from the author before posting.

Second, open posting of a published *TUGboat* issue will be deferred up to one year from the mailing date of that issue. However, immediate on-line access will be provided to members; the mechanism has not yet been implemented, but will be in place by the time the first issue of volume 26 is distributed, and members will be notified in due time.

Justin Howes, 1963–2005

We report with sadness the death on March 1, 2005, of Justin Howes, the typographer who developed the font, Founders Caslon, that was used to typeset the EuroTEX 2003 proceedings, *TUGboat* **24**(3).

Howes, who was born April 4, 1963, was devoted to typographic history, and actively sought to preserve the artifacts and archives of British typefounding. Thanks largely to him, the holdings of Stephenson, Blake Ltd, of Sheffield, the last major firm of this kind, were saved for posterity by the Type Museum in 1996; these included materials dating back to the Moxon era of the 17th century. For the past two years, Howes worked part time as curator of this collection.

He became attracted to the potential of the computer to aid in the preservation of type designs. Unlike many other digitized versions of old faces, Howes' rendering of Caslon was not only true to the original, but was implemented in several distinct design sizes.

Howes was about to embark on a six-month visit to the Plantin-Moretus Museum in Antwerp, Belgium, where he had looked forward to casting letters and working with their 16th and 17th century materials.

He died at his desk of a heart attack, aged 41.

An extensive obituary from the London Times can be found on-line at http://www.timesonline.co.uk/article/0,,60-1505298_2,00.html, from which much of the information in the present note is abstracted. Additional information about his work can be found at http://www.microsoft.com/typography/links/news.aspx?NID=4665.

John Seybold, 1916–2004

John Seybold, the father of computer typesetting, died on March 14, 2004, in Haverford, Pennsylvania.

Seybold became involved with publishing after World War II, during the era in which offset printing was beginning to replace the metal technologies. In 1963, he was introduced to an early implementation of computer hyphenation, in conjunction with paper tape control of an early phototypesetter. He became convinced that computers could do more than just hyphenation, and founded the Rocappi company (Research on Computer Applications in the Printing and Publishing Industries) to develop a system that could tackle the entire process of editing, manipulating and formatting text to produce "commercial quality" published materials.

In 1970, after selling Rocappi, Seybold undertook consulting, and in September 1971, he and his son Jonathan launched *The Seybold Report*, a newsletter that became the most reliable source of information on the computer publishing industry.

TEX was the subject of an extensive article in *The Seybold Report*, and Seybold organized a small gathering at Stanford in 1983 to investigate METAFONT

FONT, a gathering which I was privileged to attend. One whimsical product of this experiment was the letter "Knu", a compound of an uppercase "K" and lowercase "n"; the resulting glyph, sadly, appears to have been lost.

A brief biography and other memorabilia can be found on-line at `http://www.johnwseybold.com/bio.htm`.

Word Hy-phen-a-tion by Com-put-er

Frank Liang's Stanford Ph.D. dissertation has, with Frank's permission, been scanned and posted on-line for unlimited distribution. This work presents the hyphenation algorithm that is standard in TEX, and has been adapted for use with numerous languages.

The dissertation was scanned by Petr Sojka and his colleagues (to whom many thanks), and can be obtained via links on the page `http://tug.org/docs/liang/`.

Error in *TUGboat* 24:2 Zapfino article

The article "There is no end: Omega and Zapfino" by William Adams has in the upper right-hand corner of most right-hand pages a series of figures intended to be viewed by flipping the pages, spelling out the name of the font in an animation. Unfortunately, owing to a lapse in communication with the printer, the figures were cropped incorrectly, and the effect is not what was intended.

The article as posted on line has the correct, uncropped figures. Look for it via the issue contents: `http://www.tug.org/TUGboat/Contents/contents24-2.html`.

Historic LaTeX distributions

Ulrik Vieth has installed a collection of historic LaTeX distributions dating from 1983 on `ftp://ftp.tug.org/historic/macros/latex-saildart`. This collection includes LaTeX 2.0 for TEX 1.0 (released on 11 December 1983) and some even earlier versions. The material is based on archive tapes from Stanford's SAIL system.

Ulrik has long been interested in TEX history and is responsible for other collections as well. If you know of, or have, any material that isn't included in the historical archives on the TUG machine, but should be, let us know, and we will help you to connect with Ulrik.

The LaTeX Companion, 2nd edition

The second edition of *The LaTeX Companion* contains numerous examples illustrating the many packages described in the book. These examples are significantly revised from those that appeared in the first edition. The revised examples can be found at CTAN in `info/examples/tlc2`.

Some errors have already been found. These too are at CTAN, and also available at `http://www.latex-project.org/guides/tlc2.err`.

Addison-Wesley and the authors have started a bug contest — any mistake found and reported is a gain for all. A prize will be awarded every half year for 6 periods, in May and October, through May 2007. The eligible person who finds the largest number of bugs during each period will have free choice of any single computing book (no boxed sets or multiple volume offers) on the AW Professional web site, `http://www.awprofessional.com`. A person can receive at most one prize, ever; errors found by any of the authors do not count.

Start reading, and good luck.

techexplorer available once again

The techexplorer Hypermedia Browser, originally created by IBM, has been acquired and is now available from Integre Technical Publishing Co. as licensed, sponsored freeware. For details, see `http://www.integretechpub.com/techexplorer/`.

Central European Diacritics: *TYPO Magazine*

TYPO Magazine is a bimonthly, full-color magazine published in both Czech and English on topics related to typography, graphic design and visual communication. The September 2004 issue contains an interesting article on the design of central European diacritics.

Back issues are posted on line at `http://www.magtypo.cz/`; the cited issue is No. 10.

Extra time? Proofread for Project Gutenberg

The goal of Project Gutenberg is to make available, in electronic form, books that are out of copyright (published before 1924) in different languages. Proofreading and correction are accomplished by volunteers. More than 15,000 e-books have been made available to date.

If you have some free time, and wish to aid this effort, you can find information at `http://www.pgdp.net`.

◇ Barbara Beeton
American Mathematical Society
201 Charles Street
Providence, RI 02904 USA
bnb@ams.org

Tutorials

CTAN for Starters

Jim Hefferon

Newcomers to TEX can have trouble finding their way around. As with other community-supported projects, beginners can feel that only insiders or old-timers can get the tool to do its magic.

One of the secrets to TEX success is to know where on the Internet there are resources that you can use. A key TEX community resource is our archive. This article takes you through how to find and use this site.

1 On your mark ...

All TEX users should know: *if you need something, then the right place to look is the Comprehensive TEX Archive Network.*

CTAN is authoritative: if you need something TEX related that is out there, then chances are that you can get it in here. And, most of CTAN's holdings are freely available, so you can just pick them up and use them.

This article will take you around the site a bit, so that you can get an idea of what is here. Start by browsing to the top page: `http://www.ctan.org`.

2 Get ready ...

From the home page, click on the "Look through" link to get to `http://www.ctan.org/tex-archive`. There you see the CTAN's top-level organization, with a brief description of each branch. If you click around, you will get some sense of the great amount of material, and of the wide variety of material, that is available to you.

As a beginner, the first thing to get is a distribution — a collection of packages and programs, suitable for your computer platform, with what you need to start working. We have all of the major free and shareware distributions: TEX Live for Windows, Macintosh, and Unix, MiKTEX for Windows, gwTEX with i-Installer for Macintosh OS X, and teTEX for Unix and Mac OS X.

Go back to the "Look through" page and click on "systems" to go to where system-specific software lives. Click on the type of system that you have. For instance, if you work under Windows then you can follow the `win32` link (or the `texlive` link). One of the options there is `miktex`, and the material on that page tells you to install by reading what is in the `setup` directory. By following those directions you will get a complete TEX system on your computer.

3 Get set ...

After you've installed a distribution, you next need a tutorial. There is no substitute for a good book, but CTAN can still help you here, too. Go back to the top page and again follow the "Look through" link (`http://www.ctan.org/tex-archive`) to the top of the file structure.

Click on the "info" link to go to `http://www.ctan.org/tex-archive/info`. Here are many tutorials, and a great deal of other documentation.

Most people do their TEX work via the LATEX macro package, and one of the choices now on your screen is "lshort". Click on it to get to `http://www.ctan.org/tex-archive/info/lshort`, which contains the widely-recommended *The (Not So) Short Guide to LATEX 2ε* (the current version of LATEX is called LATEX 2ε). There are many translations there; select one and save or print it.

4 Go!

With that, you now have a full TEX system and enough documentation to do tremendous things.

5 Through the back straightaway ...

CTAN is not just for getting up to speed, it can also help you move ahead in your TEX work.

Imagine that you've used LATEX a bit and have gotten comfortable with the tutorial. A colleague sends you a file to use, but running it gives you an error message that your system cannot find `SIunits`. This package might not have come with your distribution (the distribution's builders try to balance completeness and size). However, CTAN has it.

From the top page `http://www.ctan.org`, take the "search" link to `http://www.ctan.org/search`.

In the first text box, type `SIunits` and hit Enter. You get a list of links, including a directory called `macros/latex/contrib/SIunits`. Click on the directory name to see what's there. You get a page listing the files.

Also on that directory page is a link to get the contents of the "entire directory". Click on it, and you will be offered the files from the directory, bundled up as `SIunits.zip` or `SIunits.tar.gz`. That's the right way to get the materials, so that you will not miss any.

Click on one of the links to get it to your machine (if you don't know which to use, get the `.zip`). You may get a page that asks you to select a mirror from a list. Many sites around the world generously

Editor's note: This article was first published in *The PracTEX Journal*, 2005(1), `http://tug.org/pracjourn`.

help out by offering the contents of CTAN to the public; you are seeing a list of these. Choose one from the list that says it offers the kind of archive that you want, `.zip` or `.tar.gz`, and you will be sent a cookie so that your browser can remember your preference in the future.

With that, you have the bundle on your computer containing the files that you want. What you need next is directions to install the material. CTAN can help you here, also. Back at the search page `http://www.ctan.org/search`, look for the "Frequently Referenced Links". One of these is to Robin Fairbairns's English language FAQ, `http://www.tex.ac.uk/faq`. One of the answers on that list, "Installing a new package", tells you just what you need to know.

You may want to bookmark the search page `http://www.ctan.org/search`; it's one of the most convenient ways to get at the information on CTAN.

6 Out of the final turn ...

Now you know how to get publicly available materials, if you know exactly what files you want. What if you instead need some particular feature, but don't know a specific name? As with the documentation, there is no substitute for a good book, but the search page can help.

Suppose that you need to work with your page footers. Go to `http://www.ctan.org/search` and use the "Search the Catalogue" box (the *Catalogue* is a large collection of TEX package descriptions). Enter `footer`. You get a page of links, one of which is `fancyhdr`, with the abstract "Extensive control of page headers and footers in LaTeX 2_ε". Also there is a link to the directory, so you can look through the documentation file.

Your distribution already has this package, so there is no need to download it. Nonetheless, the lesson here is that CTAN is useful for things other than getting materials; it is also a source of information on those materials.

7 Across the line

One thing that places an experienced person ahead of a beginner is an awareness of what resources to use to solve problems. For TEX users, CTAN is one of the most important resources.

8 In the circle with the leaders

As your TEX sophistication grows, you may well develop some software or documentation of your own. For instance, perhaps you are writing a thesis in TEX, and you find that none of the available thesis styles quite suit your university. You solve the problem by combining some packages from CTAN with some programming of your own to write a style that works.

When you do that, please consider contributing your work to CTAN. A link on the top page takes you to a page with instructions on how to upload. Typically, you only need make a `.zip` file with the software and enough documentation to help people get started using your work.

Contributions like this help us to build our community!

9 A note on places

CTAN is a network because it consists of a number of cooperating sites. This article consistently uses `http://www.ctan.org` URL's but you have other options, which may give you better network access.

The three core sites are `http://dante.ctan.org` in Germany, `http://cam.ctan.org` in England, and `http://tug.ctan.org` in the United States; this last is an alias for `http://www.ctan.org`. The three have different interfaces, but have the same holdings.

These three sites are active — they install newly uploaded material, etc. There are also many mirror sites that help out by just copying the content from a core site and then also offering the material to the public. Please use a mirror if you can; see the full list at `http://www.ctan.org/tex-archive/README.mirrors`.

The three core sites are sponsored by TEX user groups: DANTE in Germany, UK-TUG in England, and TUG in the US. There are many more user groups; see `http://tug.org/usergroups.html` for the complete list. If you find CTAN and TEX useful, please consider joining or supporting the user group best for you.

10 One more note: what shows

The CTAN team is working on some changes that may affect the look of the web site. Thus, in the future, some of the web interface details described here may change. Of course, we hope that these changes make the site even more useful to the TEX community.

◇ Jim Hefferon
St. Michael's College
Vermont, USA
`ftpmaint@alan.smcvt.edu`

\starttext: Practical ConTEXt

Steve Peter

Abstract

In this column, I introduce the reader to the ConTEXt macro package, showing a few practical examples along the way.

1 Introduction

Welcome to the first installment of the `\starttext` column. Together we'll explore the vast world that ConTEXt offers. If you don't already have ConTEXt installed on your system, head over to the Pragma web site at **www.pragma-ade.com**. You can get just ConTEXt, or a complete system with the underlying TEX distribution.

To get the most out of this column, you should have ConTEXt running on your system, and you should type in the examples as we go. You can use any editor that outputs plain text, such as emacs, vi, Text Edit, or Notepad. Don't use a word processor like Word or OpenOffice Writer. There are also complete TEX editing environments like TeXShop on Mac OS X and TEXnicCenter on Windows that allow you to edit your files, run TEX, and view the output from within a single application.

If you don't have TEX, don't have a computer, or are just curious about ConTEXt and would rather read than type, I've also supplied some illustrations.

Let's get started!

2 Hello, World!

Since ConTEXt is a TEX macro package, we'll follow the standard workflow by first entering the text of our document into a plain text file, interspersed with commands that tell TEX to do something with the text (e.g., make it bold, or format it like a footnote). Then we run TEX on the file, and finally we look at the beautiful output.

So fire up your favorite text editor and enter the following:

```
\starttext
Hello, World! This is \ConTeXt.
\stoptext
```

Editor's Note: This article is reprinted, with additions, from *The PracTEX Journal*, 2005(1), `http://tug.org/pracjourn`.

The body of your document is enclosed in a `\start`—`\stop` pair. `\starttext` handles various setup details for you. Save the file as `document.tex`.

If you've used any variety of TEX before, the next step is slightly different, so watch out. (And be amazed!) To run this document through TEX, we'll use `texexec`, a front end script that greatly simplifies life. More about that in a bit. For now, just type the following in a shell window (if you're not using an editing environment as discussed above):

```
texexec document
```

You should now have a new file, `document.dvi`, in your directory. You can view the file with, e.g., xdvi on Unix, TEXniscope on Mac, or yap on Windows. You can convert the dvi (DeVice Independent) file to pdf with the `dvipdfm` utility, or use `texexec --pdf`. Or perhaps your machine may be configured to run pdfTEX automatically (as my machine is). In that case, simply open the resulting `document.pdf` file. Whether dvi or pdf, the result should look something like this:

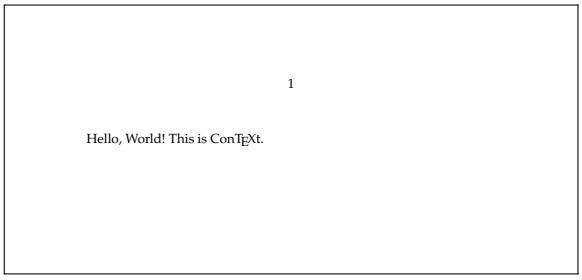


Figure 1 Your first ConTEXt document!

The page number at the top tells us that this is a default ConTEXt document, and not simply a Plain TEX one. For our first experiment, let's put the number into the footer.

Setting up something like the location of the page number is done with a `\setup` command in ConTEXt. Don't worry right now about the exact form of the command. We'll go over them in much greater detail in a later column. For now, to put the number in the footer, add the following line to the top of your document, before the `\starttext`. Run it through `texexec` and look at the file produced.

```
\setuppagenumbering [location=footer]
```

Now the folio is in the footer.

Text of any length is usually subdivided. Let's put in some sections. This time *after* `\starttext`,

put the line

```
\section{First section}
```

Add a few more `\section`s with some text. We'll need them for the next section. To get a bunch of text quickly, try `\dorecurse{20}{\input knuth \par}`.

3 texexec

I mentioned before that `texexec` greatly simplifies life. Why is that? Well, typesetting is a complicated business, and TEX frequently has to collect information on one run to use in a later run. For example, let's add a table of contents. Just after `\starttext`, add:

```
\completecontent
```

But how does TEX know what page the second `\section` is on until after it has typeset the document? The answer, of course, is that it doesn't. TEX gathers up information from all the `\section`s you have in the document and writes that information to an auxiliary file. Normally, you have to then run TEX a second time so that TEX can read that information in and set the table of contents. (And if the TOC is long, it will push everything down, meaning that you have to rerun TEX again!)

Sometimes you find yourself rerunning TEX needlessly just to make certain there aren't any unresolved references. But `texexec` changes that. It automatically reruns TEX as many times as necessary, so you can go refill your coffee.

4 Fun and fancy

Just to whet your appetite, let's take a quick look at a couple of fancier things ConTEXt can do. We'll go into details in future columns. I realize these are a bit of a jump from the basic formatting considered in the other sections, but since we're just setting out, I thought I'd give you a glimpse of some really fancy stuff.

To maintain high typographic standards (cf. the discussion, for example, in Robert Bringhurst, *The Elements of Typographic Style*) you often have to align text, graphics, etc., to a grid, and your text should maintain a consistent position on the baseline grid. Add this to the top of your document and process it with `texexec`.

```
\setuplayout[width=middle,location=middle,
     grid=yes,marking=color]
\moveongrid[both]
\showgrid
```

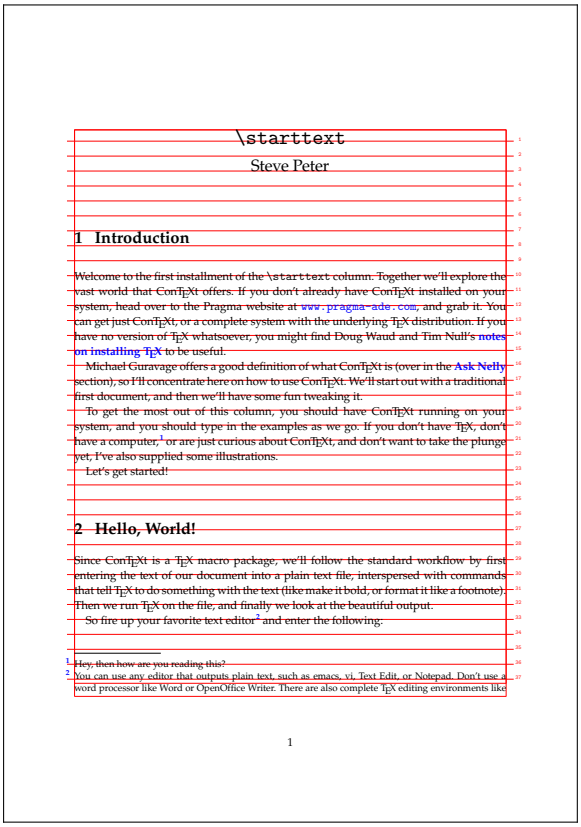


Figure 2 The matrix? No, it's the grid.

If you've ever tried to do that with Plain TEX, or even LATEX, you know what a pain it can be. However, ConTEXt does it easily, and even shows you where the grid is, so you can debug troublesome documents.

After all TEX is, when you get down to it, a programming language. That means at some point you'll need to debug your documents. The grid feature is but one of several nice visual debugging tools provided with ConTEXt. For another one, add this to the beginning of your document to gain a view of how TEX puts boxes and glue together:

```
\showmakeup
```

In figure 3 we can see the bounding box for the E in TEX, along with the negative kerns, shown as the thicker boxes near the base of the E.

One more useful visualization command shows you the layout on the page of your text block, mar-

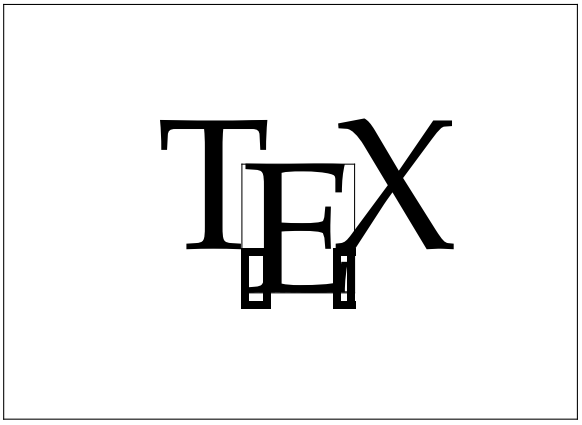


Figure 3 What TEX might look like to TEX

gins, headers and footers:

```
\showframe
```

For more on this aspect of visual debugging, see Hans Hagen's paper in *TUGboat* vol. 19, no. 3.

You don't need a fancy commercial page layout program to set crop marks or do imposition. Put these lines before `\starttext`, run `texexec`, and watch the magic! (Figure 4.)

```
\setuppapersize [A7][letterpaper]
\setuparranging [2*2,rotated,doublesided]
\setuppagenumbering [
     alternative=doublesided]
\setuplayout [margin=0pt,width=fit]
\setupbackgrounds [text][text][
     background=screen]
\setupcolors [state=start]
\setuplayout [location=middle,
     marking=color]
\setuptolerance[tolerant]
\setupbodyfont [palatino,6pt]
```

You can even produce a negative by replacing the first line above with

```
\setuppapersize [A7][letterpaper,
     negative,mirrored]
```

I won't show it here, due to obvious ink costs. But if you ever need to generate film output, this is a lifesaver.

5 Links

I hope you've enjoyed this first look at ConTEXt.

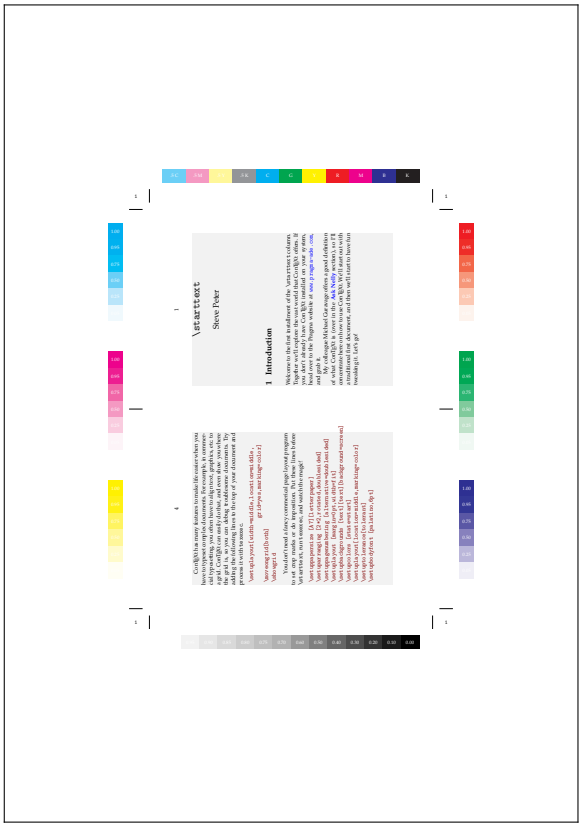


Figure 4 Imposition with ConTEXt

There are numerous topics we haven't addressed yet, such as cross references, hyperlinks, indexes, MetaPost figures and other graphics, and ConTEXt's incredible support for pdf trickery.

There's a lot of information out there and plenty to explore. Start with the documentation on the Pragma web site (`www.pragma-ade.com`).

For examples, check the ConTEXt wiki at `contextgarden.net` and work your way through Bill McClain's excellent page detailing ConTEXt at `home.salamander.com/~wmcclain/context-help.html`. Last, but certainly not least, you can jump into the never-ending discussion on the official mailing list at `www.ntg.nl/mailman/listinfo/ntg-context`.

Join us here in future issues of *TUGboat* for more on the practical use of ConTEXt.

◇ Steve Peter
Beech Stave Press,
310 Hana Road,
Edison, NJ 08817
`speter@beechstave.com`

Virtual fonts — a tutorial

Thomas A. Schmitz

Lots of information on TEX's virtual fonts can be found on the web and in books (e.g., Knuth's very own "Virtual Fonts: More Fun for Grand Wizards", `http://www.ctan.org/tex-archive/info/virtual-fonts.knuth`). However, there doesn't seem to be a step-by-step tutorial for non-wizards like myself. I have experimented with virtual fonts recently. It took me a while to understand the basics, so I thought that other people might find it useful to hear about this and avoid some common mistakes.

1 Basic facts about virtual fonts

Let's start by discussing two immensely useful things that virtual fonts can do.

First, they can remap characters within the same font. If you have a font `foo` with files `foo.pfb` and `foo.tfm`, you can make a virtual font `foobar` that will be identical to `foo` but print an "A" whenever you have a "B" in your TEX file.

This may sound absurd at first, but there are cases where it is useful. For instance, some fonts offer alternative forms for letters. With the help of a virtual font, you can remap the letters and thus switch to these alternative forms without changing your TEX source.

The second use for virtual fonts is much more common: Given a font `foo`, you can create a virtual font `foobar` that will include some characters from a second font, say `bar`. This is often used to include old-style numerals or additional ligatures that are not provided by the normal font.

2 Copying the font files

So, let's begin. We assume that we will be using two PostScript fonts, `foo` and `bar`. Usually, for each of these fonts, we will have two files `foo.tfm` (resp. `bar.tfm`) and `foo.pfb` (resp. `bar.pfb`) and nothing else, so we need to create a virtual font (`.vf`) file from scratch. I didn't find it mentioned anywhere that this is not only possible, but even fairly easy.

We'll perform these operations on the command line in a working directory, such as `/tmp`. So the first step is to copy `foo.tfm` to this directory:

```
cd /tmp
cp /PATH/TO/foo.tfm .
```

(Watch the trailing period, it's necessary!)

3 Create a human-oriented property list

The file `foo.tfm` is a binary file, in a format that TEX can read. If we want to edit it, we will have to convert it to a so-called "property list" file (typically given the extension `.pl`), which is a plain text file that can be read by humans. We will be using tools that come with any complete TEX installation. From the command line:

```
tftopl foo foo
```

(Yes, that's right: we have to type `foo` twice!)

4 Open the property list

We now have a new file `foo.pl` which contains all the information about the font that TEX needs. Open it in your favorite text editor. If you're editing in a non-Unix environment, such as Windows or Mac OS X, make sure that your editor is set to use Unix line endings, unless you know for certain that your TEX utilities don't mind. The first few lines will read like this:

```
(FAMILY TEX-FOO)
(FACE F MRR)
(CODINGSCHEME FONTSPECIFIC + TEX TEXT)
(DESIGNSIZE R 10.0)
...
(LIGTABLE ...
```

If there is a line `(CHECKSUM O ...)`, delete it; it will be regenerated later.

5 Editing the property list

In order to generate a virtual font, we need to modify this file. First, we have to tell TEX which fonts our new virtual font will be referring to. Let's say they are `foo.tfm` and `bar.tfm` — needless to say, both have to be installed and functional in your TEX installation.

As a first step, we will create a virtual font that will remap some characters within foo. So just before the line starting with `(LIGTABLE`, add this:

```
(MAPFONT D 0
   (FONTNAME foo)
   (FONTDSIZE R 10.0)
   )
```

The `FONTDSIZE` of `foo` is found from the `(DESIGNSIZE R 10.0)` line above; all we have to do is copy this information.

6 Remapping a character

Now let's scroll down in this file. The `LIGTABLE` (containing information about ligatures and kerning) will end with two lines

```
(STOP)
)
```

After this, the section with information about all the defined characters in the font will follow, probably starting something like this:

```
(CHARACTER O 0
   (CHARWD R 0.674)
   (CHARHT R 0.726)
   )
```

TEX itself only cares about the dimensions of characters, as stored in the `.tfm` file, when doing the typesetting; it essentially leaves room for an empty box with these dimensions. The actual characters (the visible "glyphs") are put into these boxes only when the final PostScript or PDF output is made.

```
(CHARACTER C A
   (CHARWD R 0.747)
   (CHARHT R 0.747)
   )
(CHARACTER C B
   (CHARWD R 0.739)
   (CHARHT R 0.726)
   )
```

TEX will be using the box described as here, so we want the box for "B" to have the dimensions of the box for "A". Hence, the first thing to do is copy the dimensions of "A" into "B". Then the section should look like this:

```
(CHARACTER C A
   (CHARWD R 0.747)
   (CHARHT R 0.747)
   )
(CHARACTER C B
   (CHARWD R 0.747)
   (CHARHT R 0.747)
   )
```

Next (and this is the magic of virtual fonts) we tell TEX that it should remap "B" to "A". Just before the closing parenthesis of `CHARACTER B`, we insert a new section, so that "B" will look like this:

```
(CHARACTER C B
   (CHARWD R 0.747)
   (CHARHT R 0.747)
   (MAP
      (SETCHAR C A)
      )
   )
```

One of the things that can be a bit confusing about these property lists is that (apart from the numbers and the 26 letters of the English alphabet) characters are referred to by "octal numbers". If you want to know what character corresponds to what octal number, you can have a look at the tables created by testing the font (section 9).

7 Saving the file

That's it! We have modified the font description; now we need to generate the binary files for TEX to use. The next step is extremely important: **save the file to a different name**.

In our case, let's say we call the new virtual font `foobar`. The base name doesn't much matter, but the extension should be `.vpl`; so let's save to `foobar.vpl`.

8 Generating the binary files

Back to the command line. We now run a program that will convert `foobar.vpl` into two new files, `foobar.tfm` and `foobar.vf`:

```
vptovf foobar.vpl
```

This will not only do the conversion, it will also check whether the `.vpl` file is in good order. It is very picky about the right indentation level and parentheses; if there is a problem it will give the exact line number. So if you get errors, just go back and edit `foobar.vpl` again.

`vptovf` may also tell you that it had to "round some units"; that's OK.

9 Installing the new font

So now we should have `foobar.vf` and `foobar.tfm`. The next step is to copy both files into the right place. I would suggest you create your own `texmf`-branch in your home directory, for instance, under `~/Library/texmf`, or `~/` (depending on your local setup as defined in `texmf.cnf`). For the sake of our example, we'll use the former:

```
cp foobar.tfm ~/Library/texmf/fonts/tfm/
cp foobar.vf ~/Library/texmf/fonts/vf/
```

(You will have to create these directories if they don't exist yet.)

Since we're only using characters from within a single font (`foo`), we don't need to fiddle with any "map files". When the final output is made, only font `foo` will be needed, which was already functional.

Before embarking on a long journey with this new virtual font (say your 1200-page thesis that is due in two weeks), let's test it on its own:

```
cd ~
pdfetex testfont
```

`pdfetex` will respond something like this:

```
This is pdfeTeX, Version 3.1415...
...
(/usr/local/.../plain/base/testfont.tex
Name of the font to test =
```

We now type the name of our font:

```
foobar
```

and `pdfetex` will respond:

```
Now type a test command (\help for help):)
*
```

We give the command:

```
\table
```

pdfetex will come back with another asterisk, and now we're done:

```
\end
```

If all goes well, a file testfont.pdf will be created with a table showing that font foobar does not have a letter "B", but twice the letter "A" — which is just what we wanted.

If your TEX distribution includes the ConTEXt format (see http://tug.org/pracjourn/2005-1/peter/), you can also create a very nice colorized table. Make a file test.tex like this:

```
\starttext
\showfont[foobar]
\stoptext
```

Then, run this file through ConTEXt:

```
texexec --pdf --nonstopmode test.tex
```

You'll get a table with all the glyphs; every cell will indicate the decimal, hexadecimal, and octal value of the glyph (very handy for editing property lists).

10 Adding a second font

Now let's work on including further characters, like old-style numerals or ligatures from font bar. We go back to our working directory and delete the old file foobar.vpl. Don't worry, we'll create it again:

```
rm foobar.vpl
vftovp foobar foobar foobar
```

We do this because vftovp will automagically include one important piece of information; every character description will now look like this:

```
(CHARACTER C A
   (CHARWD R 0.747)
   (CHARHT R 0.747)
   (MAP
      (SETCHAR C A)
      )
   )
```

That's already not bad, but to mix glyphs from two fonts, we have to say which font to use in every instance. So in an editor, we perform a "find and replace" that will find every instance of (MAP and replace it with

```
(MAP
   (SELECTFONT D 0)
```

In Emacs, it is possible to include the line break in the replace pattern. (I don't know how other editors can handle it.)

Now add the information about the second font, just as described in section 5:

```
(MAPFONT D 1
  (FONTNAME bar)
  (FONTDSIZE R 10.0)
  )
```

To get the DSIZE of bar, we can again just convert bar.tfm into bar.pl, open this file and look into the first lines. Or, if you want to be fancy:

```
tftopl `kpsewhich bar.tfm` | grep DESIGNSIZE
```

(watch the "backticks", those are single opening quotes!).

11 Including glyphs from a second font

Now look for the section containing the numerals. It should start like this:

```
(CHARACTER C 0
   (CHARWD R 0.514)
   (CHARHT R 0.628)
   (CHARDP R 0.1)
   (MAP
      (SELECTFONT D 0)
      (SETCHAR C 0)
      )
   )
```

Again, the first thing to do is copy the dimensions of CHARACTER C 0 from bar.pl to foo.pl. Then the new step, to use bar instead of foo: replace (SELECTFONT D 0) with (SELECTFONT D 1). Repeat for all the other numerals.

We then follow the same procedure as before: generate the tfm/vf pair (section 8) and copy these files to the right directories (section 9). Now TEX and friends will look at the new tfm/vf, and take the numerals from font bar, and everything else from font foo. Again, test the results!

12 Post-install

After you have edited your virtual font, you can discard the .vpl file. If you ever want to edit your font again, you can recreate it by copying both the .vf and the .tfm into the same directory and running

```
vftovp foo foo foo
```

Have great fun and feel like a great wizard!

13 Further reading

http://www.cl.cam.ac.uk/users/rf/pstex/index.htm

http://homepage.mac.com/bkerstetter/tex/fonttutorial-current.html

http://zoonek.free.fr/LaTeX/Fontes/fontes.html (in French)

◇ Thomas A. Schmitz
Institut fuer Klassische und Romanische Philologie
Universitaet Bonn, 53113 Bonn
Germany
thomas.schmitz@uni-bonn.de

Typography

Typographers' Inn

Peter Flynn

1 Devil in the Details

One of the hallmarks of publication-quality typesetting is that it is correct in the fine detail. Not just in positioning, spacing, balance, weight, and other niceties of layout, but in thre key areas: speling, punk chew asian and consistency. Few people have the head for detail required of a typographer, but most ordinary readers are quite capable of telling poorly-set work from well-set, even if they cannot put their finger on what is actually wrong with it.

If you can't spell, find someone who can. There is no shame in admitting to a spelling problem: it's very common. Spellcheckers can be useful, but they tend to be error-prone on complex text unless you spend a long time training them.

If you can't punctuate, follow your publisher's rules; buy a copy of *Eats, shoots, and leaves* [4]; or if you're self-publishing, see Figure 1.

If you're by nature inconsistent, it will probably show, so hire a proofreader.

Beyond these three, there are three further levels at which attention to the details of formatting can be applied:

Looking acceptable. It's not hard to get a degree of regularity sufficient to pass muster with the average reader, or even your pointy-haired boss. LaTeX will almost always get the positioning right for major structural blocks like section headings, lists, and paragraphs. It may not be the most elegant, but if it's consistent and readable, and doesn't interfere with the sense for the reader, it's acceptable. You can even get this far with a word processor, if you're feeling masochistic, but see Figure 3. At this level, it's probably OK for *office use* or as *drafts*.

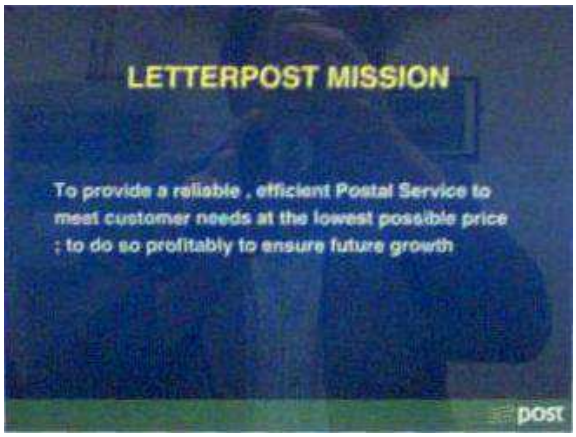
Looking 'right'. This is a cultural thing. LaTeX's defaults look right to an American, because it grew up there. The extensive language customization available in the babel package includes many typographic tweaks, but as far as I'm aware there is nothing affecting font size or vertical spacing: these you have to add yourself.[1] A similar requirement applies to other

[1] It would be useful to produce an equivalent set of layout default changes on a cultural (linguistic? national?) basis. Several LUGs have already done this for their own constituencies, so perhaps this could be extended.

These are guidelines that I have found useful in the absence of a formal set of rules. Some are cultural and need adapting as appropriate. Books of rules exist in many cultures (the *Chicago Manual of Style* [5]; the *Guide* of the Modern Language Association [2]; the various volumes of the *Duden* [1]; or any of the successors to *Hart's Rules* [3]), but these are sometimes slow to reflect cultural changes, and may mislead users into perpetuating an inappropriately antique style. Typists are taught to splatter their work with unnecessary punctuation — try to avoid this temptation.

1. Use punctuation sparingly.
2. Always use a space *after* closing punctuation unless another punctuation sign follows, in which case a `\thinspace` is appropriate. Don't bother typing multiple spaces: LaTeX will adjust the spacing.
3. Never use a space between a word and the punctuation which belongs after it (see Figure 2!) unless required by the cultural style.
4. Use the apostrophe correctly:
   (a) use it when something belongs to someone (Flynn's Rules = the rules belong to me);
   (b) avoid it with simple plurals (Pizzas €5.99) or a number (1940s);
   (c) use it without the extra 's' when Rule 4a applies but the word already ends in an 's' (Jones' Pizzas are better);
   (d) use it where there's a letter missing from the word (there's = there is; don't = do not);
   (e) avoid it when personal possessives already end in 's' (yours, his, theirs).
5. Use a full point at the end of a sentence.
6. Use a comma between phrases of a sentence only if there is a shift in meaning or emphasis.
7. Use a colon between two related but distinct (or opposing) thoughts in a sentence.
8. Use a semicolon between items in a list when they all form part of a greater whole (as the sub-list at Rule 4 above) and use a full point after the last item (assuming it's the end of the sentence).
9. Only ever use one exclamation or question mark at a time.
10. Never abbreviate unless you're short of space (exception: personal titles like Dr. and Ms.).
11. Never use full points in acronyms or abbreviations (IBM not I.B.M.) unless you're trying for that 1940s effect.
12. Be consistent with single quotes and double quotes: if you use double quotes for speech, use single quotes for quotations within speech.
13. If a sentence ends with a URI, separate the full point with a `\thinspace` so that novices don't think it's part of the Web address. If it comes at the end of a paragraph, consider omitting the full point entirely.

Figure 1: Rough Guide to Punctuation



Not only have they put a space *before* the comma in the first line, but the space before the semicolon has permitted a linebreak which they clearly don't see as being wrong! [Irish Postal Service mission statement, displayed in every post office.]

Figure 2: Wrong spacing for punctuation

typographic defaults, like the use of 1º instead of 1st for the ordinal.[2] At this level, a document is probably *publishable*.

Being invisible. The objective of typographic design is to help the author communicate ideas to the reader without getting in the way. Unless you are explicitly trying for special effects (common in advertising, for example, where almost anything goes to attract attention), the niceties of typography should recede into the woodwork or blend into the wallpaper. The reader should be unaware that any special effort has gone into the setting. Extra attention to detail can help achieve this, ironing out the remaining inconsistencies and minor infelicities, by adding manual micro-adjustments here and there to create that smooth, even look that makes a document easy to read and does not cause the reader to stumble over some unexpected bullet, font, or oddity of spacing. But this can take a lot of additional time, and the nature of the job should indicate whether it is worth it or not (and it comes naturally to some people, like the editors of *TUGboat*). By this stage, your typography has become *invisible*.[3]

[2] And in passing I can't avoid repeating that the use of a superscripted ordinal in Anglo-American typography is a Victorian relic, obsolete since before WW1, and unkindly reintroduced by word processors. Avoid it (it can be turned off — with some difficulty — in word processors) and use lining lowercase (1st) instead.

[3] That is, only your fellow ~~conspirators~~ compositors will notice what you have done.

If your organization is joined at the hip to Microsoft Word, you can still use LaTeX to create PDFs by starting your documents like this:

```
\documentclass[12pt]{article}
\usepackage[margin=1in]{geometry}
\usepackage{pslatex,sectsty,parskip}
\setcounter{secnumdepth}{0}
\allsectionsfont{\sffamily}
\makeatletter
\renewcommand{\maketitle}{%
   \section{\@title}%
   \subsection{\@author}%
   \subsubsection{\@date}}
\makeatother
```

It takes a little more effort to tweak lists into looking as ugly as Word's default, but it's possible.

Figure 3: Faking it for Word

I am aware that some of the guidelines in Figure 1 conflict with some received wisdom and I would welcome comments.

2 The Atlantic Divide

The *TUGboat* editors reminded me during the writing of this piece that the differences between Anglo and American typography still cause authors and publishers some difficulty. The first thing the readers will comment on is the additional comma placed in inline lists in the US, so that the (UK) 'apples, pears and bananas' becomes (US) 'apples, pears, and bananas'.

It *looks* as if there should be a semantic difference here, but there isn't: to an American the UK usage imples that pears and bananas are to be taken as a group because there's no comma: to the BritEng reader, the US usage makes it look as if bananas are some kind of afterthought. Again, consistency is the watchword. If you're using a programmable system like XSLT or LaTeX's paralist environment, it is even possible to omit the commas in a list and make the macros do the work.

The trailing period is another bugbear. I used one above after 'bananas' but the Modern Language Association (all stand and uncover, please) demands that the full point go *inside* the quotes even when it's not a part of the quotation! In a discussion on style and punctuation this is probably misleading, but it is the normal US convention — which *TUGboat* and many other technical works flout.

Increased use of email and text messaging has probably led to a closing of the divide, but I remember feeling distinctly uneasy at the idea of 'busing' children to school (I pronounced it 'bewsing' at first

sight, and had to ask what it meant) where 'bussing' would have been normal to me — if anything can be said to be normal about verbing nouns.

3 Comeuppance

It's always good to see the engineer hoist with his own petard, so I suppose I had it coming to me for the series of rants on 'reversed quotes' (Typographers' Inn, *ad nauseam*). It's still a pet hate, largely because it looks so silly, but it has become a shibboleth among designers: you can tell one who knows what she is doing by her avoidance of it.

I thought the earliest example I had seen was in the 1970s volume I of the late and much missed Spike Milligan's war autobiography, where I put it down to someone messing around with filmset matrices.

Last Sunday, irreverently gazing at the beautifully engraved tablets on the walls of my city's cathedral during the Nine Lessons and Carols, I spotted a memorial to a soldier of the Great War, killed in what was then called the Soudan, which had two quotations both in double reversed quotes. Presumably the engraver felt it was more symmetrical.

The permanence of engraving on stone cannot be underestimated: it lasts for thousands of years, far longer than any print or type. With luck it will outlast that other horror of the word processor, the automated apostrophe-becoming-an-opening-quote, which silently turns '94 into '94 because it is counting odd and even occurrences regardless of any preceding space, and thinks this is the start of reported speech.

This and the ordinal[2] are really good grounds for ditching the word processor and using LaTeX ...

References

[1] [Duden Editors]. *Die deutsche Rechtschreibung*. Brockhaus, Mannheim, 23rd edition, 2004.

[2] Joseph Gibaldi. *MLA Style Manual and Guide to Scholarly Publishing*. Modern Language Association, New York, NY, 2nd edition, 1998.

[3] R.M. Ritter and Horace Hart. *Oxford Guide to Style*. OUP, Oxford, 28th edition, 2002.

[4] Lynne Truss. *Eats, Shoots & Leaves: The Zero Tolerance Approach to Punctuation*. Profile Books, London, 2003.

[5] University of Chicago Press Staff. *The Chicago Manual of Style*. University of Chicago Press, Chicago, IL, 15th edition, 2004.

◇ Peter Flynn
University College, Cork, Ireland
pflynn@ucc.ie
http://imbolc.ucc.ie/~pflynn

Philology

Philological facilities for the Coptic script

Claudio Beccari and Cristiano Pulone

Abstract

Since 1995 some Coptic fonts by Serge Rosmorduc have been available on CTAN, along with minimal support for using them with LaTeX. We have extended them a little bit and added some support for philological typesetting, including hyphenation patterns and a small collection of macros for the ease of the philologists.

1 Introduction

Thanks to Serge Rosmorduc, since 1995 one Coptic font has been available on CTAN together with the font description file necessary for its use with LaTeX.

Rosmorduc provided the METAFONT description with file `copte.mf`; apparently he obtained the contour descriptions by tracing some fonts very similar to those that appear in a hieroglyphic dictionary [2]; one line of that source METAFONT file says:[1]

```
{limn output Sep 24 17:59:49 1995 from
imageto output Sep 24 16:54:15 1995}
```

but nothing else is said about the source images he operated on. He put his fonts in the public domain with a generic sentence, but an auxiliary file of his bundle contains the whole specification of the Free Software Foundation Licence.

We therefore felt free to add and modify Rosmorduc's files, by changing names and giving him full credit for the original work he had done. His work continues to be at least 80% of the new files. In particular his approach has been almost completely maintained in this sense: his tracing algorithm gave him the Bezier nodes and control points, therefore his METAFONT description is explicit, not a parameterized algorithmic one as we are accustomed to from the Computer Modern font source files; on this point you may see [5] for further information.

On the other side the fonts whose pictures have been traced by Rosmorduc have a very interesting appearance, since they give the impression of being stroked with a quill pen or some other old handwriting instrument of that sort.

Since Coptic fonts appear mostly in Christian liturgical texts we added some symbols that frequently occur in such texts. Nevertheless, philolo-

[1] `limn` and `imageto` are two early programs for tracing font contours; they are part of the GNU Font Utilities.

gists today dedicate most of their attention to para-Christian texts, especially Gnostic ones, as those in the renowned Nag Hammadi Library. Such signs as ⲻ ⲽ ⸗ ⁕ are specific to ritual texts, not necessarily strictly Christian ones; ⳿ and ☥ are of evident Christian origin, but they frequently appear in pagan ritual texts, Gnostic cosmologies, etc. For the philologist's ease we also added other glyphs that are in use in their texts.

We decided also to collect all the signs on the first page of the font table; in other words, we used only the first 128 slots of the font table. We did not care much about the encoding; Rosmorduc himself had in mind a philologist writing critical texts, not a theologian writing a whole text in Coptic, thus requiring numbers, punctuation, extra signs, etc. We are aware that there is an effort among the clergy in the Coptic Church trying to define a common encoding scheme (see [3], for example). On the other hand, the Unicode standard allocates the unique Coptic signs in slots `0x03E2` through `0x03EF`, while apparently the other symbols derived from Greek share the same positions of the Greek letters.

For reasons of compatibility we retained the ligatures defined by Rosmorduc, so that a text originally written to be typeset with the `copte` fonts can be processed with our fonts `copto` (ordinary upright font) and `copti` (inclined font), obtaining the same output except for a possible inclination.

In the end, the ordinary Coptic font turned out as shown in Table 1.

We also provide the font definition files, and, most important of all, we provide Type 1 versions of the fonts. So there should be no difficulty in producing fine documents in PostScript or PDF format.

2 The Coptic font

As we said in the introduction, Rosmorduc's fonts were obtained by tracing the images of Coptic glyphs taken from some source where they had suitable dimensions. The METAFONT specifications are in terms of contour Bezier nodes and control points, although the latter are specified indirectly.

The font design size is declared to be 10pt and everything inside the METAFONT files is specified in terms of a unit u declared to be one tenth of the design size, therefore nominally $u = 1\,\mathrm{pt}$. In practice the heights, widths and depths of all characters were as the tracing algorithm had determined; virtually all of them were distinct. No *x_height*, *quad_width*, etc., were defined, but it turned out that the average *x_height* of the various glyphs was significantly larger than the corresponding dimension of comparable fonts; in particular, it was almost 25% larger than the corresponding height of the CM fonts.

We decided to modify the font metric dimensions and to specify the other missing font dimensions that define the font-dependent TEX units.

We modified several glyphs; the most significant modification was the one concerning the letter "shima" whose upper stroke protrudes far out of the glyph bounding box, so that it would not butt against the ascenders or the next right character, specifically with "lauda", as in the word ϭⲗⲟϭ.

We added some special symbols, such as

ⲻ ⲽ ⳿ ☥ ⸗ ⳾ ⳽ ⳼ ⳻ ⳺ ⸚

and redefined the ligature table to cope with new glyphs and their ligatures, while preserving Rosmorduc's ligatures, for compatibility reasons. Probably in this respect more work should be done, but the result appears acceptable.

3 The font keyboard mapping

We use the word *transliteration* instead of encoding, because we had the philologist in mind; that is, a person who is writing critical texts in Coptic, but using a "Latin" keyboard. In fact, we made all our experiments with the Italian keyboard which lacks some important ASCII characters, but does have various others that are missing from a "normal" US keyboard.

For example, we had to define the grave accent. The Italian keyboard lacks the key with the grave accent (or back tick); we used the apostrophe key instead, on the assumption that the acute accent is seldom if ever used in Coptic. On the other hand, we defined a macro `\°` because the `°` sign is on the Italian keyboard and the macro appears less obtrusive when inserted into the source file. We provided the `\0` alias for those who use keyboards without the `°` sign.

The correspondence between the Latin and the Coptic signs is as shown in Table 2. It is based on the correspondence of the "sounds" or of the "shapes" or ... on the availability of a free key!

Notice that in Table 2 we make no attempt to spell out the names of the Coptic letters; there are several naming conventions that depend on the Western language of those who named them. At the web site `http://www.copticchurch.net` there is a short outline regarding Coptic fonts with their names. The point is that the Coptic letters should carry a phonetic value, but those who know how to read and write this language do not agree on their pronunciation, therefore the phonetic transliteration is not unique. Nevertheless, those who master the Coptic language can perfectly understand Table 2.

	’00	’01	’02	’03	’04	’05	’06	’07	’10	’11	’12	’13	’14	’15	’16	’17	
’000	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	"00
’020	16	17	18	19	20	¨ 21	– 22	23	24	25	26	27	28	29	30	31	"10
’040	32	·: 33	:· 34	ψ 35	Ψ 36	37	38	39	Θ 40	ξ 41	Ξ 42	φ 43	Φ 44	- 45	· 46	ⳉ 47	"20
’060	ⳉ 48	49	ⳉ 50	ⲧ 51	ⲧ 52	ⲣ 53	ⳉ 54	55	ⲟ 56	ⳑ 57	- 58	ⲭ 59	Ⲭ 60	⸗ 61	ⳒⳒ 62	ⳓ 63	"30
’100	64	Ⲍ 65	Ⲃ 66	Ⲥ 67	Ⲗ 68	Ⲅ 69	Ⲏ 70	Ⲙ 71	Ⲏ 72	Ⲓ 73	ⲏ 74	Ⲕ 75	Ⲗ 76	Ⲩ 77	Ⲏ 78	Ⲟ 79	"40
’120	Ⲡ 80	Ⲋ 81	Ⲣ 82	83	Ⲧ 84	Ⲩ 85	ⲇ 86	Ⲓⲓ 87	Ⲭ 88	Ⲯ 89	Ⲍ 90	[91	92	] 93	94	95	"50
’140	` 96	ⲁ 97	ⲃ 98	ⲥ 99	ⲍ 100	ⲉ 101	ⲏ 102	ⲙ 103	ⲏ 104	ⲓ 105	ⳋ 106	ⲕ 107	ⲗ 108	ⳒⳒ 109	ⲛ 110	ⲟ 111	"60
’160	ⲛ 112	Ⲋ 113	ⲣ 114	115	ⲧ 116	ⲩ 117	ⳍ 118	ⲱ 119	ⲭ 120	ⲯ 121	ⲍ 122	123	124	125	126	127	"70
	"00	"01	"02	"03	"04	"05	"06	"07	"08	"09	"0A	"0B	"0C	"0D	"0E	"0F	

Table 1: The ordinary upright Coptic font

4 The Coptic fonts in Type 1 format

As mentioned previously, we produced our Coptic fonts in Type 1 PostScript format.

We followed the near-standard procedure of tracing large-scale raster fonts made with METAFONT by means of `mftrace` [7] and pipelining the output to `pfaedit` (now `fontforge` [8]).

We did not try to create an encoding vector with any kind of names for the Coptic letters, not even trying to “copy” them from existing encoding vectors. Our fonts are rather non-standard, so their use is confined to (LA)TEX use only.

5 The Coptic font description file

Since we produced two basic shapes for the Coptic font family, we had to produce a new LATEX font definition file, `lcopcoptic.fd`, substantially different from the single-shape one by Rosmorduc.

In addition to the obvious point of declaring two shapes, instead of just one, the main changes are the following:

1. The font encoding was named `LCOP` in accordance with the recommendation of the LATEX3 team that any non-standard encoding should start with the letter `L`, for “local encoding”.
2. The font is loaded with a default magnification of 0.83. This scales the Coptic font to have a closer match between its x-height and the one of the surrounding text in Latin characters.

As with the original font by Rosmorduc, our fonts come in one size, although the METAFONT source files may be invoked on the fly by modern TEX systems so as to generate the raster files at the desired magnification. The heavy strokes of these fonts cope well with shrinking, but we think they become too black when enlarged too much, as seen in titles. In any case, we produced the PostScript Type 1 versions of these fonts so that a single source is used at all magnifications and PostScript or PDF documents can be well typeset, easily readable on screen and perfectly printable on paper.

6 Coptic hyphenation

We produced also a pattern file for the hyphenation of the Coptic language; it was “hand made” since we were not able to find either a word list or a Coptic dictionary specifying hyphenation points.

We worked on and implemented grammar rules [6] based on open and closed syllables, similar to the ancient Greek and Italian rules. While typesetting a master’s thesis on some ancient Coptic texts, we entered the words into a word list without hyphenation points and checked the hyphenations by means of a little LATEX program implementing V. Eijkhout’s `\printhyphens` macro [4].

The present patterns appear to hyphenate correctly all the words on the word list (a few hundred). Some possible hyphenation points may have been missed, but this is not a real inconvenience. (Every time the `patgen` program is used to create new patterns or analyze existing ones, the statistics of the missed hyphens are output. This is useful information when `patgen` is used to create or to modify hyphenation patterns from a hyphenated word list, but is of minor importance when analyzing existing patterns.)

In fact, typographical hyphenation does not necessarily coincide with grammatical hyphenation, even though, of course, it must not violate grammatical rules. Typographical hyphenation fulfills two main purposes: (a) allowing the typeset text to be broken into lines and justified without ugly white gaps, and (b) keeping the reader comfortable in reading broken lines. For the second purpose it may be desirable to refrain from certain hyphen-

Latin	Coptic	Latin	Coptic
a	ⲁ	A	Ⲁ
b	ⲃ	B	Ⲃ
c	ⲥ	C	Ⲥ
d	ⲇ	D	Ⲇ
e	ⲉ	E	Ⲉ
f	ⲫ	F	Ⲫ
g	ⲅ	G	Ⲅ
h	ⲏ	H	Ⲏ
i	ⲓ	I	Ⲓ
j	ⲧⲁ	J	Ⲧⲁ
k	ⲕ	K	Ⲕ
l	ⲗ	L	Ⲗ
m	ⲙ	M	Ⲙ
n	ⲛ	N	Ⲛ
o	ⲟ	O	Ⲟ
p	ⲡ	P	Ⲡ
q	ⲕⲟ	Q	Ⲕⲟ
r	ⲣ	R	Ⲣ
t	ⲧ	T	Ⲧ
u	ⲩ	U	Ⲩ
v	ⲫ	V	Ⲫ
w	ⲱ	W	Ⲱ
x	ⲭ	X	Ⲭ
y	ⲯ	Y	Ⲯ
z	ⲍ	Z	Ⲍ
8	ⲑ	81	Ⲑ
ks	ⲝ	KS, Ks	Ⲝ
ps	ⲯ	PS, Ps	Ⲯ
dj	ⲋ	DJ, Dj	Ⲋ
hj	ⲧⲁ	HJ, Hj	Ⲧⲁ
tj	ⲧ	TJ, Tj	Ⲧ
h1	ⲏ	H1	Ⲏ
h2	ⲏ	H2	Ⲏ

Table 2: Correspondence between Coptic and Latin signs or sign sequences on a Latin keyboard

command	example	output
\H	\H	ⲏ
\h	\h	ⲏ
\=	\={me}	ⲙⲉ
\"	\"	¨
\"i	\"i	ï
\"u	\"u	ü
\’	\’e	è
\’m	\’m	ⲙ̀
\’n	\’n	ǹ
\°[a]	\°	-
\0	\0	-
\+[b]	\+	≠
\pont	\pont{c}	ⲥ̇
\trepun	\trepun	·:
\threedots	\threedots	·:
\trepund	\trepund	:·
\sic	\sic e.\=nk	ϵ.n̄k [sic]
\dubious	\dubious{anokpe}	ⲁⲛⲟⲕⲡⲉ
\barretta	\barretta{dj}	ⲋ
\Asterisk	\Asterisk	*
\Crux	\Crux	☩
\crucicula	\crucicula	☩
\iesus	\iesus	ⲓc̄
\djois	\djois	x̄c̄
\xcr	\xcr	x̄cp̄
\xc	\xc	x̄c

[a] Besides the sign - this command and its alias \0 introduce a discretionary break after the short hyphen.

[b] This double inclined dash mark inserts an unbreakable point within a word, but it does not inhibit hyphenation in the remaining part of the word.

Table 3: New commands with the Coptic fonts

ations; this is certainly so in Italian, and therefore we followed our Italian tradition.

7 Macros for Coptic philologists

We completed the Coptic bundle with a `coptic.sty` file containing some useful macros in order to easily typeset Coptic source `.tex` files for critical texts.

The style file obviously provides a command and an environment, `\textcoptic` and `coptic` respectively, for typesetting the marked text using the Coptic alphabet and hyphenation. There is also a command `\textlatin` for inserting some text in the Latin alphabet. For compatibility reasons the code fragment "`coptic`" may be substituted by "`copte`" or "`copto`", so old documents, the source files of which were composed with the original Rosmorduc files or with our alpha versions of the package, are still usable, with no need to correct the commands and the environments.

It must be noted that the loading order of the packages `fontenc` and `coptic` makes a difference. If the former precedes the latter, `coptic` remembers the correct Latin encoding; if the latter precedes the former, `coptic` remembers the default encoding, presumably `OT1`, before the subsequent `fontenc` package changes the Latin encoding. Therefore a little care should be exercised when loading packages, or the `\textlatin` might yield unexpected results.

The specific Coptic language macros that are introduced with the package are collected in Table 3. Some of them operate on arguments, others are freestanding. The diaeresis accent may be set on every letter, but special commands are defined for

the cases when the letters are ⲓ and ⲩ, to use the special accented glyphs already present in the font. Similarly, the grave accent has special glyphs when the letter is ⲉⲉ or ⲛ.

It is a well-known problem that accent macros interrupt what TEX considers to be a word; in general, they inhibit subsequent hyphenation in the word. By resorting to special characters and the advanced LATEX 2_ε composite symbol command definitions it is possible to address the special symbols directly, thus allowing hyphenation and letting TEX work with the possible ligature and/or kerning properties of the characters involved.

Finally, we note that the `coptic` package is compatible with the `teubner` package, so that some synergy can be exploited between the two. In practice, typesetting critical texts in Coptic almost always implies citing numerous Greek references and text samples, possibly from the same ancient periods, so that all the facilities available with `teubner`, that are not directly connected with the Greek language, can be quite useful.

8 Conclusion

In preparing the modified Coptic fonts, from the original work of Serge Rosmorduc, and all the related files for typesetting critical texts in Coptic, we think we have made a first attempt to extend the present situation; but the actual approval of this work may come only from those scholars and Coptic clergy that use this alphabet and this language.

We did not experiment with `ledmac` [9] simply for lack of time; we suppose that the `coptic` and the `ledmac` packages should be compatible. If there are any, they may in fact be between `teubner` and `ledmac` and these possible bugs will be examined for the next release of `teubner`.

We are very grateful to the TEX users who have been so patient to use our material and submit constructive criticism. We will continue working on the refinement of this bundle in order to make it more useful to the Coptic experts.

As a concluding display, see in Figure 1 a sample text typeset with the Coptic fonts of this article; the versicle marks were obtained by means of the `teubner` package facilities. If upper and/or lower philological marks are present it is advisable to spread out the typeset lines a little bit; in Fig. 1 they were processed with a spread factor of 1.5.

Wien K 8304 (Rainer, AN 201)

ϯⲱⲣⲕⲉ ⲉⲣⲟⲕ ⲉⲉⲡⲟⲟⲩ ⲱ ⲥⲣⲟⲩⲫⲟⲥ ⲡⲁⲩⲩⲉⲗⲟⲥ ⲉⲧⲧⲏⲩ ϯ ⲉⲭⲉⲙ ⲧⲉⲭⲱⲣⲁ ⲛⲩⲏⲉⲉⲉ ⲭⲉⲕⲁ̣ⲥ ⲉⲕⲉⲡⲱⲣⲩ ⲛⲛⲉⲕ- ϯ ⲧⲉⲙⲥ ⲉⲭⲉⲙ ⲉⲉⲁ ⲉⲉⲁ (sic) ⲛⲓⲉⲉ ⲉⲧⲉⲣⲉⲡⲓⲥⲏⲁⲩ ⲛⲃⲁⲣⲱⲧ ϯ ⲛⲥ̣ⲏⲧⲩ ⲩⲁⲛⲧⲉⲩⲕⲧⲟⲩ ⲡⲉⲉⲁ ⲛⲧⲁⲩⲉⲓ ⲉⲃⲟⲗ ⲛⲥ̣ⲏⲧⲩ ⲛⲓ ϯ ⲉⲓⲃ̣̈ⲧ ⲉⲉⲡⲡⲉⲉⲉⲛⲧ ⲡⲉⲉⲉⲥⲓⲧ ⲉⲉⲛⲑⲁⲗⲁⲥⲥⲁ ⲉⲩⲱⲡⲉ ⲉⲩ- ϯ ⲧⲟⲉⲉⲥ ⲥ̣ⲁⲛⲕⲁⲥ ⲉⲕⲉⲟⲧⲱⲛⲥⲩ ⲉⲃⲟⲗ ⲉⲩⲱⲡⲉ ⲉⲩⲥⲏⲛ ϯ ⲥ̣ⲉⲉⲉⲉⲁ ⲛⲧⲟⲉⲉ ⲉⲕⲉⲕⲧⲟⲩ ⲉⲡⲉⲓⲉⲉⲁ ⲡⲡⲧⲁⲩⲭⲓⲧⲩ ⲉⲉⲡⲉⲣ- ϯ ⲧⲣⲉⲡⲕⲁⲥ̣ ⲧⲁⲭⲣⲟⲩ ⲥ̣ⲁⲣⲟⲩ ⲉⲉⲡⲉⲣⲧⲣⲉⲧⲡⲉ ⲉⲣⲥ̣ⲁⲓⲃⲥ ⲉⲣⲟⲩ ϯ ⲁⲧⲱ ⲉⲉⲡⲉⲣⲧⲣⲉ̣ⲗⲁⲁⲧ ⲛⲉⲉⲉⲧⲟⲛ ⲩⲱⲡⲉ ⲛⲁⲩ ⲡⲧⲏⲣⲩ ϯ ⲁ̈ⲓⲟ ⲁ̈ⲓⲟ ⲧⲁⲭⲏ ⲧⲁⲭⲏ ⲧⲁⲭⲏ

Figure 1: A sample typeset with the `copto` font

of hieroglyphic characters, Coptic and Semitic alphabets, etc., New York, Dover Publications, 1978.

[2] *The Coptic Standard Character Code (CSCC)*, `http://www.copticchurch.net/coptic_fonts/`

[3] Eijkhout, Victor, "The bag of tricks", *TUGboat* 14(4), 1993, p. 424.

[4] Hall, Timothy, "The METAFONT approach: Implicit, relative, and analytical font design", *TUGboat* 24(2), 2003, pp. 200-205.

[5] Mallon, Alexis, *Grammaire copte*, Beirut 1904, 1926.

[6] Nienhuys, Han-Wen, *mftrace — Scalable fonts for Metafont*, `http://www.xs4all.nl/~hanwen/mftrace/index.html`

[7] Williams, George, *fontforge*, `http://fontforge.sourceforge.net/`. Originally named pfaedit.

[8] Wilson, Peter, *ledmac — A presumptuous attempt to port EDMAC, TABMAC and EDSTANZA to* LATEX, present in any distribution of the TEX system when the ledmac package is installed.

◇ Claudio Beccari
Politecnico di Torino, Turin, Italy
`claudio.beccari@polito.it`

◇ Cristiano Pulone
Università di Bologna, Bologna, Italy
`c_pulone@tin.it`

RyDArab — Typesetting Arabic mathematical expressions

Azzeddine Lazrek

Abstract

TEX was adapted to handle passages of Arabic script some years ago, via packages such as ArabTEX or Ω. The package RyDArab, presented herein, extends these to handle mathematics in an Arabic presentation. That means expressions with specific symbols flowing from right to left, according to the Arabic writing, as they can be found in Arabic mathematical handbooks.

1 Overview

Although TEX [3] was designed according to the conventions of English and Western languages, it is also able to handle passages of Arabic script [4]. Systems such as ArabTEX, by K. Lagally [5], and Ω, by Y. Haralambous and J. Plaice [2], allow generating documents with passages in Arabic or some other language using the Arabic script. Even though many Arabic mathematical texts present mathematical expressions as they are in English or in French texts, a large number of Arabic handbooks display mathematical expressions using specific symbols in a writing flowing from right to left. Arabic mathematical document processing has been discussed in [10] and [11].

The RyDArab system presented here is designed for generating Arabic texts including such mathematical writing. The system consists of a set of TEX macro packages, some additional extensions and a family of symbol fonts. It will run under the Plain TEX or LATEX [6] formats. The present paper was typeset with this package.

This paper is organized as follows: in section 2, we show how to load the package. In section 3, we present some package options. In section 4, we go over commands that can be used to typeset Arabic mathematical expressions and show some examples. Of course, some problems are still to be solved. In particular, some compatibility questions are under investigation. There are still open questions that are outside the scope of this paper.

Throughout the present paper, we speak about "Arabic mathematical" texts, documents, expressions, and so on. Of course, mathematics is unique; it has nothing to do with any national specificity. When we use this appellation, it refers only to the way in which mathematics is presented in most common Arabic writing.

Editor's note: This article originally appeared in *Die TEXnische Komödie* 2/2001, and is reprinted here by permission, with many improvements by the author.

2 Preamble

To use RyDArab to generate Arabic mathematical expressions, first load the system by putting `\input rydarab` (for Plain TEX) or `\usepackage{rydarab}` (for LATEX) in the preamble of your document.

The ArabTEX package or Ω has to be loaded first. That will be the system for typesetting the textual component of the document.

3 Options

The commands defined in RyDArab are prefixed with the initials "`am`" (for Arabic mathematical). This helps to distinguish RyDArab's commands from the basic commands of TEX, LATEX, or other packages. The other part of the command names derive from the corresponding TEX or LATEX commands.

The following options are offered as commands or options of the package:

arabmath to begin an environment where Arabic mathematical expressions are generated (e.g. $\bar{\mathfrak{C}}_{\vee}$). This is the default.

latinmath to begin an environment where mathematical expressions are in their Latin form (e.g. $\sqrt{j}$).

warabnum for using the standard western Arabic digits ($\{0,1,2,3,4,5,6,7,8,9\}$). These digits — known as Arabic digits — are used in North Africa. This is the default.

earabnum for using the eastern Arabic digits ({٠, ١, ٢, ٣, ٤, ٥, ٦, ٧, ٨, ٩}). These numerals — known as Hindi digits — are used in the Middle East.

alpwithoutdots for using the alphabetic symbols without dots. This is the default.

alpwithdots for using the alphabetic symbols with dots.

funwithdots for using abbreviations representing elementary functions with dots. This is the default.

funwithoutdots for using abbreviations representing elementary functions without dots.

If these options are specified in the preamble of the input file, they work for the whole document. If they are specified at the beginning of an environment, in mathematical mode, the option is valid only through the end of the environment.

4 Commands

In addition to the basic set of TEX commands, there are new commands and commands resulting from some transformations of similar commands in TEX.

All these work in math mode only, either in display or inline environments.

The commands are listed below in an Arabic mathematical environment. The resulting text will appear in the frame following or in front of the command.

4.1 Inversion of direction

`\amrl{`*expr*`}` inverts the direction of writing in order to generate an Arabic mathematical expression *expr* from right to left. Notice that the command `\amrl` does not change the direction of text portions given in additional braces:

```
$ {\amrl{ abjd }}$
```
$ {\amrl{ abjd }}$

```
$ {\amrl{ {abjd} }}$
```
$ {\amrl{ {abjd} }}$

```
$ {\amrl{ a{bj}d }}$
```
$ {\amrl{ a{bj}d }}$

If the command `\amrl` is used in the argument of `\amrl`, both it and its argument must be enclosed in braces:

```
$ {\amrl{ a\amrl{bj}d }}$
```
$ {\amrl{ a\amrl{bj}d }}$

```
$ {\amrl{ a{\amrl{bj}}d }}$
```
$ {\amrl{ a{\amrl{bj}}d }}$

```
$ {\amrl{ \amrl{abjd} }}$
```
$ {\amrl{ \amrl{abjd} }}$

```
$ {\amrl{ {\amrl{abjd}} }}$
```
$ {\amrl{ {\amrl{abjd}} }}$

In general, all commands with arguments that are used in an `\amrl` command, and their arguments, should be written in braces unless the argument consists of a single character. In the latest versions of RyDArab, the use of the `\amrl` command in an Arabic mathematical environment is transparent for the user. It is added automatically to the expression in an Arabic environment.

```
\arabmath
${\hat b} , b{\sp{{17}}} ,
 {\sqrt{s+3}}$
```
$\hat{ب} , ب^{17} , \sqrt{ص + 3}$

4.2 Alphabetic symbols

The RyDArab system provides various Arabic characters without dots or vowels or diacritics, including three shapes of Arabic characters (initial, isolated and with a tail) in bold or contour forms.

Arabic literal symbols are given via the transliteration TransTec [1] (see Table 1), which is an adaptation of ArabTEX's [7] font `xnsh`. This coding differs slightly from the basic one used in ArabTEX to generate text in Arabic. This transliteration can be used either in text or mathematical expressions. In order to use this transliteration in text also, the user should load the transtec[1] package and add the command `\setcode{transtec}` in the preamble. The transliteration in use is not optimal. Since the package is intended for an Arab user, it would be better to get a direct coding scheme from the keyboard.

Some Arabic literal symbols can be obtained from the font NasX [8] also. In order to use this font, the user should load the NasX[2] package unless

Letter	Text	Mathematical expression					
ا	a	ا	a				
ب	b	ب	b	ر	B	ـبـ	\amb
ت	t						
ث	F						
ج	j	ح	j	ـحـ	J	ـجـ	\amj
ح	H						
خ	X						
د	d	د	d				
ذ	Z						
ر	r						
ز	z	ر	z				
س	s	س	c	ـسـ	C	ـسـ	\amc
ش	C						
ص	S	ص	s	ـصـ	S	ـصـ	\ams
ض	D						
ط	T	ط	T			ـطـ	\amt
ظ	V						
ع	E	ع	e	ـعـ	E	ـعـ	\ame
غ	G						
ف	f	ف	f	ـفـ	F	ـفـ	\amf
ق	q	ق	q				
ك	k	ك	k			ـكـ	\amk
ل	l	ل	l	ـلـ	L	ـلـ	\aml
م	m	م	m	ـمـ	M	ـمـ	\amm
ن	n	ن	n	ـنـ	N		
ه	h			ـهـ	H	ـهـ	\amh
ط	Q						
و	w	و	w				
ي	y	ى	y	ـيـ	Y	ـيـ	\amy
ى	Y						
ـَ	e						
ـُ	u						
ـِ	i						
ـٕ	eN *or* ee						
ـٗ	uN *or* uu						
ـٖ	iN *or* ii						
ـّ	W						
ـْ	o						
ء	A-	ء	A				
أ	Aa						
إ	AY						
ؤ	Aw						
ئ	Ay						
آ	Ma						
آ	La						
ـ	K *or* -						

Table 1: TransTec transliteration

[1] The transtec package was developed by K. Lagally following our propositions. Our thanks to K. Lagally.

[2] The NasX package is the core of an Arabic mathematical font. It was developed in order to have multiple glyphs of Arabic literal symbols directly in METAFONT.

the CurExt[3] [9] package is already in use. Literal symbols are obtained via commands and not directly (see Table 2).

The Latin literal symbols are also offered:

```
${\latinletter A} ,
 {\latinletter B} ,
 {\latinletter C}$
```

$C ، B ، A$

or

```
{\latinletter $A , B , C$}
```

4.3 Accents

Accents, in various shapes, can be combined with one or several symbols. Accents can be placed beside the symbol on its left.

The prime accent can be oriented to the left:

```
$a{'} , b{'} , j{'}$
```

or

```
$a{}^{\prime}, b{}^{\prime},
 j{}^{\prime}$
```

ا′ ، ب′ ، ح′

and the prime accent can be oriented to the right:

```
$a{}^{\amprime} ,
 b{}^{\amprime} ,
 j{}^{\amprime}$
```

ا‵ ، ب‵ ، ح‵

We can also have multiple prime accents oriented to the left:

```
$b{'} , b{'}{'} ,
 b{'}{'}{'}$
```

or

```
$b{}^{\prime} ,
 b{}^{\prime\prime} ,
 b{}^{\prime\prime\prime}$
```

ب′ ، ب″ ، ب‴

and multiple prime accents oriented to the right:

```
$b{}^{\amprime} ,
 b{}^{\amprime\amprime} ,
 b{}^{\amprime\amprime\amprime}$
```

ب‵ ، ب‶ ، ب‷

Several accents are offered:

```
${\hat b} , {\check b} , {\tilde b} ,
 {\acute b} , {\grave b} , {\dot b} ,
 {\ddot b} , {\breve b} , {\bar b} ,
 {\vec b} , {\overleftarrow{b}} ,
 {\overrightarrow{b}}$
```

$\overrightarrow{b} ، \overleftarrow{b} ، \vec{b} ، \bar{b} ، \breve{b} ، \ddot{b} ، \dot{b} ، \grave{b} ، \acute{b} ، \tilde{b} ، \check{b} ، \hat{b}$

[3] CurExt is an application for composing variable-sized curvilinear symbols. It allows looking after the typography of symbols such as brackets or kashida of Arabic letters in the composition of Arabic mathematical symbols or the justification of Arabic cursive texts according to strict rules of Arabic calligraphy.

Command	Glyph	Command	Glyph	Command	Glyph
\alef	ا	\aalef	ا		
\beh	ب	\bbeh	ب	\BEH	ب
\jeem	ج	\jjeem	ج	\JEEM	ج
\dal	د	\ddal	د		
\waw	و	\wwaw	و		
\zain	ر	\zzain	ر		
\tah	ط	\ttah	ط	\TAH	ط
\yeh	ى	\yyeh	ى		
\lam	ل	\llam	ل	\LAM	ل
\meem	م	\mmeem	م	\MEEM	م
\noon	ن	\nnoon	ن		
\seen	س	\sseen	س	\SEEN	س
\ain	ع	\aain	ع	\AIN	ع
\feh	ف	\ffeh	ف	\FEH	ف
\sad	ص	\ssad	ص	\SAD	ص
\qaf	ق	\qqaf	ق		
\hamza	ء	\hhamza	ء	\HEH	ه
\lamalef	لا	\llamalef	لا	\KAF	ك
\Yeh	ے	\YYeh	ے	\TTAH	ظ
\Noon	ں	\NNoon	ں		
\Zain	ڑ	\ZZain	ڑ		
\MEem	۾	\MMEem	۾		
\Beh	ڀ	\BBeh	ڀ	\BBEH	ڀ
\Jeem	ڃ	\JJeem	ڃ	\JJEEM	ڃ
\Heh	ھ	\HHeh	ھ	\HHEH	ھ
\Kaf	ڪ	\KKaf	ڪ	\KKAF	ڪ
\Lam	ڵ	\LLam	ڵ	\LLAM	ڵ
\Meem	ھ	\MMeem	ھ	\MMEEM	ھ
\Seen	۽	\SSeen	۽	\SSEEN	۽
\Ain	۽	\AAin	۽	\AAIN	۽
\Feh	ڦ	\FFeh	ڦ	\FFEH	ڦ
\Sad	ص	\SSad	ص	\SSAD	ص

Table 2: Arabic literal symbol font NasX

Some accents are variable-sized:

```
${\underline{abj}} , {\overline{abj}} ,
 {\widehat{abj}} , {\widetilde{abj}} ,
 {\overleftarrow{abj}} ,
 {\overrightarrow{abj}} ,
 {\overbrace{abj}} , {\underbrace{abj}}$
```

ابج ، ابج ، ابج ، ابج ، ابج ، ابج ، ابج

4.4 Digits

Eastern or western Arabic digits can be chosen according to the author's specifications, as follows.

For Western Arabic digits:

```
\warabnum
$0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9$
```

9 ، 8 ، 7 ، 6 ، 5 ، 4 ، 3 ، 2 ، 1 ، 0

Eastern Arabic digits:

```
\earabnum
$0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9$
```

٩ ، ٨ ، ٧ ، ٦ ، ٥ ، ٤ ، ٣ ، ٢ ، ١ ، ٠

Western Arabic digits in old style:

```
$ {\oldstylenum{\amrl{
 0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9}}}$
```

or

```
${\oldstylenum{0}} , {\oldstylenum{1}} ,
 {\oldstylenum{2}} , {\oldstylenum{3}} ,
 {\oldstylenum{4}} , {\oldstylenum{5}} ,
 {\oldstylenum{6}} , {\oldstylenum{7}} ,
 {\oldstylenum{8}} , {\oldstylenum{9}}$
```

9 ، 8 ، 7 ، 6 ، 5 ، 4 ، 3 ، 2 ، 1 ، 0

4.5 Numbers

Numbers have to be enclosed within braces. The braces can be omitted only if a number consists of a single digit. The sign of the number should be put outside the braces.

Numbers can be given with or without a fractional part, separated either by a decimal comma or a decimal dot. The numbers can be prefixed by an optional sign. The first example shows the formatting according to North African typographic conventions and the second according to Middle Eastern.

```
$7 , {5} , +{92} , -8 , {107} ,
 {12.345} , {3{\latinmath ,}14}$
```

3,14 ، 12345 ، 107 ، 8− ، 92+ ، 5 ، 7

```
\earabnum$7 , {5} , +{92} , -8 , {107} ,
 {12.345} , {3{\latinmath ,}14}$
```

٧ ، ٥ ، +٩٢ ، −٨ ، ١٠٧ ، ١٢٣٤٥ ، ٣٫١٤

4.6 Punctuation

The Arabic punctuation system is provided:

```
$, . ; : ! ? - \cdots$
```

، . ؛ : ! ؟ − ⋯

```
\latinpunct $,$ or \latinmath $,$
or $\lating$
```

،

Of course, Latin dotting is also available:

```
\latinpunct
$, . ; : ! ? - \cdots$
```

, . ; : ! ? − ⋯

4.7 Delimiters

Variable-sized delimiters with automatic adjusting are obtained as follows:

```
$( , | , [ , || , \{ ,
 \} , || , ] , | , )$
```

(، | ، [، || ، {،} ، || ،] ، | ،)

and

```
$\langle , \lbrace ,
 \rbrace , \rangle$
```

⟨، {،} ،⟩

4.8 Symbols

The basic symbols have been adapted as follows:

1. Addition, subtraction, multiplication, equality:

```
$+ , - , * , \times , =$
```

=، × ، ∗ ، − ، +

2. Division, western percentage and per mille and eastern percentage and per mille:

```
$/ , \% , \wpermille ,
 \epercent , \epermille$
```

/، ٪ ، ‰ ، ٪ ، /

3. Inferior, superior, membership and capacity:

```
$< , > , \in , \ni$
```

∈،∋،<،>

4. Assignment, equality and equivalence by definition:

```
$\seqm , \leqm , \leqv$
```

تعر تعر
≔، =، ⟺

5. Radical, angle, existential and universal quantifier:

```
$\surd , \angle ,
 \exists , \forall$
```

∀ ، ∃ ، ∠ ، √

6. Negation:

```
${\not=} , {\not<} , {\not\in}$
```

≠،≮،∉

4.9 Superscript and subscript

Superscripts and subscripts can be placed at the left of any symbol of the equation.

The command `\sp{`*expr*`}` or `^`*expr* produces *expr* as an exponent. The exponent *expr* should be given within braces unless it consists of a single token. The command `^` does not change the direction of *expr*. It can therefore be used only for a single token.

```
$b{{}\sp{{17}}}+5 ; b{{}\sp{2}}+5*s{{}^b}$
```

$ب^{17} + 5$ ؛ $ب^{2} + 5 * ص^{ب}$

The command \sb{*expr*} or _*expr* gives *expr* as an index. The index *expr* should be given within braces unless it consists of a single token. The command _ does not change the direction of *expr*. It can therefore be used only for a single token.

```
$b{{}\sb{{17}}}+5 ; b{{}\sb{2}}+5*s{{}_b}$
```

$ب_{17} + 5 \quad ؛ \quad ب_2 + 5 * ص_ب$

The empty braces {} are necessary to get the exponent or the index closer to the basic symbol.

4.10 Common functions

There are symbols for the usual abbreviations representing elementary functions in use in mathematics. Table 3 shows the predefined names assigned according to typographical conventions

Generally, the abbreviations representing elementary functions are used with dots. Sometimes, they are noted without dots.

```
\funwithdots
$\sin c + \tan s$
\funwithoutdots
$\sin c + \tan s$
```

جا س + ظا ص

حا س + طا ص

4.11 New function

The command \newfunc{*fname*} defines a function named *fname*.

```
${\newfunc{SGr}}(c) = \cos(c{{}\sp 2}) - 6$
```

صغر (س) = جتا (س^2) − 6

4.12 Function defined with cases

The command \cases{*array*} generates a function defined with different cases presented in *array*.

```
$d(c) = {\cases{
 {-4c} & {{\arhbox{ AYZa kan }} c<0} \cr
 { 4c} & {{\arhbox{ AYZa kan }} c>0} \cr
 {-2}  & {{\arhbox{ Gyr Zlk }}     } \cr
 }}$
```

$$د(س) = \begin{cases} -4س & \text{إذا كان } س > 0 \\ 4س & \text{إذا كان } س < 0 \\ -2 & \text{غير ذلك} \end{cases}$$

Mathematical Arabic symbols that stretch or shrink according to the context are provided by the system as well.

4.13 Root

The command \sqrt{*expr*} gives the square root of *expr*.

```
${\sqrt{{{ {b*9} \over {lc} }}}} -
 {\sqrt{c{{}\sp 2}}} + {\sqrt{5a}}$
```

$\sqrt{\frac{ب*9}{لا}} - \sqrt{س^2} + \sqrt{ا5}$

Name	Example	Result
Sine	$\sin c$	جا س
Cosine	$\cos c$	جتا س
Tangent	$\tan c$	ظا ص
Cotangent	$\cot c$	ظتا س
Secant	$\sec s$	قا س
Cosecant	$\csc c$	قتا س
Arc cine	$\arcsin c$	زجا س
Arc cosine	$\arccos c$	زجتا س
Arc tangent	$\arctan c$	زظا س
Arc cotangent	$\arccot c$	زظتا س
Arc secant	$\arcsec c$	زقا س
Arc cosecant	$\arccsc c$	زقتا س
Hyperbolic sine	$\sinh c$	جاز س
Hyperbolic cosine	$\cosh c$	جتاز س
Hyperbolic tangent	$\tanh c$	ظاز س
Hyperbolic cotangent	$\coth c$	ظتاز ص
Hyperbolic secant	$\sech s$	قاز س
Hyperbolic cosecant	$\csch c$	قتاز س
Arc hyperbolic sine	$\arcsinh s$	زجاز س
Arc hyperbolic cosine	$\arccosh c$	زجتاز س
Arc hyperbolic tangent	$\arctanh c$	زظاز س
Arc hyperbolic cotangent	$\arccoth c$	زظتاز س
Arc hyperbolic secant	$\arcsech c$	زقاز س
Arc hyperbolic cosecant	$\arccsch c$	زقتاز س
Logarithm	$\lg c$	لو س
Exponent	$\exp c$	قهس

Table 3: Usual functions

The command \root{*expr1*} \of {*expr2*} gives the *expr1* root of *expr2*.

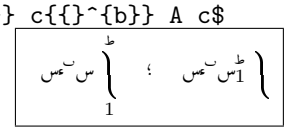
```
$ {\root{3b} \of
 {2+{\frac{b*9}{c}}}}$
```

$\sqrt[3]{2 + \frac{ب*9}{س}}$

4.14 Integral

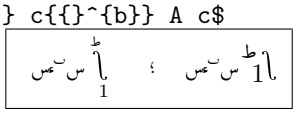
The command \lint_{*expr1*}^{*expr2*} gives the integral from *expr1* to *expr2* using the reversed symbol ⨕.

```
${\lint_{1}^{T}} c{{}^{b}} A c ;
 {\lint\limits_{1}^{T}} c{{}^{b}} A c$
```



The command `\sint_{`*expr1*`}^{`*expr2*`}` gives the integral from *expr1* to *expr2* using the reversed symbol ⨑.

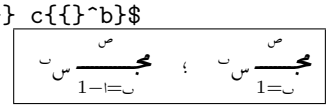
```
${\sint_{1}^{T}} c{{}^{b}} A c ;
 {\sint\limits_{1}^{T}} c{{}^{b}} A c$
```



4.15 Sum

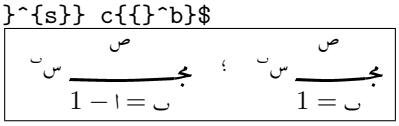
The command `\lsum_{`*expr1*`}^{`*expr2*`}` produces the sum from *expr1* to *expr2* using the conventional Arabic symbol ⨋.

```
${\lsum_{b=1}^{s}} c{{}^b} ;
 {\lsum_{b=a-1}^{s}} c{{}^b}$
```



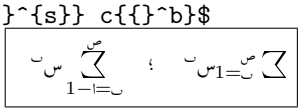
The command `\csum_{`*expr1*`}^{`*expr2*`}` produces the sum from *expr1* to *expr2* using the conventional Arabic curved symbol ⨌. This command needs the CurExt application.

```
${\csum_{b=1}^{s}} c{{}^b} ;
 {\csum_{b=a-1}^{s}} c{{}^b}$
```



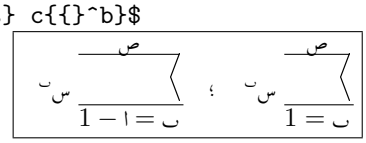
The command `\ssum_{`*expr1*`}^{`*expr2*`}` produces the sum from *expr1* to *expr2* with the inverted symbol ⅀.

```
${\ssum_{b=1}^{s}} c{{}^b} ;
 {\ssum\limits_{b=a-1}^{s}} c{{}^b}$
```



The command `\ssum_{`*expr1*`}^{`*expr2*`}` produces the sum from *expr1* to *expr2* using the inverted big symbol as shown (the broken corner in this symbol is a known flaw to be repaired in a future version).

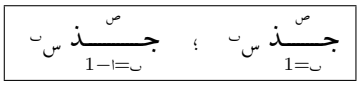
```
${\gsum_{b=1}^{s}} c{{}^b} ;
 {\gsum_{b=a-1}^s} c{{}^b}$
```



4.16 Product

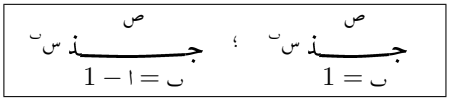
The same can be done with the product symbol of product. The command `\lprod_{`*expr1*`}^{`*expr2*`}` gives the product from *expr1* to *expr2* using the conventional Arabic symbol جـــذ.

```
${\lprod_{b=1}^{s}} c{{}^b} ;
 {\lprod_{b=a-1}^{s}} c{{}^b}$
```



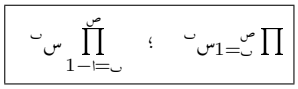
The command `\cprod_{`*expr1*`}^{`*expr2*`}` gives the product from *expr1* to *expr2* using the conventional Arabic curved symbol جـــذ. This command needs the CurExt application.

```
${\cprod_{b=1}^{s}} c{{}^b} ;
 {\cprod_{b=a-1}^{s}} c{{}^b}$
```



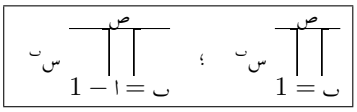
The command `\sprod_{`*expr1*`}^{`*expr2*`}` gives the product from *expr1* to *expr2* using the symbol ∏.

```
${\sprod_{b=1}^{s}} c{{}^b} ;
 {\sprod\limits_{b=a-1}^{s}} c{{}^b}$
```



The command `\gprod_{`*expr1*`}^{`*expr2*`}` gives the product from *expr1* to *expr2* using the big symbol ⨉ (see remarks about the big summation symbol).

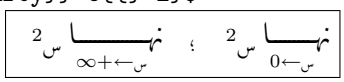
```
${\gprod_{b=1}^{s}} c{{}^b} ;
 {\gprod_{b=a-1}^s} c{{}^b}$
```



4.17 Limit

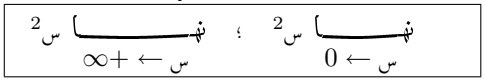
The command `\lim_{`*expr1*`} \to {`*expr2*`}` gives the limit when *expr1* tends to *expr2* using the conventional Arabic symbol نهـــا.

```
${\lim_{c \to 0}} c{{}^2} ;
 {\lim_{c \to +\infty}} c{{}^2}$
```



The command `\clim_{`*expr1*`} \to {`*expr2*`}` gives the limit when *expr1* tends to *expr2* using the conventional Arabic curved symbol نهـــا. This command needs the CurExt application.

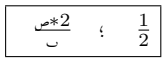
```
${\clim_{c \to 0}} c{{}^2} ;
 {\clim_{c \to +\infty}} c{{}^2}$
```



4.18 Fraction

The command `\frac{`*expr1*`}{`*expr2*`}` gives a fraction with *expr1* as numerator and *expr2* as denominator.

```
${\frac{1}{2}} ;
 {\frac{2*s}{b}}$
```



4.19 Equation numbering

Equations can be numbered or labeled at the right or the left in display math mode.

```
$$2c{{}\sp 2} - 3c + 5 = 0 \leqno (3a)$$
```

(٣) $$0 = 5 + 3س - {}^{2}س2$$

```
$$2mc{{}^2} - 3c + 5 = 0 \eqno (3b)$$
```

$$0 = 5 + 3س - {}^{2}م2 \qquad (ب3)$$

4.20 Matrix

The command `\matrix{`*array*`}` generates a matrix. The element's *array* flows from right to left.

The brackets are rendered linear by `\left(` and/or `\right)` or curvilinear by `\parentheses` as well as `\openparentheses`/`\closeparentheses`. In the last case, these commands need the CurExt application.

```
$m = {\left(}
 {\matrix{
 {1}&{2s}&{3}&{+5}\cr
 {1}&{2s}&{3}&{+5}\cr
 {4a}&{5}&{6}&{-2}\cr
 {4a}&{5}&{6}&{-2}\cr
 }} {\right)}$
```

$$\begin{pmatrix} 5+ & 3 & 2ص & 1 \\ 5+ & 3 & 2ص & 1 \\ 2- & 6 & 5 & ٤ \\ 2- & 6 & 5 & ٤ \end{pmatrix} = م$$

```
$m = {\parentheses{
 {\matrix{
 {1}&{2s}&{3}&{+5}\cr
 {1}&{2s}&{3}&{+5}\cr
 {4a}&{5}&{6}&{-2}\cr
 {4a}&{5}&{6}&{-2}\cr}}}$
```

$$\left(\begin{matrix} 5+ & 3 & 2ص & 1 \\ 5+ & 3 & 2ص & 1 \\ 2- & 6 & 5 & ٤ \\ 2- & 6 & 5 & ٤ \end{matrix}\right) = م$$

4.21 System of equations

The command `\system{`*system_of_equations*`}` generates a system of equations with a big closing brace on the right, as in Latin notation — but that is an opening brace in Arabic notation.

```
$$ {\system{
 {5c - 4s + 6} & { = 8s} \cr
 {5c - 7s + 9} & { = 3} \cr
 {5c - 7s + 9T} & { = {12}c} \cr }}$$
```

$$\left.\begin{matrix} 5س - 4ص + 6 = & 8ص \\ 5س - 7ص + 9 = & 3 \\ 5س - 7ص + 9ط = & 12س \end{matrix}\right\}$$

and there is also:

```
$${\left\{}
{\offinterlineskip \openup1\jot
 \tabskip=0pt plus1fil
 \vcenter{\halign{\tabskip=0pt
 & \hfil$#${\ }& \hfil$#${\ }\cr
 \amrl{&{3c}&+&{5s}&&& =&8}\cr
 \amrl{&&-&{6s}&-&{4m}& =&9}\cr
 \amrl{-&{5c}&&&+&{6m}& =&2}\cr}}}
 {\right.}\leqno (1b)$$
```

(ب1) $$\left.\begin{matrix} 8 = & 5ص + 3س \\ 9 = 4م - & 6ص \\ 2 = 6م + & 5س - \end{matrix}\right\}$$

4.22 Array

The system can also compose tables, including the possibility of combining lines and columns.

```
$${{{{\vbox{\tabskip=0pt
\offinterlineskip\def\tablerule{\noalign
{\hrule}}\halign{\strut#&\vrule#\tabskip
=1.5em plus 2.5em&\hfil#&\vrule#&\hfil#
\hfil&\vrule#&\hfil#&\vrule#\tabskip=0pt
\cr\tablerule\amrl{&{\multispan5\hfil
{\arhbox{almbyan}}\hfil}&&}\cr\tablerule
\amrl{&&&&{\omit\hidewidth{\arhbox
{alAYHdaFyat}}\hidewidth}&&&}\cr\amrl{&
{\raisebox{2ex}[0cm]{\arhbox{alnqT}}}&&
{\omit\hidewidth{\arhbox{alsynyQ}}
\hidewidth}&&{\omit\hidewidth{\arhbox
{alSadyQ}}\hidewidth}&&}\cr\tablerule
\amrl{&$a$&&&$5$&&&$7$&&&}\cr\tablerule
\amrl{&$b$&&&$8$&&&$9$&&&}\cr\tablerule
\noalign{\smallskip}}}}}}}$$
```

المبيان		
النقط	الإحداثيات السينية	الإحداثيات الصادية
ا	5	7
ب	8	9

4.23 Size variation

Some symbols, as well as superscripts, subscripts and delimiters, vary their size according to nesting.

```
$\sint , \ssum , \sprod$
```

$\sint , \ssum , \sprod$

```
$$\sint , \ssum , \sprod$$
```

$$\sint , \ssum , \sprod$$

```
$b{\sp{b{\sp{b{\sp{b}}}}}}$
$b{\sb{b{\sb{b{\sb{b}}}}}}$
```

$b{\sp{b{\sp{b{\sp{b}}}}}}$
$b{\sb{b{\sb{b{\sb{b}}}}}}$

```
${\Bigg(}{\bigg(}{\Big(}
 {\big(}( ){\big)}
 {\Big)}{\bigg)}{\Bigg)}$
```

$${\Bigg(}{\bigg(}{\Big(}{\big(}()){\big)}{\Big)}{\bigg)}{\Bigg)}$$

```
${\left(}{\left[}{\Bigg(}{\left[}{\bigg(}
 {\left[}{\Big(}{\left[}{\big(}{\left[}()
 {\right]}{\big)}{\right]}{\Big)}{\right]}
 {\bigg)}{\right]}{\Bigg)}{\right]}{\right)
 }$
```

$$\left(\left[\left(\left[\left(\left[\left(\left[\left(\left[()\right]\right)\right]\right)\right]\right)\right]\right)\right)$$

4.24 Translation

The system can translate mathematical expressions and link-words from Arabic to French or English and vice versa. The user can get different results from the following mathematical expression `\expression` depending on the environment, which is easily specified:

```
$$d(c) = {\cases{
 {{\ssum\limits_{b=1}^{s}} c{{}^b}} &
  { {\amhbox{AYZa kan} } c < 0} \cr
 {{\sint\limits_{1}^{s}} c{{}^b}}   &
  { {\amhbox{AYZa kan} } c > 0} \cr
 {{\sin} \pi} & {{\amhbox{Gyr Zlk}}} \cr
 }}$$
```

`\arabmath \artoar \expression`

$$د(س) = \begin{cases} \sum_{ب=1}^{ص} س_{ب} & \text{إذا كان } س > 0 \\ \int_{1}^{ص} س_{ب} & \text{إذا كان } س < 0 \\ \pi \text{ جا} & \text{غير ذلك} \end{cases}$$

Figure 1: Egyptian Arabic

`\latinmath \artoar \expression`

$$d(c) = \begin{cases} \sum_{b=1}^{s} c^b & \text{إذا كان } c < 0 \\ \int_{1}^{s} c^b & \text{إذا كان } c > 0 \\ \sin\pi & \text{غير ذلك} \end{cases}$$

Figure 2: Moroccan Arabic

`\latinmath \artofr \expression`

$$d(c) = \begin{cases} \sum_{b=1}^{s} c^b & \text{si } c < 0 \\ \int_{1}^{s} c^b & \text{si } c > 0 \\ \sin\pi & \text{sinon} \end{cases}$$

Figure 3: French

`\latinmath \artoen \expression`

$$d(c) = \begin{cases} \sum_{b=1}^{s} c^b & \text{if } c < 0 \\ \int_{1}^{s} c^b & \text{if } c > 0 \\ \sin\pi & \text{otherwise} \end{cases}$$

Figure 4: English

4.25 Miscellaneous

The command `\arhbox{`*string*`}` introduces the Arabic string *string* in an expression, `\time` the current time, `\day` the current day, `\month` the current month, `\year` the current year.

`$ {\arhbox{mFal}}$` → مثال

`$\time , \day , \month , \year$` → 2005 ، 6 ، 6 ، 872

The command `\amlwm` lists western Arabic month names.

`$\amlwm$` →

يناير، فبراير، مارس، أبريل، ماي، يونيه،
يوليوز، غشت، شتنبر، أكتوبر، نونبر، دجنبر

The command `\amlem` lists eastern Arabic month names.

`$\amlem$`

كانون الثاني، شباط، آذار، نيسان، أيار، حزيران، تموز،
آب، أيلول، تشرين الأول، تشرين الثاني، كانون الأول

The command `\wtoday` gives the current date consisting of the day, the western Arabic month name and the year. To give the eastern Arabic month names, `\etoday` can be used; `\numtoday` shows a numerical month.

`\warabnum$\wtoday$` → 6 يونيه 2005

`\earabnum$\etoday$` → ٦ حزيران ٢٠٠٥

`\warabnum$\numtoday$` → 2005/6/6

`\earabnum$\numtoday$` → ٢٠٠٥/٦/٦

Notes:

- One should not put mathematical expressions in display mode in the interior of an Arabic environment.
- Any command that requires arguments must be between braces (for example, `{\command{arg_1} ... {arg_n}}`).
- If an expression is made up of only one element with at least one argument, it should be preceded by a space (e.g. `$ {\command{arg}}$`).

5 Conclusion

The RyDArab package can be adapted to different needs and situations. Styles can be designed according to the typographic context. The present transliteration is still hard to use, and many open questions remain about this issue. The user should be aware of the use of braces but should not need to know all the details of the `\amrl` command. In recent

versions, automatic management of spaces among terms works well.

All commands can be renamed. For instance, the command `\amsqrt` can be changed into `\jdr` in order to get a name closer to the Arabic pronunciation of the symbol.

RyDArab has been improved, mainly with respect to the transparency of the command `\amrl` for inversion of writing expressions, and the generalization of the possibility to use the same names of commands as for Latin mathematics. Therefore, the system can provide an automatic translation of mathematical expressions from Arabic to Latin and vice versa.

The system can be used with Ω as well as with ArabTEX. In the near future, the system will be able to provide a multilingual scientific e-document by encoding with Unicode, structuring in XML and MathML and the use of a new Arabic mathematical font.

Acknowledgements: Thanks to all the people who have helped to improve the earlier versions. A special thanks to Barbara Beeton, Karl Berry and Steve Peter for proposing this work for reprinting, and their editorial corrections and contributions.

References

[1] Mostafa Banouni, Azzeddine Lazrek et Khalid Sami. Une translittération arabe/roman pour un e-document. In *5e Colloque International sur le Document Électronique*, pages 123–137, Conférence Fédérative sur le Document, Hammamet, Tunisi, 2002. `http://www.ucam.ac.ma/fssm/rydarab/doc/communic/cide5.pdf`.

[2] Yannis Haralambous and John Plaice. Multilingual typesetting with Ω, a case study: Arabic. In *Proceedings of the International Symposium on Multilingual Information Processing*, pages 137–154, Tsukuba, 1997.

[3] Donald Ervin Knuth. *The TEXbook*. Addison-Wesley, 1984.

[4] Donald Ervin Knuth and Pierre MacKay. Mixing right-to-left texts with left-to-right texts. *TUGboat* 8(1):14–25, 1987.

[5] Klaus Lagally. ArabTEX — Typesetting Arabic with vowels and ligatures. In *EuroTEX'92*, pages 153–172, Prague, 1992.

[6] Leslie Lamport. *LATEX: A Document Preparation System*. Addison-Wesley, 1986.

[7] Azzeddine Lazrek. Aspects de la problématique de la confection d'une fonte pour les mathématiques arabes. *Cahiers GUTenberg* 39–40, Le document au XXIe siècle: 51–62, 2001. `http://wwww.ucam.ac.ma/fssm/rydarab/doc/communic/gut39_40.pdf`.

[8] Azzeddine Lazrek. *Vers un système de traitement du document scientifique arabe*. Thèse d'État ès-Science, Université Cadi Ayyad, Marrakech, 2002. `http://www.ucam.ac.ma/fssm/rydarab/doc/communic/these.pdf`.

[9] Azzeddine Lazrek. CurExt, Typesetting variable-sized curved symbols. *TUGboat* 24(3):323–327, 2003. Proceedings of EuroTEX 2003: 14th European TEX Conference, Brest, France. `http://www.ucam.ac.ma/fssm/rydarab/doc/communic/curext.pdf`.

[10] Azzeddine Lazrek and Khalid Sami. Arabic mathematical document processing perspectives. In *Symposium on computer science and software development and its future impact*. Federation of Arab scientific research councils and University AlHadbaA Faculty, Iraq, 2000.
عزالدين لزرق وخالد سامي، آفاق معالجة النصوص الرياضياتية باللغة العربية، مؤتمر المعلوماتية والصناعة البرمجية ودورها المستقبلي، اتحاد مجالس البحث العلمي العربية وكلية الحدباء الجامعة، الموصل، العراق، 2000

[11] Azzeddine Lazrek and Khalid Sami. Towards a system for Arabic mathematical document processing. In *7th symposium of Sciences Arabization, Sciences Arabization in development system*. Egyptian Society for Arabizing Science and Ayn shams University, Egypt, 2000.
عزالدين لزرق وخالد سامي، نظام مبتكر لمعالجة النصوص الرياضياتية العربية، المؤتمر السابع لتعريب العلوم، تعريب العلوم في منظومة التنمية القومية، الجمعية المصرية لتعريب العلوم وجامعة عين شمس، القاهرة، مصر، 2001

◇ Azzeddine Lazrek
Department of Computer Science,
Faculty of Science,
University Cadi Ayyad, P. O. Box 2390,
Marrakech, Morocco
Phone: +212 44 43 46 49
Fax: +212 44 43 74 09
`lazrek@ucam.ac.ma`
`http://www.ucam.ac.ma/fssm/rydarab`

Software & Tools

PerlTEX: Defining LATEX macros using Perl

Scott Pakin

Abstract

Although writing documents with LATEX is straightforward, *programming* LATEX to automate repetitive tasks — especially those involving complex string manipulation — can be quite challenging. Many operations that a novice programmer can express easily in a general-purpose programming language cannot be expressed in LATEX by any but the most experienced LATEX users. PerlTEX attempts to bridge the worlds of document preparation (LATEX) and general-purpose programming (Perl) by enabling an author to define LATEX macros in terms of ordinary Perl code.

1 Introduction

Although TEX is a Turing machine and can therefore express arbitrary computation, the language is not conducive to programming anything sophisticated. As in an assembly language, arithmetic expressions are written in terms of register modifications (e.g., "`\advance\myvar by 3`") and relational expressions involving conjunction and disjunction are constructed from nested comparison operations (e.g., "`\ifnum\myvar>10 \ifnum\myvar<15`"). Loops are expressed in terms of tail-recursive macro evaluation. The only forms of string manipulation are single-token lookahead (`\futurelet`) and macro argument templates that either match a pattern or abort TEX. Finally, there are scalars but no aggregate data types (although these can sometimes be faked with clever use of macro expansion). While the LATEX kernel and various packages slightly raise the level of programming abstraction, the typical programmer is rapidly frustrated when attempting to code anything nontrivial.

Perl, in contrast, offers a rich programming environment with most of the features one expects from a modern high-level language. However, Perl has no inherent support for document typesetting. For short or highly repetitive documents, it is reasonable to write a Perl script that outputs a `.tex` file and runs it through latex. However, it is generally inconvenient to include a full-length article in its entirety within a Perl script just so it can invoke some simple function which is easier to express in Perl than in LATEX. Furthermore, a LATEX-generating Perl script supports only one-way communication: Perl can pass information to LATEX but not the other way around.

```
\newcount\n
\newcommand{\astsslow}[1]{%
  \n=#1
  \xdef\asts{}%
  \loop\ifnum\n>0 \xdef\asts{\asts*}\advance\n-1
  \repeat}
```

(a) Slow version from *The TEXbook*

```
\newcount\n
\newcommand{\astsfast}[1]{%
  \n=#1
  \begingroup
  \aftergroup\edef\aftergroup\asts\aftergroup{%
    \loop \ifnum\n>0 \aftergroup*\advance\n-1
    \repeat
  \aftergroup}\endgroup}
```

(b) Fast but non-scalable version from *The TEXbook*

```
\newcommand{\asts}{}
\perlnewcommand{\astsperl}[1]
  {'\renewcommand{\asts}{' . '*' x $_[0] . '}'}
```

(c) Fast PerlTEX version

Figure 1: Macro to define `\asts` as a sequence of N asterisks

In this article, we present PerlTEX, a package that consists of a Perl script (`perltex.pl`) and a LATEX 2_ε style file (`perltex.sty`). The user simply installs `perltex.pl` in an executable directory and `perltex.sty` in a LATEX 2_ε style-file directory, incorporates "`\usepackage{perltex}`" into any documents which need PerlTEX's features, and compiles such documents using `perltex.pl` instead of the ordinary latex command. Together, `perltex.pl` and `perltex.sty` give the user the ability to define LATEX macros in terms of Perl code. Once defined, a PerlTEX macro becomes indistinguishable from any other LATEX macro. PerlTEX thereby combines LATEX's typesetting power with Perl's programmability.

1.1 A simple example

A PerlTEX macro definition can be as simple as

```
\perlnewcommand{\hello}{"Hello, world!"}
```

which is essentially equivalent to:

```
\newcommand{\hello}{Hello, world!}
```

```
% Given a list of words, build up a \measurements macro as alternating
% words and word width in points, sorted by order of increasing width.
\perlnewcommand{\splitandmeasure}[1]{
    return
        "\\edef\\measurements{}%\n" .
        join ("",
              map "\\setbox0=\\hbox{$_}%\n" .
                  "\\edef\\measurements{\\measurements\\space $_ \\the\\wd0}%\n",
                  split " ", $_[0]) .
        "\\sortandtabularize{\\measurements}%\n";
}

% Given the \measurements macro produced by \splitandmeasure, output a
% two-column tabular showing each word and its width in points.
\perlnewcommand{\sortandtabularize}[1]{
    %word2width = split " ", $_[0];
    return
        "\\begin{tabular}{|l|r|} \\hline\n" .
        "  \\multicolumn{1}{|c|}{Word} &\n" .
        "  \\multicolumn{1}{c|}{Width} \\\\ \\hline\\hline\n" .
        join ("",
              map ("  $_ & $word2width{$_} \\\\ \\hline\n",
                   sort {$word2width{$a} <=> $word2width{$b}} keys %word2width)) .
        "\\end{tabular}\n";
}
```

Figure 2: A PerlTEX-defined LATEX macro that outputs a table of words sorted by typeset width

(The extra " characters delimit a string constant in Perl.)

To better motivate the use of PerlTEX, consider the first programming challenge in the "Dirty Tricks" appendix of *The TEXbook* [3]: construct a macro that accepts an integer N and defines another macro, `\asts`, to be a sequence of N asterisks. Figure 1(a) presents a LATEX wrapper, `\astsslow`, for the initial *TEXbook* solution. Besides relying on a set of TEX primitives which are unlikely to be familiar to a LATEX user, the code is slow; `\astsslow{10000}` takes over 6 seconds to run on the author's 2.8 GHz Xeon-based workstation.

Figure 1(b) presents a LATEX version of the "fast" solution from *The TEXbook*. `\astsfast` is highly unintuitive; it exploits artifacts of macro expansion and execution that occur when used in the context of the TEX `\aftergroup` primitive. Furthermore, it squanders space on TEX's input and save stacks, limiting the number of asterisks to fewer than 300 when run using the default latex program that ships with teTEX v1.02.

In contrast to *The TEXbook*'s solutions, the PerlTEX solution is fast, scalable, and should be comparatively easy to understand by anyone with basic Perl-programming and LATEX macro-writing skills. Figure 1(c) presents an `\astsperl` macro that takes an argument and returns a `\renewcommand` string which LATEX subsequently evaluates. `\astsperl{10000}` takes less than a second to run on the same 2.8 GHz Xeon system as did the previous macros and uses no TEX primitives, only ordinary LATEX and Perl commands.

1.2 A more complex example

One of PerlTEX's capabilities which is not available with a Perl script that outputs a `.tex` file is the ability to pass data bidirectionally between LATEX and Perl. Suppose, for example, that you wanted to write a macro that accepts a string of text, splits it into its constituent space-separated words, and outputs a table of those words sorted by their typeset width. Neither LATEX nor Perl can easily do this on its own. LATEX can measure word width but cannot easily split a string into words or sort a list; Perl cannot easily determine how wide a word will be when typeset but does have primitives for splitting and sorting strings.

A PerlTEX macro to do the job, named `\splitandmeasure`, is presented in Figure 2. It

```
\edef\measurements{}%
\setbox0=\hbox{How}%
\edef\measurements{\measurements\space How \the\wd0}%
\setbox0=\hbox{now}%
\edef\measurements{\measurements\space now \the\wd0}%
\setbox0=\hbox{brown}%
\edef\measurements{\measurements\space brown \the\wd0}%
\setbox0=\hbox{cow?}%
\edef\measurements{\measurements\space cow? \the\wd0}%
\sortandtabularize{\measurements}%
```

(a) Result of the call to `\splitandmeasure{How now brown cow?}`

```
How 19.44447pt now 17.50003pt brown 26.97227pt
cow? 21.11113pt
```

(b) Final contents of `\measurements` after evaluating the code in Figure 3(a)

```
\begin{tabular}{|l|r|} \hline
  \multicolumn{1}{|c|}{Word} &
  \multicolumn{1}{c|}{Width} \\ \hline\hline
  now & 17.50003pt \\ \hline
  How & 19.44447pt \\ \hline
  cow? & 21.11113pt \\ \hline
  brown & 26.97227pt \\ \hline
\end{tabular}
```

(c) Result of the call to `\sortandtabularize{\measurements}`

Word	Width
now	17.50003pt
How	19.44447pt
cow?	21.11113pt
brown	26.97227pt

(d) Final typeset table

Figure 3: Overall PerlTEX processing of `\splitandmeasure{How now brown cow?}`

accepts a string, splits it into words, and writes LATEX (or more accurately in this case, TEX) code which builds up a `\measurements` macro consisting of alternating words and word widths. This code is followed by a call to a second PerlTEX (helper) macro, `\sortandtabularize`, which accepts a list of alternating words and word widths (i.e., `\measurements`), sorts the list by word width, and outputs a `tabular` environment for LATEX to typeset.

Figure 3 illustrates the step-by-step operation of `\splitandmeasure`. Processing begins with LATEX invoking the `\splitandmeasure` macro, causing Perl to output LATEX code which measures each word (Figure 3(a)). LATEX then evaluates that code, producing the definition of `\measurements` shown in Figure 3(b) followed by an invocation of `\sortandtabularize`. Control once again passes to Perl, which sorts `\measurements` by word width and outputs a LATEX `tabular` environment (Figure 3(c)). LATEX then evaluates the `tabular`, producing the typeset output shown in Figure 3(d).

Macros such as `\splitandmeasure` which pass control from LATEX to Perl to LATEX to Perl and back to LATEX are comparatively easy to implement with PerlTEX — `\splitandmeasure` consists of a single Perl statement; its helper macro, `\sortandtabularize`, consists of only two Perl statements. However, it would be very difficult to implement comparable functionality without the help of PerlTEX.

The rest of this article proceeds as follows. Section 2 highlights some of the design decisions that went into PerlTEX's implementation. We contrast those design decisions to the ones made by similar projects in Section 3. Section 4 describes the mechanisms PerlTEX uses to transfer data betwen LATEX and Perl. Defining Perl macros in LATEX was the greatest challenge in implementing PerlTEX and required some fairly sophisticated LATEX trickery. The solutions that were developed are described in Section 5. By comparison, the Perl side of PerlTEX is comparatively straightforward and is described briefly in Section 6. Section 7 presents some avenues for future enhancements to PerlTEX. Finally, we draw some conclusions in Section 8.

2 Design decisions

There are multiple ways that PerlTEX could have been implemented. The following are the primary alternatives:

- Use the semi-standard "`\write18`" mechanism to invoke the perl executable.
- Patch the TEX executable to interface with the Perl interpreter.
- Implement a Perl interpreter in LATEX.
- Construct macros that enable LATEX to communicate with an external Perl interpreter.

The final option is the one that was deemed best for PerlTEX. The "`\write18`" approach is a security risk; enabling it (e.g., using the `-shell-escape` command-line option present in some TEX distributions) permits not only PerlTEX but any LATEX package to execute arbitrary programs on the user's system. Patching TEX is inconvenient for the user, who will need to recompile TEX (plus pdfTEX, ε-

TEX, pdf-ε-TEX, Ω, and any other TEX-based system for which the user wants to add Perl support) then re-dump the LATEX 2ε format file for each Perl-enhanced build of TEX. Implementing a Perl interpreter in LATEX has the advantage of not requiring a separate Perl installation. However, a LATEX-based Perl interpreter, besides being extremely difficult to implement, would necessarily support only a small subset of Perl, as much of the language cannot be expressed in terms of the mechanisms provided by TEX.

As this article will demonstrate, providing LATEX-level mechanisms to facilitate communication between LATEX and an external Perl interpreter enables safe execution of Perl code, ease of installation, compatibility with any underlying TEX implementation, and access to every feature of the Perl language.

3 Related work

PerlTEX is not the first system that attempts to augment LATEX macro programming with a general-purpose programming language. However, PerlTEX's approach, as outlined in the previous section, makes it unique relative to other, similar systems. Note that many of the following systems support not only LATEX but other formats as well (e.g., Plain TEX, ConTEXt, and Texinfo); for the purpose of exposition we limit our discussion to LATEX.

After releasing PerlTEX, the author discovered an existing program written by Alexander Shibakov also called PerlTEX [6]. Unlike the PerlTEX described in this paper, Shibakov's version is implemented as a patch to TEX. That is, the user must recompile TEX (and all its variants) with the PerlTEX patches and re-dump the desired formats. The result is that Perl is more integrated into TEX than is otherwise possible. All code between `\perl` and `\endperl` is executed by Perl. Furthermore, Shibakov's PerlTEX also supports two-way communication between TEX and Perl by enabling code within a `\perl`...`\endperl` block to insert characters and control sequences into the TEX input stream. While Shibakov's PerlTEX works with any TEX format — Plain TEX, LATEX, ConTEXt, Texinfo, etc. — the PerlTEX described in this paper works only with LATEX. However, this paper's PerlTEX has the important advantage of not requiring TEX recompilation, which is tedious and may not be possible when using a commercial TEX implementation.

Paraschenko takes a similar approach to Shibakov's with his sTEXme [4], which uses Scheme rather than Perl as the TEX extension language. sTEXme adds a single command to TEX: `\stexme`, which works like `\input` but accepts the name of a Scheme file rather than a TEX or LATEX file. When the Scheme interpreter evaluates the given file, output procedures such as `newline` and `display` write into the TEX input stream. Two new procedures, `pool-string` and `get-cmd`, provide access to TEX internal state. As with Shibakov's PerlTEX, sTEXme's tight integration with TEX comes at the cost of having to recompile TEX and re-dump all of the format files before the extension language can be used.

TEX2page [7] uses also uses Scheme as a TEX extension language. However, its design is closer to that of (this paper's) PerlTEX than to sTEXme's. TEX2page provides an `\eval` macro which brackets Scheme code. The document is first compiled using the ordinary latex executable. As part of that process, `\eval` simply writes its argument to a file. The user then runs tex2page, which invokes the Scheme interpreter on the extracted Scheme code and writes the resulting LATEX code to a file. Finally, the user re-runs latex and, on this pass, `\eval` loads the Scheme-produced LATEX code into the document, where it is typeset normally. Although TEX2page's multi-pass approach supports two-way communication betwen LATEX and Scheme, it does require an extra run of tex2page and an extra run of latex for each nesting level. For large documents or heavily nested `\eval` calls, this can be slow and tedious. PerlTEX, in contrast, requires no more latex runs than the document would otherwise require.

The idea behind PyTEX [1] is to use Python, not LATEX, as the document's top-level language. With PyTEX, the user's Python code passes strings to a TEX dæmon [2] to evaluate. PyTEX supports only one-way communication (i.e., Python to LATEX but not LATEX to Python). PerlTEX, in contrast, supports two-way communication, which is necessary when writing code in a general-purpose language that requires access to typesetting information such as string widths, page counts, or register contents.

$A^{m}{}^{r}_{i}{}^{t}a$ [5] presents an integration framework based on re-entrant *here* documents which supports communication among a variety of languages such as Perl, Python, LATEX, Ruby, and POV-Ray. Each language can generate code to be executed by any other language. The result of each execution (which itself may recursively generate code for additional languages) is code to be executed by the parent language. While $A^{m}{}^{r}_{i}{}^{t}a$ is a highly capable system, its power necessarily introduces an extra level of complexity to the user. Relative to the generality of $A^{m}{}^{r}_{i}{}^{t}a$, PerlTEX's niche is that it enables users to

Table 1: Files used for communication between Perl and LaTeX

Filename	Meaning	Purpose
`\jobname.topl`	"to" (Perl)	LaTeX to Perl communication *also*: signal Perl that `\jobname.frpl` has been read
`\jobname.frpl`	"from" (Perl)	Perl to LaTeX communication
`\jobname.tfpl`	"to flag"	signal Perl that `\jobname.topl` is ready to be read
`\jobname.ffpl`	"from flag"	signal LaTeX that `\jobname.frpl` is ready to be read
`\jobname.dfpl`	"done-with-from-flag flag"	signal LaTeX that Perl is ready for the next transaction

add a few Perl macros to an existing LaTeX document with minimal hassle and without having to buy into a more comprehensive software framework.

4 Communication between LaTeX and Perl

PerlTeX has two main components: a Perl script (`perltex.pl`) and a LaTeX 2_ε style file (`perltex.sty`). PerlTeX is invoked by running the command perltex.pl, just as one would run latex. `perltex.pl` itself is fairly simple; essentially, it installs a "server" which executes incoming Perl code and outputs the LaTeX result. More information is provided in Section 6.

`perltex.sty` provides the `\perlnewcommand`, `\perlrenewcommand`, `\perlnewenvironment`, and `\perlrenewenvironment` macros which are analogous to their non-`perl` namesakes but are defined with Perl code instead of LaTeX code in the macro body. When a PerlTeX macro is defined, `perltex.sty` instructs `perltex.pl` to define a corresponding Perl subroutine with the given body. Then, when the macro is invoked, `perltex.sty` instructs `perltex.pl` to execute the subroutine. A similar process is performed when defining PerlTeX environments but involving two behind-the-scenes macros, one for the "begin" code and one for the "end" code.

Almost by necessity, communication between LaTeX and Perl is implemented via the filesystem. TeX provides primitives for creating new files, opening existing files, reading and writing files, and closing files, but no other mechanisms that can be used to communicate with entities outside of TeX (excluding `\write18`, which has security implications, as mentioned in Section 2). TeX returns a failure code when trying to open a nonexistent file; this condition can safely be tested from within TeX.

The primary challenge in transferring data via the filesystem is detecting when a file is no longer being written to. This challenge needs to be addressed both on the Perl side of the transfer and on the LaTeX side. The solution that PerlTeX takes is to employ some auxiliary "flag" files that signal when an associated file is complete. Table 1 describes the complete set of files used for communication between Perl and LaTeX.

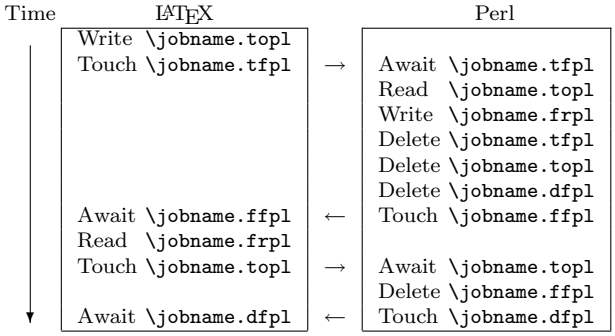


Figure 4: LaTeX/Perl communication protocol

The communication protocol proper, which is illustrated in Figure 4, is necessarily complex because it needs to work around two important limitations of the TeX system:

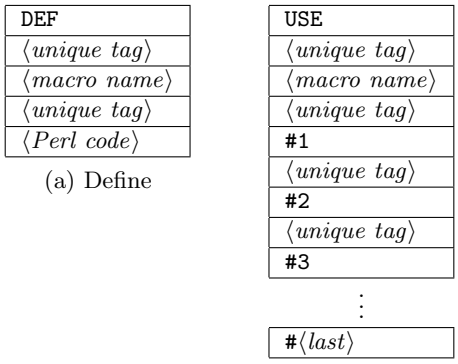
1. TeX lacks a mechanism for deleting files.
2. The latex executable — at least the version shipped with the teTeX TeX distribution — is prone to crash when opening a file for input while an external process is in the midst of deleting that file. (Recall that testing if a file exists means opening the file for input and checking for success.)

If it were not for those limitations, the protocol would require only one flag file and half as many steps.

The `\jobname.frpl` file contains ordinary LaTeX code that simply gets `\input` into the document. `\jobname.topl`, in contrast, contains not only Perl code but also some metadata that helps offload some string manipulation from LaTeX to Perl. Consider passing the LaTeX string

```
In C it's \texttt{printf("Hello!")}.
```

as an argument to a function declared with `\perlnewcommand`. Because the string contains both



(a) Define

(b) Invoke

Figure 5: Data written to `\jobname.topl` to define or invoke a Perl subroutine

single and double quote characters, every occurrence of at least one type of quote will need to be backslash-escaped for Perl. Rather than do this on the LaTeX side, `perltex.sty` sends the string as-is to `perltex.pl`, which automatically quotes the string while reading it from `\jobname.topl`. The implication is that `perltex.sty` cannot pass raw Perl code to `perltex.pl` to evaluate.

Hence, `\jobname.topl` needs contain some metadata telling `perltex.pl` what to do with the rest of `\jobname.topl`'s contents. This metadata is of one of two types. When `\perlnewcommand` or any of the other PerlTeX macros is invoked, `perltex.sty` sends `perltex.pl` the information shown in Figure 5(a). Then, when a macro defined by one of PerlTeX's `\perl`... macros is called, `perltex.sty` sends `perltex.pl` the information shown in Figure 5(b). In Figure 5, ⟨*unique tag*⟩ refers to a sequence of 20 letters that `perltex.pl` generates randomly at initialization time and passes to `perltex.sty` via the `latex` command line. The ⟨*unique tag*⟩ is used as a separator, so `perltex.pl` knows where one piece of information ends and the next one begins. ⟨*macro name*⟩ is the name of the macro to be defined or used. `perltex.pl` defines a Perl subroutine named ⟨*macro name*⟩ but with the leading backslash replaced with "`latex_`". The subroutine body contains ⟨*Perl code*⟩ verbatim. When a PerlTeX-defined macro is invoked, `perltex.sty` passes `perltex.pl` the name of the macro plus all of the arguments as expanded LaTeX code.

Figures 6 and 7 present a more concrete expression of a LaTeX/Perl file transfer. Figure 6(a) shows the contents of the `\jobname.topl` file that LaTeX writes as part of the `\perlnewcommand` invocation presented previously in Figure 1(c); Figure 6(b) shows the contents of the `\jobname.frpl` file that Perl writes in response. Figure 7(a) shows the contents of the `\jobname.topl` file that LaTeX writes while executing "`\astsperl{10}`" and Figure 7(b) shows the `\jobname.frpl` file that Perl writes in response to that.

```
DEF
TKOUVLRCDIVSVSIZVHFI
\astsperl
TKOUVLRCDIVSVSIZVHFI
'\renewcommand{\asts}{' . '*' x $_[0] . '}'
```

(a) Macro definition (`\jobname.topl`)

```
\endinput
```

(b) Result of macro definition (`\jobname.frpl`)

Figure 6: LaTeX/Perl communication associated with the code in Figure 1(c)

```
USE
TKOUVLRCDIVSVSIZVHFI
\astsperl
TKOUVLRCDIVSVSIZVHFI
10
```

(a) Macro invocation (`\jobname.topl`)

```
\renewcommand{\asts}{**********}\endinput
```

(b) Result of macro invocation (`\jobname.frpl`)

Figure 7: LaTeX/Perl communication associated with an invocation of "`\asts{10}`"

Expansion is a tricky issue in PerlTeX's design and, in fact, is handled differently in PerlTeX v1.1 than in earlier versions of PerlTeX. The challenge is that Perl cannot evaluate LaTeX code; it requires all subroutine parameters to be ASCII strings. Consider this invocation of some PerlTeX macro `\mymacro`:

```
\mymacro{Hello from Perl\noexpand\TeX!}
```

How should `\mymacro`'s argument be passed to Perl? (1) Unexpanded, as

```
Hello from Perl\noexpand\TeX!
```

or (2) partly expanded, as

```
Hello from Perl\TeX!
```

or (3) fully expanded, as

```
Hello from PerlT\kern -.1667em\lower .5ex
\hbox {E}\kern -.125emX\@!
```

?

The first alternative makes PerlTEX macros behave differently from LATEX macros, which generally execute their arguments. The other two alternatives lead to unexpected behavior in cases like `\mymacro{\def\foo{world}Hello, \foo!}`, which cause latex to abort with an `Undefined control sequence` error as it tries to expand the not-yet-defined `\foo` control word which immediately follows the non-expandable `\def` control word. Execution is not an option because an invocation like `\mymacro{\mbox{Oops}}` would need to pass a box to Perl, which cannot practically be done.

PerlTEX's approach (as of version 1.1) is to partially expand macro arguments but with `\protect` mapped to `\noexpand` and with `\begin` and `\end` marked as non-expandable. In this approach, robust macros (such as many of the ones provided by LATEX) are not expanded while fragile macros (such as many of the ones defined by a user) are expanded. For example, the following sequence will write "LATEX is nice" to the typeset output, which is a fairly intuitive result:

```
\newcommand{\adjective}{nice}
\perlnewcommand{\identity}[1]{$_[0]}
\identity{\LaTeX{} is \adjective.}
```

5 Defining Perl macros from LATEX

From a LATEX programming perspective, there are two primary challenges that need to be overcome in order to implement `\perlnewcommand`, `\perlrenewcommand`, `\perlnewenvironment`, and `\perlrenewenvironment`:

1. How can syntactically incorrect LATEX code be stored and manipulated?
2. How can a LATEX macro iterate over a variable number of macro arguments?

A solution to the former question is required because `\perlnewcommand`, etc. need to write Perl code to a file. Syntactically correct Perl code is unlikely also to be syntactically correct LATEX code. For example, Perl associative arrays are prefixed with the LATEX comment character, "`%`"; Perl scalars are prefixed with "`$`", which introduces math mode in LATEX; and Perl uses "`\`" to escape special characters in strings and create variable references while LATEX expects a valid control sequence to follow. The difficulty, therefore, is in enabling a LATEX macro to manipulate one of its arguments while neither expanding nor evaluating it.

A solution to the latter question, how to iterate over macro arguments, is required because each macro argument must be passed to Perl (via the `\jobname.topl` file). Just as with `\newcommand`, a macro defined by `\perlnewcommand` accepts a user-defined number of arguments (e.g., `\perlnewcommand{\mymac}[5]{...}`). However, TEX requires that macro arguments be referenced by a literal number (e.g., "`#3`"); variable argument numbers (e.g., "`#\argnum`") result in a TEX error. The challenge is to construct a loop that iterates over a variable number of arguments, writing each argument to a file, yet does not use a variable to reference any arguments.

5.1 Storing non-LATEX code

The final argument to `\perlnewcommand` is a block of Perl code which will almost certainly cause errors if evaluated by LATEX. Storing this Perl code in a macro is similar to outputting non-LATEX code using the `\verb` macro. The difference is that `\verb` does not need to store its argument.

The solution taken by `perltex.sty` works as follows. First, `\perlnewcommand` is defined to read one fewer argument than actually needed; the Perl code is considered the first piece of text *following* `\perlnewcommand`'s argument list. `\perlnewcommand`'s last action is to begin a new variable scope with `\begingroup` and, within that scope, set the TEX category codes for all characters to "other" (i.e., 12) to prevent "`%`", "`$`", "`\`", and so forth from being treated specially. The only exceptions are that "`{`" and "`}`" retain their original meanings so that TEX brace-counting will indicate when the Perl code has ended. Also, the end-of-line character is made significant because it has meaning within a Perl string.

The next task involves figuring out how to store the Perl code following `\perlnewcommand` and then reset all of the category codes back to their prior values. The trick that `perltex.sty` relies upon is the TEX `\afterassignment` primitive, which specifies a command to execute after the next assignment takes place. The following are the last two lines of `\perlnewcommand`'s implementation:

```
\afterassignment\plmac@havecode
\global\plmac@perlcode
```

In other words, the `\plmac@havecode` macro should be executed after the next assignment. Then, `\perlnewcommand` ends with an assignment to the global token register `\plmac@perlcode`. The right-hand side of the assignment is the block of Perl code, which is already within a pair of curly braces, as required by a token-register assignment. After the assignment takes place, control automatically transfers to the `\plmac@havecode` macro. Before

changing category codes, \perlnewcommand began a new scope with \begingroup; \plmac@havecode resets the category codes by executing the matching \endgroup. The result is that the Perl code is stored unevaluated in the \plmac@perlcode token register, as desired, and LaTeX can continue compiling the user's document.

```
\def\plmac@havecode{%
            ⋮
  \let\plmac@hash=\relax
  \plmac@argnum=\@ne
  \loop
    \ifnum\plmac@numargs<\plmac@argnum
    \else
      \edef\plmac@body{%
        \plmac@body
        \plmac@sep\plmac@tag\plmac@sep
        \plmac@hash\plmac@hash
          \number\plmac@argnum}%
      \advance\plmac@argnum by \@ne
  \repeat
  \let\plmac@hash=##%
            ⋮
}
```

Figure 8: perltex.sty code that iterates over macro arguments

5.2 Iterating over macro arguments

One limitation of TeX's macro-processing facility is that macro arguments must be referred to by a literal argument number. Hence, "#2" is acceptable but \newcommand*{\whicharg}{2} followed inside a macro definition by "#\whicharg" results in an "Illegal parameter number" error. Even worse, the error occurs at macro-definition time; even if a macro containing "#\whicharg" is never invoked it will still cause TeX to report an error and abort.

Fortunately, the aforementioned limitation is not insurmountable but it does require a bit of trickery. The solution is to replace "#" with a control sequence that is let-bound to \relax. TeX does not expand such control sequences. After the macro is defined, the control sequence can then be let-bound to #, making it work as desired.

There are two caveats to this approach. First, # can be used only within a macro definition; hence, the macro definition must itself be within a macro definition in order for the let-binding to succeed. Second, when the macro is executed, # must be followed by a literal argument number. The let-binding trickery merely delays the literal-number check from definition time to execution time — but this is sufficient for the purpose of accessing a variable-numbered macro argument. Careful use of \edef and \noexpand can then make it possible to iterate over macro arguments, as desired.

Figure 8 presents an excerpt of code from perltex.sty which constructs a \plmac@body macro that references in turn each argument from 1 up to \plmac@numargs. In this code, \plmac@hash is the placeholder for the # character and \plmac@argnum is the argument number, which varies from 1 to \plmac@numargs. In each iteration of the loop, \plmac@body is redefined as the concatenation of its old value, a carriage-return character (\plmac@sep), a unique tag as described in Section 4, another carriage-return character, and "##" (doubled because the \edef is nested within another macro) followed immediately by the argument number. Only at the end of the loop, after \plmac@body has its final contents, is \plmac@hash set to an actual # character (written as "##" because it occurs within the definition of \plmac@havecode).

6 Processing Perl code

While perltex.sty contains rather complex LaTeX code, perltex.pl contains fairly straightforward Perl code. perltex.pl's basic structure is as follows:

1. Parse the command line.
2. Create a secure sandbox in which to execute Perl code coming from the document.
3. Spawn a latex process, passing it a variety of macro definitions in addition to the name of the user's LaTeX source file.
4. Repeatedly poll for new Perl code to execute, execute that code in the secure sandbox, and return the (LaTeX) result.

perltex.pl uses the Safe and Opcode modules to create a secure sandbox in which to execute code. The idea behind a sandbox is that it limits the types of code that can be executed. Code deemed too dangerous to run (e.g., an attempt to delete a file or to kill a running process) produces a run-time error. Sandboxing the code passed from LaTeX to perltex.pl enables users to build a PerlTeX document created by a third party without having to worry about it containing malicious or otherwise destructive Perl code. The default set of sandbox permissions is Opcode's ":browse" permissions, which enable the core Perl language features such as arrays, loops, variable assignment, and function definitions, but forbid creating and opening files, spawning child processes, communicating

with other processes, and performing most other input/output functions. A command-line option selectively enables individual functions or groups of functions. (Another command-line option disables sandboxing altogether, although this is not generally recommended.)

After spawning latex (alternatively, pdflatex, elatex, vlatex, or any other LaTeX compiler), `perltex.pl` makes that the *foreground* process, leaving itself in the background. Doing so makes it possible for latex to run interactively (e.g., when encountering an error), which it could not do as easily as a background process.

Finally, `perltex.pl` enters a loop in which it polls the filesystem for incoming Perl code, executes the code, and returns the (LaTeX) result via the filesystem. The LaTeX/Perl communication protocol is as described in Section 4. The loop terminates when the latex process exits.

7 Future work

Although PerlTeX performs its tasks reliably, there are a variety of avenues for future expansion and enhancement, mostly suggested by PerlTeX users. First, while PerlTeX's `\perlnewcommand`, `\perlrenewcommand`, `\perlnewenvironment`, and `\perlrenewenvironment` macros provide a faithful Perl analogue to LaTeX's command- and environment-defining macros, a useful addition would be a way to execute Perl code directly. Such a feature would be useful when writing Perl code that is executed only once, such as program initialization or generation of a particularly unique list, table, or equation.

The performance of the PerlTeX implementation could be improved. Although filesystem-based communication between LaTeX and Perl is portable, file activity — especially over a remote filesystem — can be a performance bottleneck when compiling PerlTeX-intensive documents.

One alternative to using the filesystem is to communicate using standard input and standard output. There are two challenges in implementing this approach. First, TeX lacks a mechanism to explicitly flush standard output. Depending on how latex is implemented, a deadlock can result if LaTeX sends a command to Perl and blocks waiting for the result while Perl never sees the command because the standard-output buffers have not been flushed. Second, maintaining support for user interaction (e.g., to diagnose error conditions) may be complicated if PerlTeX needs to compete with the user for control over standard input and standard output.

A second alternative to filesystem-based communication is to use named pipes, an internal operating-system data structure for interprocess communication. A problem with named pipes is that they are not as portable as files; not every operating system supports named pipes or implements them in the file namespace (i.e., they might be accessed via a different interface, making them inaccessible to TeX). In addition, while Perl can create named pipes, TeX cannot. This restriction may limit their usefulness in the context of PerlTeX.

Finally, a meaningful follow-on to PerlTeX would be an ⟨*anything*⟩TeX system. Most of PerlTeX's magic is in the extension-language-independent `perltex.sty` file. The Perl-specific `perltex.pl` file performs only simple file and string manipulation and should easily be portable to any other programming language. Users could then write LaTeX macros in the language (or languages) with which they are most comfortable.

8 Conclusions

As this article has demonstrated, PerlTeX takes a practical, portable approach to augmenting TeX's typesetting finesse with Perl's power in string manipulation and general-purpose programming. The importance of PerlTeX's design — a Perl "server" that accepts Perl input and produces LaTeX output — is that it enables two-way communication between LaTeX and Perl. As Section 1.2 demonstrated, LaTeX can invoke a Perl subroutine which can produce LaTeX code that itself invokes a Perl subroutine which outputs some final LaTeX code. Support for this dynamic usage model is a clear advantage of PerlTeX over a custom Perl script which generates a static LaTeX document. By exploiting Perl's sandboxing features, users can compile PerlTeX documents written by others without fear of their system being harmed by malicious Perl code.

A key design decision in PerlTeX's implementation was to keep the perl and latex programs largely decoupled. The advantage of decoupling the two programs is that PerlTeX remains compatible with every underlying TeX variant — TeX, pdfTeX, ε-TeX, pdf-ε-TeX, Ω, etc. — and does not require the user to recompile the base TeX executable or re-dump a LaTeX 2ε format. The disadvantages are that Perl cannot directly access TeX's internals and that TeX can communicate with external applications only via the filesystem (not counting the security-risk-prone `\write18` mechanism or by revoking user control over standard input and standard output). This article has presented a filesystem-based communication protocol that en-

ables LaTeX and Perl to communicate even though the two systems are asymmetric in terms of the types of file operations each supports. Even though TeX cannot, for example, delete a file, the protocol ensures correct behavior, including in the presence of mutually recursive LaTeX and Perl routines such as those utilized in Section 1.2.

Finally, this paper presented solutions to two challenging LaTeX puzzles: how to store and manipulate syntactically incorrect LaTeX code; and, how to iterate over a variable number of macro arguments. The former problem is solved using a token-register assignment at the end of a macro call with `\afterassignment` used to transfer control to a continuation macro. The latter problem is solved using a control sequence bound to `\relax` while defining a macro but bound to `#` afterwards. Neither of those techniques is specific to PerlTeX; advanced LaTeX users can readily employ them in their own macros.

In summary, PerlTeX combines Perl's fortes of string manipulation, regular-expression processing, and general programmability with LaTeX's typesetting capabilities. A few lines of PerlTeX can easily replace their much longer, more complex equivalent coded in ordinary LaTeX. PerlTeX thereby makes sophisticated LaTeX macro programming more accessible to the novice and more convenient for the advanced user.

The PerlTeX distribution is available for download from CTAN at `http://www.ctan.org/tex-archive/macros/latex/contrib/perltex/`.

9 Acknowledgments

The author would like to thank all of the people who have provided feedback, suggestions, and bug reports for PerlTeX including Andrei Alexandrescu, José Pedro Oliveira, Fernando P. Schapachnik, Ivo Welch, James Quirk, Michele Dondi, Hans Fredrik Nordhaug, and everyone else who helped make PerlTeX a success. Also, thanks to James Quirk for critiquing the PerlTeX examples originally used in this paper's Introduction section.

References

[1] Jonathan Fine. PyTeX: Python plus TeX. `http://www.pytex.org/`.

[2] Jonathan Fine. TeX as a callable function. In *Proceedings of the 13th European and 10th Polish TeX Conference (EuroBachoTeX 2002)*, pages 26–35, Bachotek, Poland, April 29–May 3, 2002. Available from `http://www.pytex.org/doc/euro2002.pdf`.

[3] Donald E. Knuth. *The TeXbook*. Addison-Wesley, 1986. ISBN 0-201-13447-0.

[4] Oleg Paraschenko. sTeXme: TeX + Scheme. `http://stexme.sourceforge.net/`.

[5] James J. Quirk. Programming dynamic LaTeX documents. In *Proceedings of the 24th Annual Meeting and Conference of the TeX Users Group (TUG 2003)*, Waikoloa, Hawai'i, July 20–25, 2003. Slides available from `http://www.tug.org/tug2003/bulletin/highlights/slides/2_Monday/4_Quirk/4_quirk.pdf`.

[6] Alexander Shibakov. PerlTeX—a fusion of Perl and TeX via Web2C. `http://www.math.tntech.edu/alex/`.

[7] Dorai Sitaram. TeX2page. `http://www.ccs.neu.edu/home/dorai/tex2page/`.

◇ Scott Pakin
4975 S. Sol
Los Alamos, NM 87544-3794, USA
`scott+tb@pakin.org`
`http://www.pakin.org/~scott`

TeX and prepress

Siep Kroonenberg

Abstract

This article discusses preparing documents for professional printing with TeX and pdftex, including color printing and prepress standards.

1 History

Most of us aren't graphics professionals. Still, now and then we have things that need to be printed professionally at a conventional printshop.

A bit of historical perspective: originally, we dealt with this by supplying 'camera-ready' laser-printer output to the printshop, from which printing plates were created photographically. This method certainly prevented surprises, but was not the way to get quality output.

During the nineties, PostScript dumps became increasingly popular among TeX users as an alternative. Professional-quality output became a real possibility. But it might take some effort to find a printshop willing to process raw PostScript. The usual practice in the graphics industry was handing off application files. Of course, this had its drawbacks: it was easy to forget to include a font or a graphic file in the job, and the printshop from its

Editor's note: This article was first published in *MAPS* 30 (Spring 2004), and is reprinted by kind permission of the author and editor.

side had to watch against reflow, i.e. changes in linebreaks. For TEX users, this practice was no option at all.

TEX users have for a long time been using Ghostscript for previewing, converting and printing PostScript. However, most printshops seem to have been unaware of such tools. And without such tools, a PostScript file is pretty much a black box.

Then Adobe developed PDF, a derivative of PostScript, and has had some success in persuading the graphics industry that a PDF-based 'workflow' is the way to go. By now, it is not that hard to find printshops accepting jobs in PDF format.

2 PDF tools

PDF has been developed both as a more tractable format for print production and as a format for various interactive uses. Whereas PostScript is a full-fledged programming language, PDF lacks programming features. Presumably, this made it easier to write software for it, and we certainly have seen a flood of software for PDF. Just pay a visit to `www.planetpdf.com` to convince yourself.

These include of course the Adobe Acrobat programs: the free Reader (which is now named Adobe Reader) and the various commercial editions of Acrobat. All these commercial editions include Distiller for converting PostScript to PDF. As of this writing, the latest versions (6.xx) of the Reader and the other Acrobat programs are available only for Windows and Mac OS X.[1]

Other PDF tools include various third-party Acrobat plugins, for prepress functions such as color separation and page imposition, and for limited editing. There are also toolkits/libraries for programmers, some of them open source. There are also commercial and free alternative PostScript-to-PDF converters, Ghostscript not the least among them. Mac OS X Panther contains a command-line utility pstopdf which is quite good. Many programs now can generate PDF directly.

The principal freely available PDF readers are Ghostscript (via a suitable frontend such as gv or GSView), and xpdf. The latter is part of a suite. Xpdf itself requires X11, but the rest of the suite consists of some very useful command-line utilities which are also available for Win32. I'll mention some of them below.

3 Routes to PDF

The principal routes to generate PDF from TEX are:

- from TEX to dvi to PostScript, and then running the PostScript file through Distiller or another PostScript-to-PDF converter
- from TEX directly to PDF, using pdf[e]tex
- from TEX to dvi and then with dvipdfm[x] to pdf. Dvipdfm-cjk, a.k.a. dvipdfmx, offers extended support for CJK (Chinese/Japanese/Korean) languages with their huge character sets.

One reason for choosing the roundabout way via PostScript is when you use PostScript-specific features, such as the pstricks package, which haven't been adapted to PDF. Another reason is that you may need Distiller's extra prepress-related controls.

If you need pdftex-specific features but also Distiller's controls, then you can go from PDF to PostScript, and then back to PDF. For the first conversion, you can use either Adobe Reader or Ghostscript or pdftops (from the xpdf tools suite); for the second one, use either Distiller or one of its alternatives. This usually works just fine.

3.1 Ghostscript as a PDF generator

Many of Distiller's prepress-related controls are also available via Ghostscript; a fairly thorough description can be found in the ps2pdf manual that is included in the Ghostscript distribution.

4 Preventing font problems

Acrobat used to come with a base set of fonts: Courier, Helvetica, Times, Symbol and Zapf Dingbats. Therefore, these fonts were customarily not embedded. To the dismay of the TEX community, in Acrobat 4 Times was replaced with Times New Roman, and Helvetica with Arial. Grudgingly, we concluded that it was better to avoid ambiguity and embed *all* fonts for print, including the base 14, and just put up with the increase in file size. Fortunately, this version of Acrobat also introduced joboptions files, which are named sets of Distiller settings. This made it easier to switch between generating unambiguous pdfs for prepress and small pdfs for online viewing, where you may prefer to exclude the base-14 fonts.

Another point of concern is METAFONT-generated bitmapped fonts. Although these may look fine in print, they usually look pretty bad on screen, and PDF validation tools will probably flag them as undesirable or illegal.

Font embedding is controlled by map files. For teTEX/fpTEX/TEX Live, these used to be located under `texmf/dvips/` and `texmf/pdftex/`, but are being relocated to `texmf/fonts/map/`*engine*, *engine*

[1] Since then, Adobe has released Acrobat 7, in which the Reader is once more available for Linux.

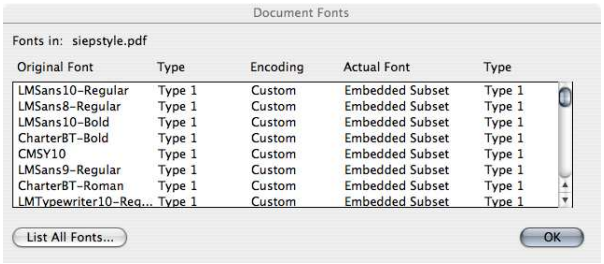


Figure 1: Adobe Reader: Document Fonts

being e.g. dvips or pdftex. Make sure that the relevant map files contain entries for the Computer Modern fonts, and that all entries contain a font filename:

```
ptmr8r NimbusRomNo9L-Regu
  " TeXBase1Encoding ReEncodeFont "
  <8r.enc <utmr8a.pfb
```

(a single line), rather than

```
ptmr8r Times-Roman
 " TeXBase1Encoding ReEncodeFont " <8r.enc
```

The first version downloads the URW Times clone included in most free TEX distributions; the second references a version of Times which should be available to either Acrobat or the printer/typesetter.

As of the 2003 editions of teTEX/fpTEX/TEX Live, map files are generated with a utility updmap, and configured either by editing `web2c/updmap.cfg` or with command-line parameters. Also check `texmf/pdftex/config/pdftex.cfg` (for 2003 and earlier) to see which map files are used by pdftex.

Changes are planned for future releases, so be sure and check the documentation if things don't work out.

As to MikTEX: The manual mentions the file `updmap.cfg` for manual configuration and the command `initexmf --mkmaps` for forcing regeneration of the map files.

You can check your fonts with the Reader by *first scrolling through the entire document* and then either click File/Document Properties/Fonts... or by clicking the right-pointing arrow above the vertical scrollbar and select Document Fonts...; see figure 1.

If Acrobat doesn't support your platform, then use pdffonts from the xpdf suite instead:

```
> pdffonts siepstyle.pdf
name                  type   emb sub uni object ID
--------------------- ------ --- --- --- ---------
GZLRCN+LMSans8-Regular Type 1 yes yes no      10  0
EQOQAE+LMSans10-Bold  Type 1 yes yes no      13  0
...
```

5 Preventing problems with figures

Included figures also may cause problems:

- Fonts: keep in mind that included pdfs may also contain fonts and font problems. If a font is embedded in a pdf that you are trying to include, and pdftex complains that it can't find the font, it may be that the font is present in the map file but absent from your installation. In that case, create a custom version of the map file without the entry. This will hopefully no longer be a problem with pdftex version 1.20.
- Lines with width 0, as produced by several graphics programs when you select 'hairline'. Width 0 means one pixel wide. This looks fine with 300 dpi output from a desktop printer, but becomes completely invisible with high-resolution typesetter output. A width of 0.3 pt should be safe.
- Resolution of pixel-based images. With the wrong Distiller settings, they might inadvertently get downsampled to screen resolution.
- Inappropriate use of jpeg:

The left picture is a jpeg of 1138 bytes, the right one a png of 571 bytes. Jpeg is fine for photographs, but if your image contains large solid areas and sharp transitions, then lossless compression such as used by the png format is probably better.

Some of these problems can be spotted by zooming in on your figures in the Reader.

6 Page size and other properties

With the traditional LATEX plus dvips plus Distiller route, you needed to tell all three programs about the desired page size. With pdftex, you only need to specify page dimensions once, in your TEX source. Use the pdftex primitives `\pdfpagewidth` and `\pdfpageheight`, or use the geometry package.

While you are at it, ensure also that the PDF version is no higher than it needs be, since your printshop may not have the latest versions of everything. A good version to aim for is 1.3, which corresponds to Acrobat 4. This can be set either in `pdftex.cfg` or in your TEX source:

```
\pdfoptionpdfminorversion=3
```

Again, you can check either with the Reader, using either File/Document Properties/Summary or the

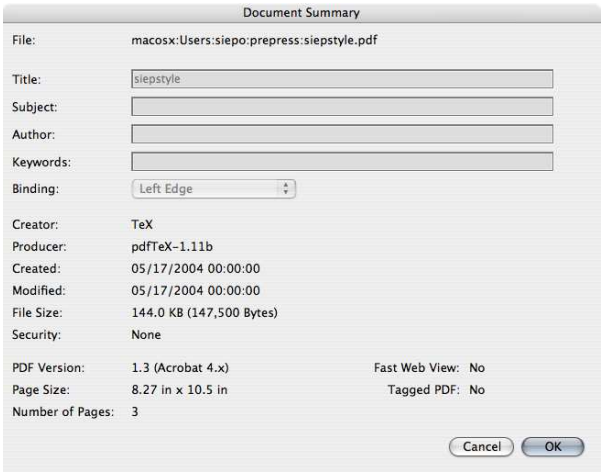


Figure 2: Adobe Reader: page dimensions and PDF version

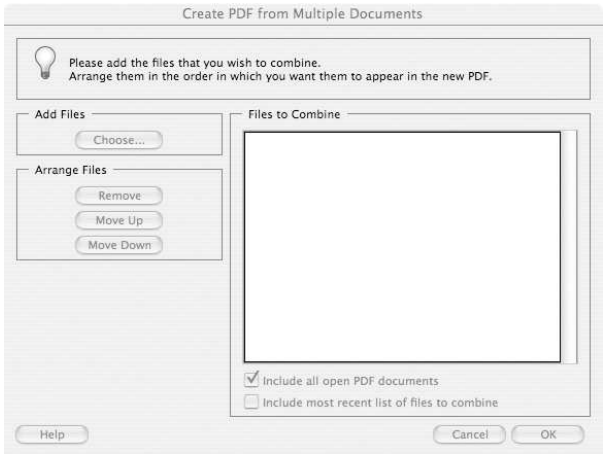


Figure 3: Combining pdfs interactively with Acrobat

Document Summary tab under the right-pointing arrow above the vertical scrollbar; see figure 2. With the xpdf utilities, use pdfinfo:

```
> pdfinfo siepstyle.pdf
Title:          siepstyle
Creator:        TeX
Producer:       pdfTeX-1.11b
CreationDate:   20040601
ModDate:        20040601
Tagged:         no
Pages:          3
Encrypted:      no
Page size:      595.3 x 756 pts
File size:      148171 bytes
Optimized:      no
PDF version:    1.3
```

Page dimensions (pts) are in 'big points'.

7 Combining documents

With a journal or a proceedings, it often isn't practical to compile the entire document in a single TEX run. So you may end up with a separate pdf for each paper, which you have to combine into a single pdf somehow.

7.1 With TEX

If you have separate pdfs of arbitrary origin then TEX can collate them for you: either use the LATEX package pdfpages or use the ConTEXt utility texexec with the `--pdfarrange` switch. Including a file with pdfpages can be as simple as

```
\usepackage{pdfpages}
...
\includepdf[pages=-]{APaper}
\includepdf[pages=-]{AnotherPaper}
...
```

The teTEX/fpTEX/TEX Live distributions contain the necessary documentation for both pdfpages and texexec.

7.2 With a Distiller driver file

Another option is to generate PostScript files and feed Distiller a driver file which loads them. Such a driver file may look as follows:

```
%!
/prun {
  /mysave save def     % save first
  dup = flush          % Shows name of PS file
  RunFile              % builtin Distiller proc
  clear cleardictstack % Cleans up
  mysave restore       % Restores save level
} def

(c:/temp/apaper.ps) prun
(c:/temp/anotherpaper.ps) prun
...
```

This is documented in the Acrobat documentation; see `RunDirEx.txt` and `RunFilEx.ps`. The location of these files varies per version and platform.

If you use this approach, it is best *not* to let dvips subset fonts. That way, Distiller can create a single subset of each font for the entire volume, leading to a smaller pdf.

7.3 With Acrobat

Finally, Acrobat lets you combine pdfs interactively (see figure 3), but since you probably end up repeating the process quite a few times, the other options will almost certainly be more convenient.

8 Color separation

If you want your document to be printed in color, then the printshop has to prepare one plate for each ink. For 'full color', these inks are usually cyan, magenta, yellow and black (CMYK). This style of color printing is called process color. The best way by far is to let the printshop handle this itself. After all, they should have the specialized software and the know-how.

However, TEX users do have a few options:

8.1 Using macros

You can generate a page several times, each with different definitions for colors:

```
\def\doseparation#1{%
  \ifcase #1 % composite
     \def\sepcyan{cyan}%
     \def\sepblack{black}%
     \def\sepfigure{CKfigure}%
  \or % cyan
     \def\sepcyan{black}%
     \def\sepblack{white}%
     \def\sepfigure{Cfigure}%
       % cyan rendered as black; black omitted
  \or % black
     \def\sepcyan{white}%
     \def\sepblack{black}%
     \def\sepfigure{Kfigure}% cyan omitted
  \fi
{\color{\sepcyan} Text in cyan\par}
{\color{\sepblack} Text in black\par}
\includegraphics{sepfigure}\newpage}

%\doseparation0
%  for colored output; omitted for separations
\doseparation1
\doseparation2
```

Note that this requires preseparated external figures.

ConTEXt contains built-in macro-based color separation functionality; see `www.pragma-ade.com/general/manuals/msplit.pdf`.

8.2 Using dvips and colorsep.pro

The TEX Live distribution contains a PostScript header file `texmf/dvips/colorsep/colorsep.pro` for separation of process colors. If you run dvips as follows:

```
dvips -b 4 -h colorsep.pro filename
```

then dvips produces each page four times (`-b 4` switch), and each time the header file colorsep.pro redefines colors appropriately for a given printing plate.

8.3 Using Acrobat 6 Professional

Acrobat 6 Professional also offers color separation via the Print menu. I encountered some glitches so I recommend to have a really good look at the resulting PostScript or pdf file before submitting it to your printer.

9 Overprinting

When printing black over a colored background, color separation software typically sets the other plates to white. However, any misregistration on the press will lead to slivers of white, which might be quite distracting; see the picture below.

If the background is light enough, then you can ignore the effect, but in other cases it is better to do something about it. One solution is to use a modified black with other color components added:

```
\color[cmyk]{0,0.5,0,1}
```

Another solution is to tell PostScript or PDF to let the color continue underneath the black. This is called overprinting. For a LATEX style file and example which *tries* to implement this for dvips and pdftex, look at `http://tex.aanhet.net/overprint/`. You can judge the effect in Acrobat Pro, if you check Advanced/Separation Preview. Figure 4 shows this dialog in another context.

10 Spot colors

A popular use of color in a printed document is to print some elements such as headings or rules from a single premixed color. Printshops have books with color swatches to choose from. Pantone is the manufacturer and license holder of most of these swatch books. You can let one of the process colors, i.e. cyan, magenta or yellow, take the place of the spot color and tell the printshop which color you really want.

If you want spot color *in addition to* process color, then the above trick can't be used. However, ConTEXt offers real support for spot colors. You can do it as follows:

```
\definecolor[myspotcolor][c=.7,m=.2]
\definecolor[myspot][myspotcolor][p=1]
...
\color[myspot]{myspot}
```

Note the two-stage definition of `myspot`: if you want a separation plate for the spot color, you need to

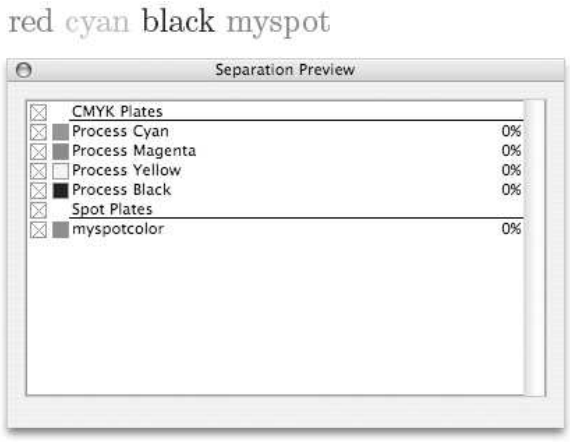


Figure 4: ConTEXt does spot colors in addition to CMYK

define `myspot` as a tint or fraction of a previously defined color. See also figure 4.

11 Color management

RGB colors are represented by three values for the three components, and process color by four values for the four process inks. These three or four values don't represent color itself but instructions for a device to apply certain colorants. The resulting color can and does depend on the device; we are all familiar with a wall of TV sets in an electronics store each displaying the same image with a different color cast. Matching screen colors with printed colors is an even worse problem. We all have seen how screen images can become disappointingly dull when printed; many brilliant screen colors simply cannot be reproduced in print.

Since graphics professionals tend to care about color consistency, color management systems have been introduced, which try to guarantee color consistency from device to device. This means either specifying color in some device-independent way or supplying device profiles to go with the color elements in your document. This is one area where open source doesn't have much to offer.

12 PDF/X and Certified PDF

PDF/X is an ISO standard for PDF files in prepress. There are two flavors: PDF/X-1a which allows process color and spot color, and PDF/X-3 which also accepts color-managed RGB. Since it is an ISO standard, you have to pay money to get the specification. However, you can download documentation and Distiller settings for free from `www.pdf-x.com`.

If you can avoid RGB color altogether, then it is possible to generate PDF/X with pdftex. However, don't convert existing images just for the sake of PDF/X conformance if you don't have to; check with your printshop first.

Code similar to the following should ensure that your pdf won't fail PDF/X for silly reasons:

```
\pdfpagewidth=595.3bp
\pdfpageheight=841.7bp
\pdfpageattr{/TrimBox [ 0 0 595.3 841.7] }

\pdfoptionpdfminorversion=3

\edef\pdfdate{%
  \the\year
  \ifnum \month < 10 0\the\month
    \else \the\month \fi
  \ifnum \day < 10 0\the\day
    \else \the\day \fi}
\pdfinfo{%
  /CreationDate (D:\pdfdate)
  /ModDate (D:\pdfdate)
  /Trapped (False)
  /GTS_PDFXVersion (PDF/X-3)
  /Title (\jobname)}

\pdfcatalog{
  /OutputIntents [ <<
    /Info (Euroscale Coated v2)
    /Type /OutputIntent
    /S /GTS_PDFX
    /OutputConditionIdentifier
       (OFCOM_PO_P1_F60)
    /RegistryName (http://www.color.org/)
    >> ]}
```

pdftex 1.11b already includes a creation date automatically. Hopefully, newer versions will do the same for modification date so that you can dispense with the date rigmarole altogether.

Acrobat Distiller also has options for color management and PDF/X; see figure 5.

Another initiative, from Enfocus Software, is Certified PDF. This is not just a set of requirements, but requires your pdfs to be stamped as certified by dedicated commercial software. I found no reference to this type of certification in the Acrobat documentation. See `www.certifiedpdf.net` for more information.

13 Preflight

The term preflight has come to be used for ensuring that your pdf is safe for production. I already mentioned a few simple checks that are available with

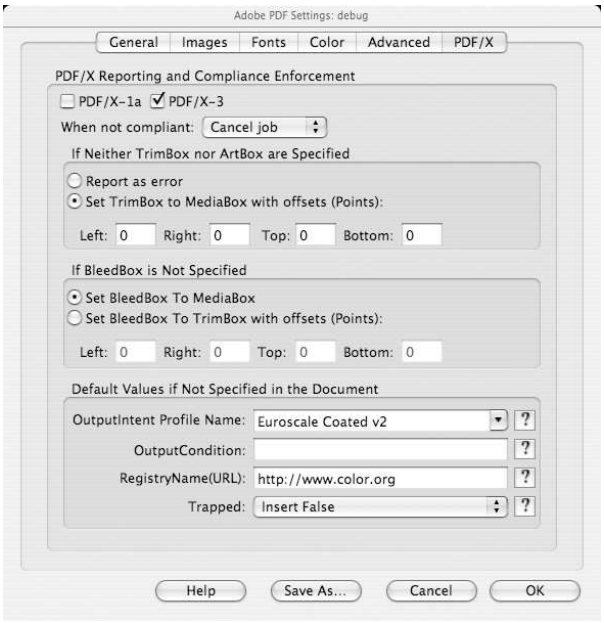


Figure 5: Distiller settings for PDF/X conformance. The Color tab (not shown) also contains relevant settings.

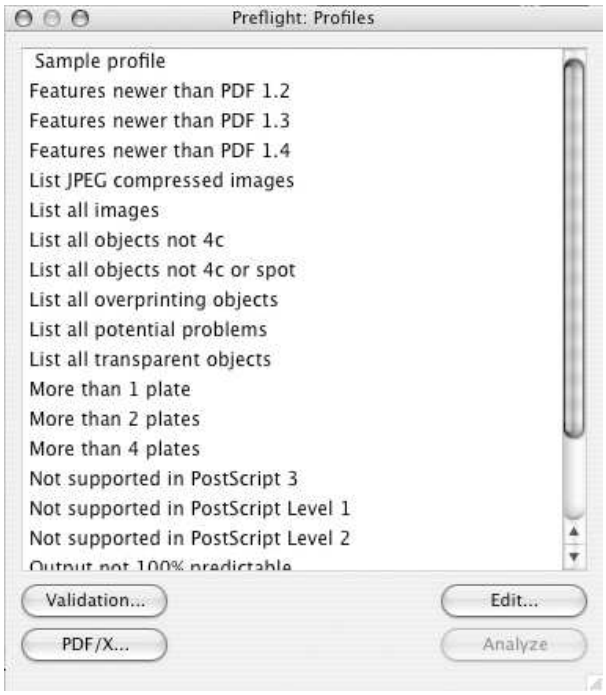


Figure 6: A long list of predefined sets of preflight profiles in Acrobat 6 Pro

the Reader and with the xpdf utilities.

Acrobat Professional has a lot of preflight options built-in, including checks on PDF/X compliance. They can be found under the Document menu. Just as with Distiller options, there are also named sets of preflight options; see figure 6.

Much of the Acrobat preflight code has been taken from Callas' PDF/X Inspector. There also used to be a free version of this tool, called PDF/X-3 Inspector.

14 Conclusion

The main points are to check what you can, and to discuss with your printshop in what form they want your document. Maybe they have a Distiller options file; even if you don't use Distiller, then it would still be useful to look at (these are plain ASCII files).

If there is color then it is highly desirable that the printshop be able to do the separations. The same is true for page imposition.

Keep also in mind that there are plenty of MS Office files which are being typeset somehow, so many printshops ought to be able to handle pdfs from outside the graphics industry.

All this having been said, I do believe that Acrobat Pro is a worthwhile investment if you can afford it at all.

15 URLs

Adobe	`www.adobe.com`
Planet PDF	`www.planetpdf.com`
Xpdf	`www.foolabs.com/xpdf/`
DviPDFm project	`project.ktug.or.kr/dvipdfmx/`
Color separation in ConTEXt	`www.pragma-ade.com/general/manuals/msplit.pdf`
overprint.sty	`tex.aanhet.net/overprint/`
PDF/X support	`www.pdf-x.com`
Certified PDF	`www.certifiedpdf.net`
Callas	`www.callas.de/en/`

◇ Siep Kroonenberg
`siepo@cybercomm.nl`

Automatic typesetting of formulas using computer algebra

Marcelo Castier and Vladimir F. Cabral

Abstract

This paper describes new procedures, written in the Mathematica® programming language, for quickly typesetting mathematical formulas in LaTeX syntax. Two main procedures provide direct interface with the user. The first of them obtains the LaTeX representation of a single formula. The second procedure analyzes a set of formulas, searching for common terms and symmetries, and breaks the original formulas input by the user in a series of calculations of intermediate terms. In either case, a list of symbols used in the formulas is automatically generated in LaTeX format. The procedures may speed up the writing of technical publications and eliminate common sources of error in their preparation.

Introduction

Several current tools can assist preparation of technical documents using computers. Voice recognition software such as ViaVoice™ and Dragon NaturallySpeaking® transform speech directly into typeset text with good accuracy. Literature references can be downloaded from databases, and software such as Natbib, Reference Manager®, and ProCite® will format them according to the rules of many scientific journals. Cross-referencing of tables, equations, and figures eliminates the need for their manual renumbering, if the manuscript has to be modified. However, authors usually typeset mathematical formulas manually, which can be tedious and time-consuming due to the need for careful reviews of complex expressions.

In fact, it may be more difficult to guarantee the correctness of a formula typeset for publication than its programmed version in a scientific language, such as Fortran or C. In the latter case, numerical tests can help locate programming errors. On the other hand, the verification of formulas typeset for publication is generally made by visual inspection.

An additional aspect related to typesetting formulas for publication is the preparation of lists of symbols. It is not unusual to find publications in which some symbols are missing from these lists.

Modern computer algebra systems (CAS), such as Maple™ and Mathematica, provide a user-friendly environment for symbolic computations, allowing the derivation of complex formulas. Moreover, both Maple and Mathematica have commands for exporting formulas to other programs in different formats. Therefore, they have the basic functionality needed for the automatic implementation of formulas, which has been used by some authors.

Motivated by the difficulty of manual symbolic computations in the area of general relativity, Klioner (1998) used Mathematica to develop a program for operations with indexed objects that can provide its results in TeX or LaTeX. Piecuch (1993) and Strange et al. (2001) used Maple to obtain LaTeX code in applications to problems in quantum chemistry. Maple was also used by Sharf (1996) for the generation of LaTeX code in the analysis of beam elements for the simulation of multibody systems. Weinzierl (2004) describes a new CAS, called gTybalt, which is freely available and has the possibility of producing TeX output. However, some operations, such as integration, are not implemented yet, which currently limits the applicability of the program. Talole and Pradke (2003) developed a program that exports text, numerical data, and plots from a Matlab® calculation to a LaTeX document. An important contribution in this area is the development of Mathscape (Barnett, 1998), which is a program in Mathematica for the automatic typesetting of formulas in LaTeX whose features are in many ways complementary to those available in the set of procedures presented here.

In this paper, we use Mathematica, and the comments henceforth about the ability to export formulas are limited to this CAS. Mathematica can export formulas as images, or in MathML or TeX formats. The use of images is inconvenient because some final editing of the formulas is often required. The MathML or TeX codes cannot be used as input in the current versions of Microsoft® Equation Editor or MathType™, which are the most commonly used equation editors for Microsoft Word. Therefore, the exchange of formulas between Mathematica and these editors that have graphical user interfaces is less flexible than would be desirable.

Use of MathML will probably spread in the future as a way of interchanging information about physical properties and models for their evaluation (Frenkel et al., 2004). However, we focus on the use of TeX or LaTeX directly, because the latter is the de facto standard used internally by many technical publishers. Mathematica has a command to generate the representation of a formula in TeX. Although very useful, this command only takes one formula at a time and does not generate the list of symbols, among other limitations.

Here, we present two new procedures. The first generates LaTeX code for a single formula. The second procedure performs a simultaneous analysis of

several formulas, identifies their common and symmetrical terms, and obtains the LaTeX representation as a sequence of intermediate formulas, thereby breaking the original expressions input by the user into a form that is more convenient for presentation. Both procedures automatically prepare a list of symbols, also coded in LaTeX, classifying them as Roman or Greek letters, or indexes.

These new procedures extend the capabilities of Thermath (Castier, 1999), a program whose current version contains approximately 6000 lines of code written in the Mathematica programming language. The original purpose of Thermath was the computer implementation of thermodynamic models for the calculation of physicochemical properties of mixtures, by providing complete subroutines automatically written in a scientific programming language. In a typical application, given a thermodynamic model, several properties are derived using computer algebra for operations such as derivation and integration in a Mathematica session. It often happens that the derived properties have formulas that are longer and more complex than the thermodynamic model that originates them. Using its internal procedures, Thermath analyzes these formulas and implements them automatically in a complete subroutine, with a drastic reduction in the need for manual coding.

Thermath has been extended to other applications such as the automatic implementation of expressions (Dominguez et al., 2002) in a format compatible with the INTBIS/INTLIB package for solving sets of nonlinear equations with interval analysis (Kearfott and Novoa, 1990), and the preparation of code for the simulation of separation equipment in the chemical industry (Alfradique et al., 2002).

The new procedures presented in this paper add the possibility of aiding in the preparation of technical documents, not only related to physicochemical properties but in many areas that require the manual typesetting of long formulas. These procedures perform extensive and intricate symbol manipulations in the expressions. Here, we present only a general description, and refer to the code, which is available from the authors on request, for all the details.

Automatic generation of LaTeX code for a single formula

Mathematica has a function called `TeXForm` that translates formulas into TeX syntax. Let us illustrate its usage with the typesetting of a simple formula: the van der Waals equation of state. Given that the emphasis here is not on the technical aspects of the equation of state, we refrain from discussing the meaning of its symbols. The `TeXForm` function is used as follows:

```
TeXForm[P == R*T/(v - b) - a/v^2]
```

This command produces one line of output, broken here in additional lines only to fit the column width of *TUGboat*, *as also done in some of the other examples of this paper*.

```
P = -\left( \frac{a}{v^2} \right)  +
 \frac{R\,T}{-b + v}
```

This output, obtained in a Mathematica session, can be cut and pasted into a document. Even though this certainly reduces the need for manual typesetting, several improvements are possible, such as automatically assigning a label to the formula for cross-referencing and generating a list of symbols.

Thermath contains a procedure, `prinTeX`, that performs several actions: (1) prepares lines that load the LaTeX breqn package for the automatic breaking of long formulas in several lines; (2) prepares an equation label containing six randomly generated digits; (3) identifies all the symbols that appear in the equation, classifying them as Roman or Greek letters or indexes; (4) implements the formula using the Mathematica function `TeXForm`. The verbatim input in Mathematica is:

```
prinTeX[P == R*T/(v - b) - a/v^2]
```

The verbatim LaTeX code obtained as output is:

```
%
%The following lines should be placed after the
%\documentclass {class} line
%in the LATEX file.
%
\usepackage[cmbase]{flexisym}
\usesymbols{msabm}
\usepackage[debug]{breqn}
\setkeys{breqn}{compact}
%
%The following lines should be placed where
%the formula should appear in the text.
%
\begin{dmath}\label{e:eqn485282}
P = -\left( \frac{a}{v^2} \right)  +
\frac{R\,T}{-b + v}
\end{dmath}

%
%The following lines create the list of symbols.
%
\section*{List of Symbols}
%
%

\subsection*{Roman Letters}
%
```

```
$a$          \\
$b$          \\
$P$          \\
$R$          \\
$T$          \\
$v$          \\
%
%The list of symbols was successfully created.
```

The parts of this output, i.e., the loading commands for breqn, the formula, and the list of symbols can then be cut and pasted at their proper positions in a LaTeX document. For example, the loading commands for breqn were pasted at the beginning of the LaTeX document of this paper for *TUGboat*. The formula and the list of symbols were pasted here. Due to differences between the `ltugboat` document class used by *TUGboat* and the `elsart` document class from Elsevier, used for testing the software, it was necessary to manually add a `\newline` command in the line after the subsection names to improve formatting. Upon processing with the LaTeX compiler, the following text is obtained:

$$P = -\left(\frac{a}{v^2}\right) + \frac{RT}{-b+v} \tag{1}$$

List of Symbols

Roman Letters

a
b
P
R
T
v

If several formulas are prepared using `prinTeX`, the loading commands for breqn need to be pasted only once at the beginning of the LaTeX document and the lists of symbols of each formula have to be manually combined to consolidate the list of symbols of the document, which most commonly constitutes one of the final sections of technical papers.

Even though Equation 1 is correct, this example also illustrates one of the difficulties with CAS. Comparing the input and output, we observe that Mathematica interchanges the order of the two terms in the right hand side of the equation and does the same in the denominator $(v-b)$. Therefore, the formula is not printed as usually represented in the literature. Unfortunately, there seems to be no straightforward solution to this problem in Mathematica. Barnett (1998) developed a function called `toEach`, in the context of Mathscape, that can reverse the order of operations, but this function was not tested here. Instead, we circumvented the problem by using the command `HoldForm`, which keeps an expression unevaluated and therefore not subject to the automatic reordering of terms performed by Mathematica. The corresponding input is:

```
prinTeX[HoldForm[P == (R*T)/(v - b) - a/v^2]]
```

After processing this input with the LaTeX compiler, the traditional representation of the van der Waals equation of state is obtained:

$$P = \frac{RT}{v-b} - \frac{a}{v^2} \tag{2}$$

The list of symbols remains unchanged and, for this reason, is not presented.

Let us consider a more complex example, which requires integration of the van der Waals equation of state at constant temperature. The Mathematica input is:

```
P = (R*T)/(v - b) - a/v^2
prinTeX[W == HoldForm[Integrate[P,
{v,alpha, beta}]] ==
Simplify[Integrate[P, {v, alpha, beta}]]]
```

The output is:

$$W = \int_\alpha^\beta P\,dv = a\left(-\left(\frac{1}{\alpha}\right) + \frac{1}{\beta}\right) + RT\,\ln\left(\frac{b-\beta}{-\alpha+b}\right) \tag{3}$$

List of Symbols

Roman Letters

a
b
R
T
W

Greek Letters

α
β

Note that the command `HoldForm` leaves the integral unevaluated between the two equal signs. The list of symbols now contains a subsection where the two Greek letters used as integration limits are identified. A current limitation of the pattern matching procedure implemented in `prinTeX` is that it does not identify dummy variables. For instance, v is a dummy integration variable, and it is not included in the list of symbols.

Automatic generation of LaTeX code for multiple formulas

In many cases, several formulas are derived using computer algebra during a Mathematica session, and instead of generating LaTeX code for each formula, it may be more convenient to generate code

for all of them simultaneously. For this, we developed two procedures that are used sequentially: `ordeqTeX` and `createTeX`.

Procedure `ordeqTeX` analyzes the expressions to be represented in LaTeX. During this analysis, subexpressions that appear several times are recursively identified and ordered, in such a way that a meaningful calculation sequence of subexpressions is obtained. The procedure also searches for subexpressions with symmetrical indexes. In addition, `ordeqTeX` can sort the subexpressions according to their dependence with respect to a list of variables input by the user, which may be useful for authors writing about the functional structure of their formulas.

Procedure `ordeqTeX` is similar to a procedure already present in the first version of Thermath, `ordeq`, whose logical analysis of expressions is discussed by Castier (1999). An important difference between them is the level of fragmentation into subexpressions. Consider, for instance, that $1/x$ is a subexpression that appears several times in a large formula. For automatic programming in a numerical language, such as Fortran or C, it is convenient to store the result of the subexpression in an intermediate variable, in order to avoid unnecessary calculations. However, a large number of simple substitutions may obscure the presentation of a formula in a text. For this purpose, the formulas should be less fragmented than for numerical calculations — but to what extent is a subjective decision.

In `ordeqTeX`, simple fractions such as the example in the above discussion, powers in which the exponent is a number, multiplications and sums of only two terms are not replaced by intermediate variables. However, the pattern matching algorithm implemented in procedure `ordeqTeX` can be easily changed to use other criteria.

Procedure `ordeqTeX` prepares detailed information about the structure of the formulas and of the subexpressions, which is then passed to procedure `createTeX`. This procedure prepares a LaTeX code that presents all subexpressions and final expressions in a feasible computation sequence.

Even though `createTeX` replaces long formulas by sequences of subexpressions, it may happen that some of these subexpressions are longer than one line of LaTeX output. In order to avoid the need for manual intervention for breaking long lines, we used the (freely available) LaTeX package breqn, which automatically chooses the breakpoints. For convenience, the output of the `prinTeX` and `createTeX` procedures includes commands for loading and using breqn, and each formula is given a unique label for cross-referencing. In the case of `prinTeX`, a six-digit random number is used to generate the label. In the case of `createTeX`, the number results from joining the name of the set of formulas being implemented, specified by the user, with a unique sequential number assigned to each subexpression.

For the preparation of the list of symbols, we use the fact that expressions are internally stored as trees in Mathematica. Using a recursive procedure developed for Thermath, the trees are spanned, searching for all the symbols they contain. From this first list of symbols, those that represent intrinsic Mathematica functions or operators, such as `Plus`, `Times`, `Log`, `Exp`, etc., are discarded.

To distinguish between intrinsic Mathematica functions and symbols entered by the user, we use the Mathematica function `Attributes` to test each symbol. Intrinsic Mathematica functions have nonempty lists of attributes, whereas a symbol entered by the user has an empty list of attributes, unless a special attribute has been explicitly assigned to the symbol. It is usually unnecessary to specify attributes to symbols, but it may happen, for example, that some matrices are intrinsically symmetrical. In these cases, it is convenient to assign the Mathematica attribute `Orderless` to the symbols that represent these values. Therefore, the symbols entered by users are located as those without attribute or only with the `Orderless` attribute.

From the remaining list, the symbols that represent numerical values, either integer, real or complex, are also discarded. At this point, the list will only have the symbols entered by the user. It then remains to verify which of the symbols are variables and which are only indexes. The convention adopted in the identification procedure is that a symbol is an index when it is the argument of a symbol entered by the user. For instance, in the expression `Sin(x(i))`, `x` is the argument of an intrinsic Mathematica function, `Sin`, and therefore is not an index. On the other hand, `i` is the argument of `x`, which is not an intrinsic Mathematica function. Therefore, `i` is assumed to be an index.

As an example, let us consider the simultaneous analysis of two simple formulas, with some characteristics that help illustrate the features of the package presented here. The input for this example is:

```
f = Sin[x[i]*x[j]] + Cos[y[k]*y[m]];
g = Cos[x[i]*x[j]] + Exp[Sin[x[i]*x[j]]];
formulas = {f, g};

analyzedformulas = ordeqTeX[formulas, {}];
createTeX[fg, analyzedformulas,
  {HoldForm[f], HoldForm[g]}];
```

In this input, the two formulas `f` and `g`, are joined in a single list, `formulas`, which is passed to `ordeqTeX`. The second argument of this call specifies how subexpressions should be grouped according to their functional dependence. In this example, an empty list is specified, meaning that no specific grouping is required.

In procedure `createTeX`, the first argument, `fg`, represents a user-defined name for the set of formulas being implemented. `createTeX` uses this name to prepare a unique label for each subexpression to be used for cross-referencing. The second argument is a list containing several pieces of information about the formulas prepared by procedure `ordeqTeX`. The last argument specifies that the left-hand side of the equations should appear as $f =$ and $g =$. The LaTeX output (slightly edited) is:

```
%
%Formulas for model: fg
%
%
%     if (
%          green(1)   .or.
%          green(2)
%        ) then
%

\smallskip

\begin{dmath}\label{e:fgeqn1}
      w_{2}\left(i,j \right)=
      \sin \left(x\left(i \right)\,
      x\left(j \right) \right)
\end{dmath}
%
%Note symmetry: w$2(j,i)=w$2(i,j)
%=============
%
%
%     end if
%
%
%     if (
%          green(1)
%        ) then
%

\smallskip

\begin{dmath}\label{e:fgeqn2}
      f\left(i,j,k,m \right)=
      \cos \left(y\left(k \right)\,
      y\left(m \right) \right) +
      w_{2}\left(i,j \right)
\end{dmath}
%
%Note symmetry: f(i,j,m,k)=f(i,j,k,m)
%=============
%
%
%Note symmetry: f(j,i,k,m)=f(i,j,k,m)
%=============
%
%
%Note symmetry: f(j,i,m,k)=f(i,j,k,m)
%=============
%
%
%     end if
%
%
%     if (
%          green(2)
%        ) then
%

\smallskip

\begin{dmath}\label{e:fgeqn3}
      g\left(i,j \right)=
      e^{w_{2}\left(i,j \right)} +
      \cos \left(x\left(i \right)\,
      x\left(j \right) \right)
\end{dmath}
%
%Note symmetry: g(j,i)=g(i,j)
%=============
%
%
%     end if
%
%
%The set of formulas was successfully created.
%
%The following lines create the list of symbols.
%
\section*{List of Symbols}
%
%

\subsection*{Roman Letters}
%
$f$          \\
[...]
%

\subsection*{Indexes}
%
$i$          \\
[...]
%
%The list of symbols was successfully created.
```

Note that the LaTeX output contains a variable of the form w_n that is automatically generated to represent an intermediate value. This output

also contains several comments that aim at helping authors to discuss the structure of their formulas, should this be desired. The parts of the output flagged with `green(1)` are relevant for the calculation of the first output variable, f, whereas `green(2)` provides a flag for the calculation of g. The output also indicates the existence of symmetry. For instance, it indicates that variables w_2 and g are symmetrical with respect to permutations of the indexes i and j, and that variable f is symmetrical with respect to some permutations of its indexes.

Compilation with LaTeX produces the following output:

$$w_2\,(i,j) = \sin\left(x\,(i)\;x\,(j)\right) \tag{4}$$

$$f\,(i,j,k,m) = \cos\left(y\,(k)\;y\,(m)\right) + w_2\,(i,j) \tag{5}$$

$$g\,(i,j) = e^{w_2(i,j)} + \cos\left(x\,(i)\;x\,(j)\right) \tag{6}$$

List of Symbols

Roman Letters

f
g
w_n
x
y

Indexes

i
j
k
m

Note that each variable appearing on the left hand side of Equations 4, 5, and 6 received the correct indexes automatically and that all the variables used in the formulas were included in the list of symbols. The exponential and cosine functions appear in reverse positions in the output compared to the input, as a result of the automatic reordering of expressions performed by Mathematica. Unlike the `prinTeX` command that was designed handle a single formula, the typical use of commands `ordeqTeX` and `createTeX` is in Mathematica sessions where several formulas are derived using computer algebra. In this context, the user has less control of the ordering used by Mathematica to present the formulas. Therefore, even though the formulas are correctly translated into LaTeX, a current limitation is that the formulas may need to be manually edited if some specific order of terms is desired in the LaTeX document.

We successfully tested the procedures discussed here with sets of formulas that are much more complex than those used in these examples. In some cases, especially when there are rather long formulas, a final manual editing step may be necessary to improve layout; the `[layout=RHS]` option of the breqn package proved particularly useful in these cases.

Conclusions

This paper presented new procedures, written in the Mathematica programming language, that automatically generate a representation of formulas in LaTeX with the corresponding list of symbols. There is the option of generating LaTeX code for a single formula or for a set of formulas. In the latter case, a comprehensive analysis of the formula structures allows the identification of common and symmetrical terms. Therefore, if one uses Mathematica as a computational environment for the symbolic and numerical calculations in a given project, it is possible to quickly obtain an exact representation, in LaTeX, of the formulas used and the list of symbols. The procedures may speed up the writing of technical publications and eliminate common sources of error in their preparation.

Acknowledgments

The authors thank Profs. Veronica M.A. Calado and Frederico W. Tavares (Universidade Federal do Rio de Janeiro, Brazil) for their suggestions. The Brazilian agencies CNPq and FAPERJ provided financial support for this research.

Code availability

The procedures developed in this work are available from the authors on request. The procedures were developed and tested using Mathematica 4.1, version 0.94 of the LaTeX package breqn, and Elsevier document classes.

References

Alfradique, M. F., R. O. Espósito, and M. Castier. "Automatic generation of procedures for the simulation of multistage separators using computer algebra". *Chemical Engineering Communications* **189**(5), 657–674, 2002.

Barnett, M. P. "Mathscape — Combining Mathematica and TeX". *TUGboat* **19**(2), 147–156, 1998.

Castier, M. "Automatic implementation of thermodynamic models using computer algebra". *Computers and Chemical Engineering* **23**(9), 1229–1245, 1999.

Dominguez, A., J. Tojo, and M. Castier. "Automatic implementation of thermodynamic models for reliable parameter estimation using computer

algebra". *Computers and Chemical Engineering* **26**(10), 1473–1479, 2002.

Frenkel, M., R. D. Chirico, V. V. Oiky, K. N. Marsh, J. H. Dymond, and W. A. Wakeham. "ThermoML — An XML-based approach for storage and exchange of experimental and critically evaluate thermophysical and thermochemical property data. 3. Critically evaluated data, predicted data, and equation representation". *Journal of Chemical and Engineering Data* **49**(3), 381–393, 2004.

Kearfott, R. B. and M. Novoa. "INTBIS, A Portable Interval Newton Bisection Package". *ACM Transactions on Mathematical Software* **16**(2), 152–157, 1990.

Klioner, S. A. "New system for indicial computation and its applications in gravitational physics". *Computer Physics Communications* **115**(2-3), 231–244, 1998.

Piecuch, P. "Maple Symbolic Computation of the Long-Range Many-Body Intermolecular Potentials — 3-Body Induction Forces Between 2 Atoms and A Linear Molecule". *International Journal of Quantum Chemistry* **47**(4), 261–305, 1993.

Sharf, I. "Geometrically non-linear beam element for dynamics simulation of multibody systems". *International Journal for Numerical Methods in Engineering* **39**(5), 763–786, 1996.

Strange, R., F. R. Manby, and P. J. Knowles. "Automatic code generation in density functional theory". *Computer Physics Communications* **136**(3), 310–318, 2001.

Talole, S. E. and S. B. Pradke. "Generating LaTeX documents through Matlab". *TUGboat* **24**(2), 245–248, 2003.

Weinzierl, S. "gTybalt — A free computer algebra system". *Computer Physics Communications* **156**(2), 180–198, 2004.

◇ Marcelo Castier
Escola de Química, Universidade Federal do Rio de Janeiro
Rio de Janeiro, RJ, 21949-900
Brazil
castier@eq.ufrj.br

◇ Vladimir F. Cabral
Departamento de Engenharia Química, Universidade Estadual de Maringá
Maringá, PR, 87020-900
Brazil
vfcabral@yahoo.com.br

Graphics

ePiX: A utility for creating mathematically accurate figures

Andrew D. Hwang

1 Introduction

Mathematical and scientific writing call for figures that accurately and attractively integrate typography and numerical data. Widely-used commercial and non-commercial drawing programs exist, as do dozens of lesser-known utilities. This article describes an addition to the list: ePiX, a collection of command line utilities for creating mathematically accurate, logically structured, camera-quality 2- and 3-dimensional figures and animations in LaTeX. Despite superficial similarities with existing programs, ePiX fills a distinct niche in the ecosystem of drawing software by providing a bridge between the powerful numerical capabilities of C++ and the high-quality typesetting of LaTeX.

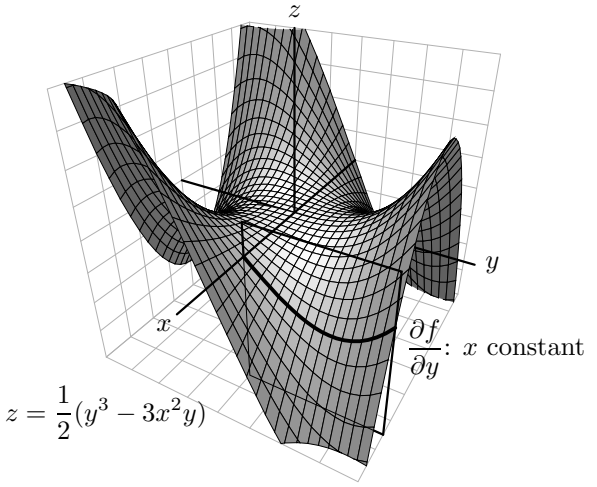


Figure 1: A surface with simulated transparency.

ePiX's relationship to a graphical drawing program is analogous to LaTeX's relationship to a word processor. A logically structured input file is prepared in a text editor, then compiled into a plain text (eepic) file that is included into a LaTeX document. Optionally, the figure may be processed into eps or pdf. This note focuses on the user interface, though certain issues of implementation arise of necessity.

ePiX's strengths include:

- Ease of use: Figure objects are specified by simple, mnemonic commands that refer to a natural coordinate system.

- Quality of output: ePiX creates mathematically accurate line figures whose appearance matches that of LaTeX. Typography is added to an ePiX figure as easily as to a LaTeX picture environment. The mechanism for text placement is robust under changes of scale.
- Wide availability: ePiX runs on platforms with a C++ compiler and the GNU shell bash, particularly on GNU/Linux, Mac OS X, Windows (Cygwin), FreeBSD, and Solaris. An output file may be incorporated into a document on any platform that supports LaTeX.
- Programming: ePiX's input is a widely-spoken, easily-learned programming language. Even simple figures can benefit from logical structuring, while complex figures may employ algorithms and generate their own numerical data.
- Extendability: Users can write custom code and incorporate the functionality with a command line switch or a Makefile. This feature, suggested by Andrew Sterian, endows ePiX with the computational power of C++.
- Economy of storage and transmission: A compressed tar file of the LaTeX sources and compiled eepic files is typically a small fraction of the size of a compressed PostScript file or a tarball containing eps files, making ePiX output particularly attractive for archiving.
- License: ePiX is *Free Software*, published under the GNU General Public License.

This note focuses on general issues of image creation and ePiX's approach to integrating numerical and algorithmic capabilities with high-quality typography. The project home page has source code, documentation, sample images, and animations:

http://mathcs.holycross.edu/~ahwang/current/ePiX.html

The latest stable version is also available from CTAN (in graphics/epix). Please visit the project page for a more thorough showcase of ePiX's capabilities.

I am grateful to Jay Belanger, Robin Blume-Kohout, Andrew Sterian, and Gabe Weaver for detailed and insightful design discussions and advice.

2 Source and Output Files

In LaTeX, a document preamble specifies the default appearance and sets up an environment by including packages and defining macros, while the body contains commands that generate the actual output. Similarly, an ePiX preamble (Figure 2) accesses library code and defines symbolic constants and functions that reflect the internal structure of the figure,

```
#include "epix.h"   // analogous to \usepackage
using namespace ePiX;

// function definition
double f(double x) { return x/(1-x*x); }

int main()
{
  unitlength(".85in");  // LaTeX unitlength
  picture(P(3, 1.5)); // printed size

  // specify corners; depict [-2,4] x [-4,4]
  bounding_box(P(-2,-4), P(4,4));

  begin();            // picture starts here
  crop();             // crop to bounding_box

  dashed();           // draw dashed lines
  line(P(-1, y_min), P(-1, y_max));
  line(P( 1, y_min), P( 1, y_max));

  solid();            // use solid lines
  h_axis(P(x_min, 0), P(x_max, 0), x_size);
  v_axis(P(0, y_min), P(0, y_max), y_size);

  h_axis_labels(P(x_min, 0), P(x_max, 0),
                0.5*x_size, P(-2,2), tl);
  bold();   // draw in bold (fonts unaffected)
  plot(f, x_min, x_max, 120);     // function plot

  label(P(2,3), P(0,0),
        "$y=\\displaystyle\\frac{x}{1-x^2}$");
  end();
}
```

Figure 2: An ePiX source file, cf. Figure 3.

while the body contains commands that adjust the appearance of objects and write the output file.

Body commands include objects, labels, and attribute declarations. ePiX supplies standard geometric primitives: points, lines, circles, spheres, planes, quadratic and cubic splines, ellipses and arcs, arrows, polygons and polylines, and coordinate grids. In addition, ePiX provides plotting: graphs, parametric curves and surfaces, data from files, vector fields, derivatives and integrals, and solutions of ordinary differential equations. Basic geometric objects can be constructed and used in mathematically natural ways, such as finding the intersection point of two lines, constructing a circle through three noncollinear points, or drawing the tangent line to a function graph.

Four internally documented shell scripts constitute the user interface: epix (creates eepic files), elaps (converts ePiX and eepic files to eps, pdf, or PostScript), flix (creates png images and mng animations), and laps (converts LaTeX to PostScript).

3 Design

The notion of "ideal" drawing software is too dependent on authors' individual needs and preferences to

be meaningful. Nonetheless, commonly useful features can be identified. ePiX does not satisfy all the criteria below, but its development has proceeded with these goals in mind.

3.1 Capabilities

A general-purpose command-driven drawing utility provides three basic services: an input language, a set of data structures for representing figure objects and their attributes, and output routines. The input language should be easy to learn and use, yet powerful, flexible, and extendable. Frequently-encountered objects and algorithms should be represented natively, allowing users to program (when necessary) in a high-level language. Both 2- and 3-dimensional figures should be supported. A variety of output file types should be available, so that the resulting images can be exchanged easily, used in printed documents, or published on the Web.

Less technical but equally important are issues of convenience and freedom. A program should supply sensible defaults, so that simple figures can be drawn without micro-management. At the same time, figure attributes should be modifiable with short, easily-remembered commands. Users' files should compile quickly, preferably in no more than a couple of seconds on a moderately fast machine. Output files should be small, perhaps tens of KB, yet of high typographical quality. The program should be widely available, and free from proprietary algorithms and file formats.

3.2 Logical Structuring and Input

Mathematical figures represent structured information. Bitmapped images, and to a lesser extent eps files, discard this structure. By contrast, a programming language exploits logical structure through use of symbolic constants, data structures, functional relationships, and algorithms, including control statements, loops, and recursion. A high-level figure description language is potentially both efficient and convenient, for the same reasons that a Taylor polynomial compactly encodes a trig table. Naturally, users do not want to learn a new language in order to create figures, but software can accommodate users by providing intuitively-named functions that implement common figure objects. Ultimately, however, a language that provides plotting and other algorithmic and numerical capabilities must utilize more complex syntax. To ease the learning curve, a scene description language might piggyback itself onto a widely-used programming language, such as C++, Fortran, or Lisp.

ePiX attempts to meet these goals by furnishing a user-friendly interface to C++, harnessing its speed, flexibility, and computational power to the creation of mathematical figures. An ePiX source file is a compact, high-level scene description written in C++. Even moderately complicated figures require no prior knowledge of C++, and the source code comes with dozens of samples files suitable for study and experimentation.

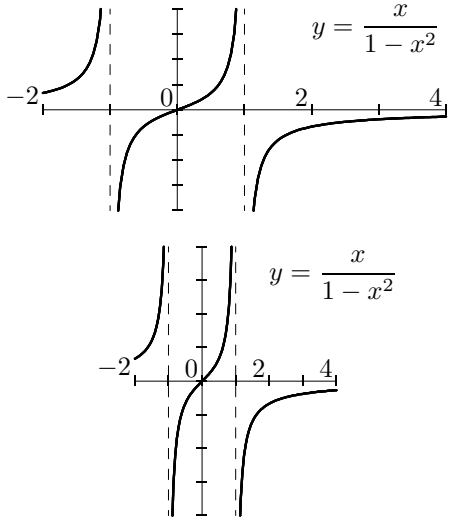


Figure 3: Rescaling: Two figures generated from the input file in Figure 2.

3.3 Page Coordinates and Resizing

Logical markup is fundamental to LaTeX: a document does not directly specify its visual appearance, but relies on packages loaded at compile time. Mathematical figures benefit similarly from logical structuring. Designing and writing a figure in page coordinates, as in the LaTeX picture environment, is conceptually WYSIWYG.

Except as required to size and place the finished product, and to align text (below), an ePiX figure refers exclusively to Cartesian coordinates. The use of logical coordinates makes the input file easier for a human to read, and enhances flexibility: software can render a figure according to user-specified criteria at compile time, changing the size, aspect ratio, or viewpoint, for example.

Incorporation of typography imposes an additional requirement on a figure's coordinate system. A text box is attached to a specific logical location in a figure. However, fonts do not (and usually should not) scale when the size of a figure changes. Consequently, a LaTeX box cannot always be placed using only its basepoint if the result is to compile attractively at various aspect ratios: the Cartesian location of the basepoint does not generally undergo the expected affine scaling when a figure is resized (Figure 3). ePiX handles this difficulty by positioning a

label "coarsely" using Cartesian coordinates, then offsetting it "finely" in true coordinates, namely, aligning the text box on a point other than its LaTeX basepoint. In other words, a scale-invariant alignment point is manually attached to each label, and Cartesian coordinates are used to position this alignment point. There seems to be no simple, high-quality alternative to aligning labels visually and individually.

3.4 Scene Representation

An `ePiX` input file describes a 3-dimensional *world*, which is represented on an abstract 2-dimensional *screen*. World and screen coordinates are Cartesian, and not directly related to the printed figure's size. The screen contains a *bounding box*, a user-specified Cartesian rectangle that is affinely mapped to a LaTeX `picture`. The overall size of the figure is given directly in the input file, while the aspect ratio is determined by the relative aspect ratios of the bounding box and the picture box.

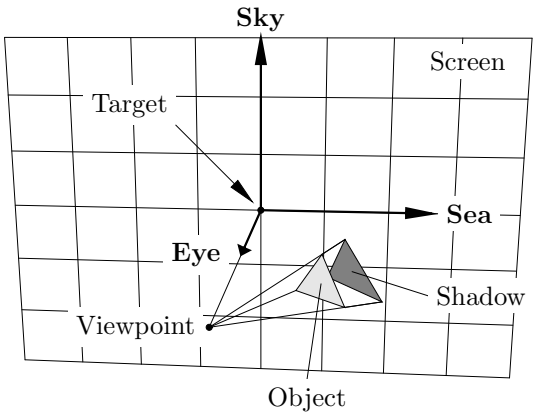


Figure 4: `ePiX`'s point-projection camera model.

The *camera*, consisting of a *body* and a *lens*, maps the world to the screen; indeed, the screen should be regarded as the camera's film plane. The camera body contains information about the location and spatial orientation of an abstract observer, while the lens is the actual mapping, point projection by default (Figure 4). The camera is designed to behave like a real camera: The viewpoint and target may be set arbitrarily, the camera rotated about its axes (sea, sky, and eye), and the lens changed.

To control the abstract and/or printed size of a figure, `ePiX` can remove figure elements that lie outside a user-specified "clip box" (Figure 1), and can "crop" a figure by masking elements that lie outside the bounding box (Figures 3 and 5). Clipping and cropping are disabled by default, in accordance with the design philosophy of imposing minimal default behavior.

3.5 Layering and Hiding

The `eepic` file produced by `ePiX` is at some stage converted to PostScript or PDF. In either case, the output is layered: objects occlude earlier parts of the file. For 2-dimensional black and white line drawings, layering is a minor concern, but for shaded, color, or 3-dimensional pictures, layering is usually important.

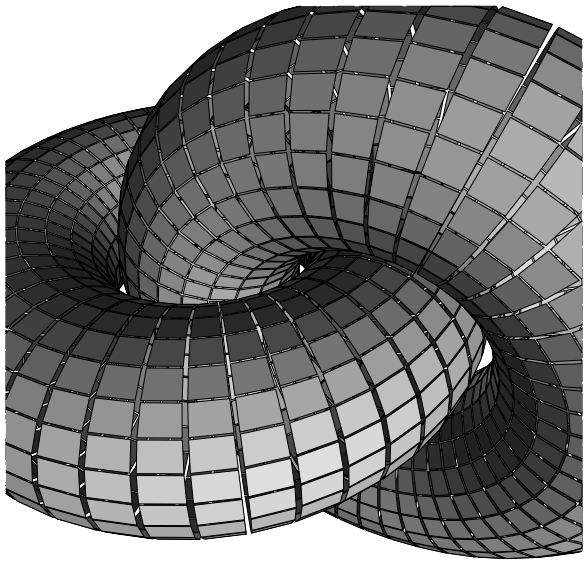


Figure 5: Layering, shading, and cropping.

`ePiX` does not currently automate hidden object removal, but manual techniques provide satisfactory results. In Figures 1 and 5, paths and surfaces are broken into mesh elements, sorted by distance to the viewpoint, and printed to the file in decreasing order of distance. The shading in these figures exemplifies the use of programming constructs in `ePiX`. For each mesh element, a normal vector and illumination vector are calculated, and the shade of gray is a simple function of the angle between these vectors. Similar techniques can be used to simulate multiple light sources, even light sources of varying colors.

3.6 Implementation

Befitting its role as a bridge between the computational power of C++ and the high-quality typography of LaTeX, `ePiX` is not a stand-alone program, but is instead assembled from standard components: the C++ compiler, libraries and binutils; GNU `bash`;

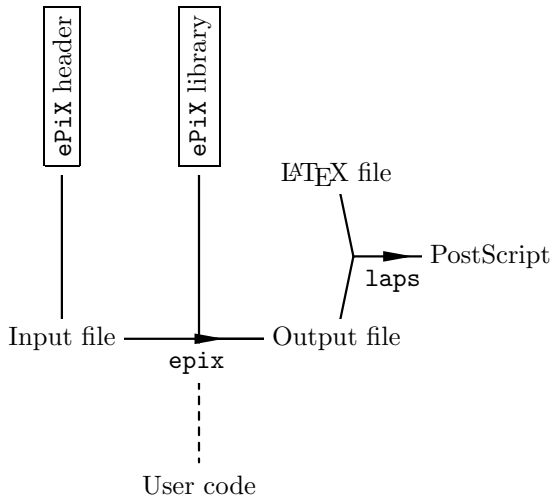


Figure 6: Processing an input file.

and optionally LaTeX, Ghostscript, and ImageMagick. The bulk of ePiX proper consists of a compiled C++ library and header file.

An input file is a short program that incorporates functions from the ePiX library. The shell script epix invokes the compiler on the input file. The resulting binary executable writes the LaTeX code of the figure, which the script directs to a file (Figure 6). Each of ePiX's scripts accepts numerous command-line options, which are listed by running "<script> --help".

From its inception, ePiX has used an external compiler to read and parse input files. This requirement, which may at first seem limiting, is not essentially different from reliance on an interpreter, be it Java, METAPOST, Perl, PostScript, Python, or TeX itself. Further, there are at least three practical reasons for utilizing the C++ compiler.

First, any program processing user-supplied input must recognize and cope with both well-formed and malformed data. The use of an existing compiler avoids both the substantial complication and needless duplication of effort that would result from coding a compiler or interpreter from scratch.

Second, separately compiled code can be incorporated in an ePiX figure with a command-line switch. Use of a widely-spoken language allows users to extend ePiX easily.

Third, when a typical plot is generated, a few functions are called repeatedly, possibly thousands of times. Compiled code runs quickly enough (compared to interpreted code) to justify the time overhead of compiling code to process a figure. When the plot depicts the outcome of a complicated algorithm (such as solving a differential equation), the extra efficiency of compiled code can be substantial.

4 Future Development

Until now, ePiX has existed as a single-developer project, and has grown primarily along lines dictated by a need for features. The current source tree is nearing an evolutionary *cul-de-sac*, and future work will focus on a redesigned and re-implemented version, known informally as The Next Generation. The author welcomes user feedback, design suggestions, and additional coders. The source tree is on the CVS server at savannah.gnu.org.

The Next Generation will separate input, representation, and output, serving as a general-purpose scene description and rendering utility rather than merely a LaTeX-specific image creation tool. However, incorporation of high-quality typography will remain a primary goal. Additional aims of TNG include providing flexible page markup, allowing multiple scenes to be placed in a single figure; more modularized output, so that a single input file can generate a sequence of output files — in various formats — from a single run; and better support for object hiding in 3-dimensional figures.

A framework for high quality scientific drawing and data visualization is of wide interest to the mathematical, scientific, and typesetting communities. It is hoped that ePiX will contribute toward the realization of a GPL-ed utility that is efficient, intuitive, computationally powerful, and sufficiently flexible to grow with its user base for the long-term future.

◇ Andrew D. Hwang
Department of Math and CS
College of the Holy Cross
Worcester, MA 01610-2395, USA
ahwang@mathcs.holycross.edu
http://mathcs.holycross.edu/~ahwang/

LaTeX in 3D: OpenDX annotations

J. P. Hagon

Abstract

We present a system, *DXfontutils*, for adding high-quality annotation to OpenDX objects using LaTeX as the typesetting engine. The system utilizes native OpenDX fonts converted from original outlines (TrueType, OpenType or PostScript) using the author's font2dx translator. Also we demonstrate how OpenDX can be used as a tool for producing special effects with OpenDX text elements which have been typeset by LaTeX.

1 Introduction

OpenDX [2] is a general purpose data visualization system similar to Khoros, IDL, AVS, Amira and others. As its name implies it is open source software and freely available. It was formerly a product from IBM known as Data Explorer. IBM released the Data Explorer source code for public use under a special licence in 1999.

OpenDX has an extremely versatile data model and an excellent visual programming interface. Figure 1 shows the output of a simple example. This output was produced with the visual program illustrated in figure 2. The program consists of an *Import* module which reads in the data, and an *Image* module which displays the data.

The modules contain input and output tags. In this case, the output tab from *Import* is connected to the input tab of *Image*. The connection is made simply by clicking and dragging with a mouse. Clicking on the *Import* icon produces an entry box in which the name of the import file is typed. The appropriate tab then appears in the closed form shown in figure 2 within the *visual programming editor* (VPE) indicating that the parameter has explicitly been set. In fact, I/O tabs can be hidden to simplify the layout — *Image* has many more input tabs than the one shown here. Note also that there can be more than one output tab — the three output tabs from *Image* provide information about the rendered object, the viewing camera and the viewing position. Here is the program:

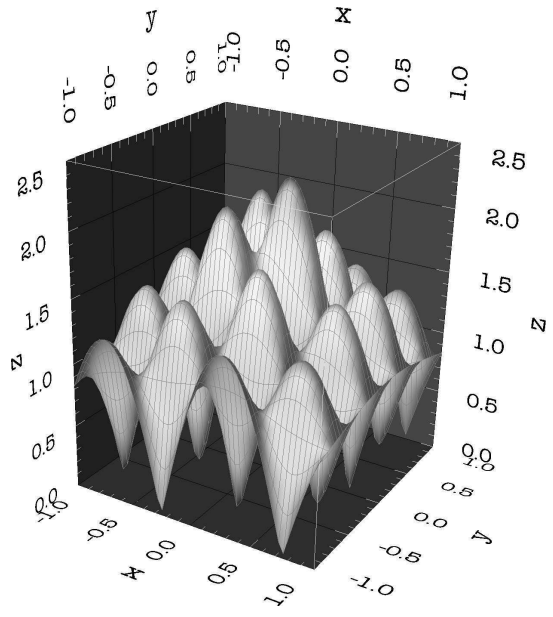


Figure 1: A simple 2D function plot in OpenDX.

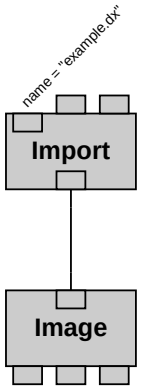


Figure 2: The OpenDX visual program which produced figure 1.

The VPE can be used to build large-scale interactive GUIs for specialized data analysis. Furthermore, the user can write custom *modules* (plugins usually written in C) and *macros* (a visual program combining other modules and macros).

The writing of modules is facilitated by a *Module Builder* interface and all of the standard OpenDX modules are available via a set of C libraries for skilled programmers. In fact, it is possible to produce an application using an external GUI library combined with the OpenDX graphics and rendering libraries. OpenDX can even be run in *script mode* using its own scripting language. Visual programs created through the VPE are stored in this scripting language.

Although OpenDX is a rather intimidating piece of software, there is extensive documentation, active user forums, commercial third party support and an introductory, tutorial based book [9]. Many useful third party macro and module libraries are available [2] for fields as wide-ranging as geophysics, medical imaging, quantum chemistry, biology, astronomy, social science, finance and engineering.

2 OpenDX Font Format

Two types of font are supported by OpenDX — 'line' fonts and 'area' fonts. The former are similar to the fonts that were common on pen-plotter output devices some years ago. Such fonts are still useful for screen display where hard copy quality is not important since they can be rendered very quickly. They are not our concern here and will not be discussed further. 'Area' fonts rely on filled polygons and are therefore capable of much higher quality than line fonts. Unfortunately there is just one such font supplied with the standard OpenDX release — the *Pitman* monospaced font.

2.1 Area Font Structure

Most outline fonts are fairly simple in concept — inner and outer boundary lines (often defined in terms of cubic splines) define an area to be filled. The spline defining the inner outline is opposite in direction (clockwise/anti-clockwise) to a spline defining an outer boundary. PostScript and TrueType fonts have opposite conventions in this regard.

Things are more complicated with an OpenDX area font. First, polygons rather than splines are used to define the outlines. Second, areas to be filled are not defined with clockwise/anti-clockwise polygons; instead, the required area must be *triangulated* to create an *area mesh*. These concepts are illustrated in figure 3.

In an OpenDX font file, the boundary polygons are defined through a set of positions and the connections defining the triangulated mesh are defined as a set of integer triples, each integer referring to a particular position. For example, a simple hyphen (essentially just a rectangle) might be defined in an OpenDX font file as shown in figure 4.

OpenDX fonts have exactly 256 entries, making them equivalent to *8-bit* fonts commonly used today. There is no flexibility in the format to allow for larger (or smaller) fonts. The files themselves adhere to the OpenDX data model and can be in text or binary format. The binary format is generally more compact. The official description of the font format can be found in the *OpenDX User's Guide* [1].

3 The Font Conversion Method

In order to get from, say, a Type 1 PostScript outline to an OpenDX font in the form illustrated in figure 4 requires roughly the following steps:

1. Obtain the boundary points corresponding to all inner and outer lines for each character in a font.

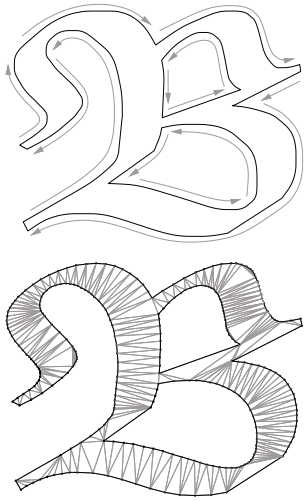


Figure 3: Comparing methodologies for PostScript/TrueType and OpenDX fonts. The letter $\mathfrak{B}$ of the AMS Euler Bold Fraktur font as represented in PostScript form (upper figure) and OpenDX form (lower). In the upper diagram there are three boundaries defined, the outer one going clockwise and the two inner ones going anti-clockwise. The lower diagram shows how a filled area is represented in a OpenDX font using a triangulated mesh bounded by the same outlines as in the upper diagram.

```
object "positions_hyphen" class array type
float rank 1 shape 3 items 4 data follows
 0.276  0.187 0.0
 0.011  0.187 0.0
 0.011  0.245 0.0
 0.276  0.245 0.0

attribute "dep" string "positions"
#
object "connections_hyphen" class array type
int rank 1 shape 3 items 2 data follows
2 1 0
0 3 2

attribute "ref" string "positions"
attribute "element type" string "triangles"
attribute "dep" string "connections"
#
object "hyphen" class field
component "positions" value "positions_hyphen"
component "connections" value "connections_hyphen"
attribute "name" string "hyphen"
attribute "char width" number 0.333
attribute "series position" number 45.000000
```

Figure 4: An entry for the hyphen character from a native OpenDX font file. Note the '`char width`' and '`series position`' attributes.

2. Triangulate the appropriate regions and obtain a set of connections for each character.
3. Output positions, connections and width information for each character in OpenDX font format.

To perform this task, we make use of three software packages, all of which are freely available. The packages are fontforge [10], pstoedit [6] and Triangle [8]. A brief description of each package follows, along with an explanation of its contribution to the OpenDX font conversion process.

3.1 fontforge

This remarkable application, by George Williams, is an outline font editor capable of creating and editing both PostScript and TrueType fonts. It is similar to commercial font editors such as Fontlab or Fontographer and provides much of the same functionality. It is available for multiple platforms and can be compiled from source if required. Further details may be obtained from the fontforge web site [10].

For all its many features, only limited use is made of fontforge in the OpenDX font production procedure. In particular it is used to obtain the following vital font information:

- The official name of the font.
- The name, ASCII code and widths of each font character. This is stored temporarily in one file for each font.
- An Encapsulated PostScript (EPS) rendering of each character in the font for subsequent processing by pstoedit.

The above procedure can be automated via fontforge's own scripting language.

3.2 pstoedit

Written by Wolfgang Glunz, this is a well-established and very useful package which converts PostScript (and PDF) files into a variety of vector formats.

pstoedit is used to extract the boundary point information for each character by converting the eps files generated by fontforge into gnuplot [4] commands. The gnuplot driver was chosen because its output is in a very convenient form for subsequent processing — the boundary points being returned as a column of (x, y) pairs. When a full closed curve is completed, this is indicated by a blank line and the next set of points started, if there is more than a single closed curve for the given character. The generated output file can then be loaded into gnuplot and viewed via the gnuplot command `plot <file>` or alternatively `plot <file> with lines` if you want to see the points joined up. Some gnuplot output for the Euler Fraktur character discussed earlier is shown in figure 5. The remaining task is to add the connection information.

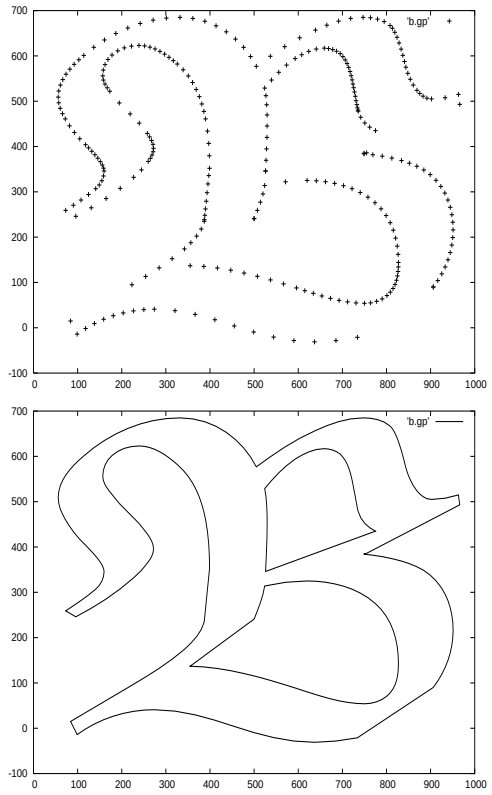


Figure 5: gnuplot rendering of the Euler Fraktur $\mathfrak{B}$. The upper diagram shows a points plot, the lower shows a line plot (each point is connected to the next point (as ordered in the input file created by pstoedit).

3.3 Triangle

This program is the work of Jonathan Shewchuk. It produces a triangulated mesh, given a set of input points and constrained segments — i.e. the boundary outlines of each font character. Triangle is a very efficient program and makes the task of triangulation relatively straightforward. The input required is a simple text file (referred to as a `.poly` file — see figure 6) with entries supplied for:

- A list of vertices — these are the nodes which form the boundary outlines for each character. They take the form of (x, y) pairs.
- A list of segments, i.e. the connection information needed to construct the boundary polygon. These are a list of integer pairs corresponding to

```
# Vertices, dimension, attributes, boundary markers
#
286 2 0 0
#
# Vertex no., x, y
#
0 0.906 0.09
1 0.734 -0.021
      .
      .
284 0.528219 0.394703
285 0.527145 0.369736
#
# Segments, boundary markers
#
286 0 0
#
0 0 1
1 1 2
2 2 3
  .
  .
192 192 193
193 193 0
194 194 195
  .
  .
243 243 244
244 244 194
245 245 246
  .
  .
284 284 285
285 285 245
#
# Holes
#
2
#
0 0.640961 0.323633
1 0.6685625 0.6155
```

Figure 6: A Triangle '`.poly`' file showing how vertices, segments and holes are set up. Note the termination segments which close each polyline and the two hole coordinates.

the vertices mentioned previously. Since all the vertices are correctly ordered, this list can be generated easily; and since all polygons are of the simple closed form, the last entry for a given polyline will be of the form $(n+m-1, n)$ where n is the starting vertex and m is the number of points in a given closed polyline.

- A list of 'holes', if any. These are points which lie within regions inside certain polylines which are not to be triangulated. In the case of the Fraktur $\mathfrak{B}$, it is clear that there are two interior polygons which enclose regions which are not to be triangulated. By specifying a hole point anywhere in a given region, Triangle is instructed not to triangulate that region.

Triangle produces a set of triangulated elements connecting polyline vertices from this input and stores the elements in a `.ele` file.

Vertices and segments are essentially provided by pstoedit but holes need to be calculated explicitly. As mentioned previously, the sense of a polygon (clockwise or anti-clockwise) determines if it should be filled or not. If it is not to be filled, then a hole coordinate must be placed somewhere within the polygon.

In the Type 1 PostScript format an anti-clockwise polygon is one forming an inner boundary and therefore containing a hole; for TrueType it's the other way round. A simple algorithm exists [5] for determining if a polygon is clockwise. For a closed polygon with n points $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$, calculate the quantity:

$$\mathcal{A} = \frac{1}{2}\sum_{i=0}^{n-1}(x_i y_{i+1} - x_{i+1} y_i) \text{ with } (x_n, y_n) \equiv (x_0, y_0)$$

If $\mathcal{A} > 0$ then the polygon is anti-clockwise, otherwise it is clockwise. Once a hole polygon has been identified, any point within it serves as a hole point for Triangle.

The following algorithm is used [3] to construct an interior polygon point:

1. Identify a convex vertex v.
2. For each other vertex q do:
   (a) If q is inside avb, where a and b are the adjacent vertices to q, compute distance to v (orthogonal to ab).
   (b) Save point q if distance d is a new minimum.
3. If no point is inside, return midpoint of ab, or centroid of avb.
4. Else if some point inside, qv is internal: return its midpoint.

Application of this algorithm usually results in a hole point being set very close to a boundary segment — so close, that to the naked eye the point often seems to lie *on* the segment.

4 Putting it all together

Two Perl scripts — g2poly and font2dx — have been written to automate the above procedure. g2poly converts a gnuplot input file (generated by pstoedit) into a `.poly` file suitable for input into Triangle. Everything else is handled within the font2dx script, which in fact calls g2poly. font2dx can optionally reencode a font according to several common encoding schemes. At present font2dx outputs fonts only in

text (ASCII) format, rather than the more compact binary format.

The general form of a font2dx command is:

```
font2dx [OPTION]... FILENAME
```

where `FILENAME` is a PostScript Type 1, OpenType or TrueType font file. At present, the following options are available:

`--noclean` Don't clean up intermediate files (usually there are *hundreds* of these!) — by default these files are deleted, leaving just the generated and original fonts.

`--scale=<integer>` Attempt rescaling of the font. This can be used to correctly scale a font in cases where the default scale factor fails.

`--negate` Reverse the normal convention for inner and outer closed polygons.

`--flat=<number>` Set the pstoedit 'flat' parameter. This defaults to 1.0 and the acceptable range of values is [0.2–100.0]. This parameter controls how accurately curves in fonts are approximated by polylines — higher numbers give rougher approximations.

`--enc=<encoding>` Change font encoding. There is a choice of many pre-defined schemes and if the encoding is not one of these, then an encoding file is looked for, with the assumed name `<encoding>.enc`. Hence, many of the standard encoding schemes in TEX can also be used.

`--help` Print usage information and help.

font2dx will process only the first 256 character glyphs in a font. Modern fonts often have many more than this. If there is a 'hidden' glyph not in one of the first 256 slots, then you could try manually re-encoding the font with a tool such as fontforge prior to running font2dx.

A further issue is that OpenDX font characters have a width attribute, but no explicitly defined height. However, TEX is perfectly happy using characters which have *zero* width. In such cases, it is impossible to correctly scale such characters unless there is a corresponding height (so that scaling ratios can be calculated). font2dx therefore adds a `char height` attribute, equivalent to $\langle height \rangle + \langle depth \rangle$ enabling proper scaling even for zero width characters.

4.1 Quality Issues

It can be worth experimenting with the `--flat` option to optimize font quality. The default value for this parameter is 1 which generally produces very good quality fonts, i.e. unless the fonts are greatly enlarged, it is almost impossible to detect the polygonal character of the outlines. In fact, a value of 10 produces pretty decent results for most text fonts we have tested. Figure 7 illustrates the effect of the `--flat` parameter for the URW Times-Roman font. For exceptionally fine and detailed fonts a flat parameter of less than 1 may prove necessary.

Figure 7: OpenDX rendering of URW Times-Roman for different flat parameters: flat = 100 (top); flat = 10 (middle); flat = 1 (bottom). Even a flat value of 100 produces recognizable text albeit in effectively a different font!

There is a trade-off between font size and quality with smaller flat parameters leading to larger file sizes, as might be expected. However it is generally true that as flat parameters get very large, the space saved is not worth the enormous loss in quality. This is illustrated in Table 1 where it is clear that there is not much space to be gained in going from a flat parameter of 10 to a flat parameter of 100 in the case of URW Times-Roman — but there is an enormous loss of quality. In the rest of this paper, we use fonts generated with a flat parameter of 1.

'flat' parameter	100	10	1
URW Times	181717	254580	543254
WebOMints-GD	417797	709443	2036248

Table 1: Font file sizes (in bytes) for different flat parameters in the case of URW Times and the ornament font WebOMints-GD.

5 Annotation in OpenDX

This ability to create native OpenDX fonts from industry-standard outlines, as described above, has the potential to greatly improve annotation quality within OpenDX. It has been common for users to post-process their OpenDX-generated images with graphical editing tools such as Gimp or Photoshop in order to add text elements. Either that, or the *Pitman* font was grossly overused (because it was the only good quality font available) making many annotated images produced by OpenDX immediately

identifiable.[1] One remaining issue is the typographical quality of OpenDX annotation, particularly with regard to mathematics. This is one area where TEX can certainly help!

5.1 OpenDX *Text* and *Caption* Modules

Text within OpenDX is treated just like any other OpenDX object. It can be scaled, rotated, coloured, and manipulated in many different ways. There are two modules within the core OpenDX system which facilitate text entry and annotation.

The *Text* module allows text to be positioned anywhere in 3D space with any rotation, size and orientation. The position and height are given in world (user) coordinates.

The *Caption* module displays a caption on the screen independently of any other OpenDX objects representing the user's data. This produces text which remains in the same position relative to the screen. The position of the text is given in screen (viewport) coordinates, i.e. a position of $[0.9, 0.5]$ means 9/10 of the way along the horizontal axis and half way up the vertical axis. The height is given in pixels.

Text and *Caption* have very rudimentary typesetting capabilities. Escape sequences (using 'backslash' as the escape character) can be used to obtain characters not available on some keyboards (e.g. diacriticals) and spacing is achieved via the 'space' character (ASCII 32) of the particular font in use. Now this latter point raises a problem if one wishes to use, say, Computer Modern Roman because this font doesn't have a space character! Position 32 is taken up by the *suppress* character — -. Of course, this isn't a problem in TEX since all spacing is calculated internally — removing the need for an explicit 'space' character.

5.2 The *LaTeXText* and *LaTeXCaption* Macros

Two new OpenDX macros were developed as alternatives to the *Text* and *Caption* modules: *LaTeXText* and *LaTeXCaption*. In OpenDX a macro is a combination of modules (and other macros) and can be created through the OpenDX visual programming editor.

The above macros take LATEX commands as their main argument. The user may enter, optionally, a set of LATEX *preamble* commands if certain packages are required. Hence,

```
\\usepackage{cmbright}
```

might be entered as a preamble option if the CM Bright fonts were required. The double backslash is not an error — it's an unfortunate consequence of the previously mentioned fact that backslash itself is treated as an escape character in text arguments of the *Text* and *Caption* modules.

If there is a lot of text to be typeset, it is more convenient to supply a file containing the text rather than type the text in as an argument to a macro. For this reason, two modified versions of *LaTeXText* and *LaTeXCaption* are available, which accept a file of LATEX commands rather than a string of commands. These macros are *LaTeXFileText* and *LaTeXFileCaption* respectively. Another advantage of using these modules, in addition to their primary purpose, is that backslash characters do not need to be doubled-up. Within the VPE, the macros appear like this:

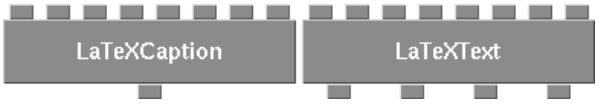


Figure 8: *LaTeXText* and *LaTeXCaption* macros as they appear in the OpenDX VPE.

LaTeXText takes the following inputs:

`latex_string` A string of LATEX commands.

`height` Height of text in user (world) coordinates.

`position` Position vector of reference point (see below) in user coordinates.

`baseline` Direction of baseline expressed as a vector.

`angle` Euler-type angle specifying rotation about the baseline axis.

`preamble` A string of LATEX preamble commands — for example, to load font definitions or special packages.

`extrusion` A scalar defining the extrusion in user coordinates. A number ≤ 0 produces no extrusion.

`reference` An integer (1–9) specifying the reference point on the formatted text object to be used for positioning. (1) refers to bottom left (the default); (2) is bottom centre; (3) is bottom right, etc., up to (9) which refers to top right.

There are 4 outputs:

`text` Complete object including extrusions and surfaces.

`nosurface` Just the extrusion (no upper or lower surfaces).

[1] Not unlike the situation some years ago where almost any document produced using TEX used the Computer Modern fonts — not because TEX was incapable of using other fonts but because at that time it was not straightforward to do so.

`top_surface` The top surface.

`bottom_surface` The bottom surface.

The four outputs allow the upper/lower surfaces and extrusion to be handled differently. For example, the upper and lower surfaces can be given different colours.

LaTeXCaption has just a single output and the following inputs:

`latex_string` A string of LaTeX commands.

`coords` An integer specifying the type of coordinates used: (1) viewport, (2) pixel, (3) world or (4) stationary. Using stationary coordinates, the text string will be attached to a particular point in world coordinates but will retain the same orientation with respect to the viewing camera.

`direction` Direction of baseline expressed as a vector.

`priority` An integer specifying how the text is layered relative to the other OpenDX objects: (−1) behind, (0) equal or (1) in front.

`position` The screen position. How this vector is interpreted depends on the value of the `coords` parameter.

`height` Height, in pixels unless stationary position, in which case world coordinates are used.

`preamble` A string of LaTeX preamble commands.

`reference` An integer (1–9) specifying the reference point on the formatted text object to be used for positioning. See description above for *LaTeXText*

The conversion of LaTeX commands to OpenDX objects is handled by two Perl scripts — dvidx and latex2dx:

dvidx is a TeX `dvi` driver program similar to dvips et al. It takes a `dvi` file as input and generates an OpenDX object. This object contains the correctly scaled and positioned characters from the OpenDX fonts converted from outline originals. It understands dvips colour specials and can output in two different OpenDX formats: a *compact* form which consists of external references to OpenDX fonts; and an *inclusive* format in which all the relevant data from the external font files is included in the output. dvidx can be used standalone to produce OpenDX output if desired. Multiple pages are handled by collecting individual pages in an OpenDX *Group* object.

latex2dx is essentially a wrapper Perl script around dvidx. It takes raw LaTeX input, produces a temporary `dvi` file and then calls dvidx to generate OpenDX output.

It is latex2dx that is actually called by the *LaTeXText* and *LaTeXCaption* macros but it is dvidx which does all the hard work.

6 The dvidx Perl Script

The writing of dvidx was made considerably easier by the use of two clever Perl packages written by Jan Pazdziora — `Font::TFM` and `TeX::DVI::Parse` [7]. The working of dvidx is roughly as follows:

1. First, run dvicopy[2] on the original `dvi` input to translate all the virtual font references to base fonts.
2. Parse the dvicopy output using the Perl package `TeX::DVI::Parse`.
3. For each font encountered, obtain the appropriate metrics from the TeX font metric (`tfm`) file using `Font::TFM`.
4. Map the base font to a raw OpenDX font and extract the appropriate characters.
5. Position and scale the character via an OpenDX rotation/translation operation (in OpenDX jargon, this is an *XForm* transformation object).

dvidx cannot read the packed font (PK) files traditionally used by `dvi` drivers and usually created (indirectly) from METAFONT source files. There is no reason in principle why it could not be made to use such files but the approach described here produces higher quality and is much easier to implement. However, it does mean that METAFONT sources which have not been converted to outline form cannot currently be rendered using dvidx.

6.1 The dvidx Map File

The mapping of raw TeX font names to OpenDX font files is done via a map file similar to (but much less versatile than) the map file used by dvips. The dvidx map file contains two columns, the first column giving the name of the raw TeX font and the second column giving the name of the corresponding OpenDX font file. It is possible that a single OpenDX font file may map to more than one raw TeX font but not vice-versa. If a raw TeX font maps to more than one OpenDX font file then the *last* entry in the map file is the one that is used.

One of the features of the dvips map file is that one can re-encode a PostScript font on the fly via a re-encoding directive within the map file itself.

[2] dvicopy is a program which is routinely available as part of all modern TeX implementations. Its primary purpose is to expand virtual font definitions. This is useful in cases where a `dvi` driver doesn't understand virtual fonts.

For example, a re-encoding to TeXBase1 is achieved within a dvips map file with the following directive:

```
" TeXBase1Encoding ReEncodeFont " <8r.enc
```

where 8r.enc is a file containing the appropriate PostScript encoding commands. This type of functionality could be added to the dvidx map file, but in many cases would be redundant. This is because OpenDX fonts always contain *exactly* 256 characters whereas Type 1 PostScript fonts generally contain 'hidden' glyphs that are not contained within the visible 256 character slots. It is often the case that the purpose of re-encoding is actually to place many of these hidden glyphs in visible slots.

Our solution to this problem is somewhat brute-force but effective. A program such as fontforge can be used to re-encode the original outline font using the required encoding file (such as 8r.enc, for example). The re-encoded PostScript font is then converted to an OpenDX font using font2dx but given a different name to the original. The convention we use, is to append the string -<enc> to the OpenDX file name. This produces map file entries like:

```
ptmri8r          Times-Italic-8r.dx
tii              Times-Italic-8y.dx
```

whereas in the dvips map file we would have:

```
ptmri8r Times-Italic
 " TeXBase1Encoding ReEncodeFont " <8r.enc
tii Times-Italic
 " TeXnANSIEncoding ReEncodeFont " <texnansi.enc
```

A similar brute-force approach can be used to deal with 'slanted' fonts created on the fly via a dvips map file entry such as:

```
ptmro8r Times-Roman " .167 SlantFont ...
```

7 Examples of Text Annotation in OpenDX

Suppose we wish to add a title to the image shown in figure 1. The normal way to do this in OpenDX would be via the *Caption* module. The visual program would look like that shown in figure 9. We have created a caption object and added it to the original object (using the *Collect* module). The result is shown in figure 10.

Now the function we are plotting is quite a complicated one:

$$z = \frac{\sqrt{\sin(\omega x)\cos(2\omega y) + 1}}{1 + \sqrt{x^2 + y^2}}$$

and we have tried to indicate the form of the function in the caption. The core facilities of OpenDX limit what we can do here and the result is both difficult to read and cumbersome to position because it consists of one relatively long line of monospaced text.

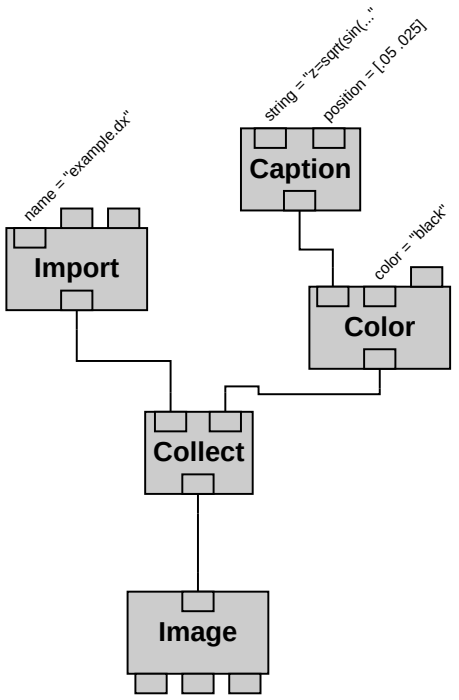


Figure 9: Modified visual program using *Caption*.

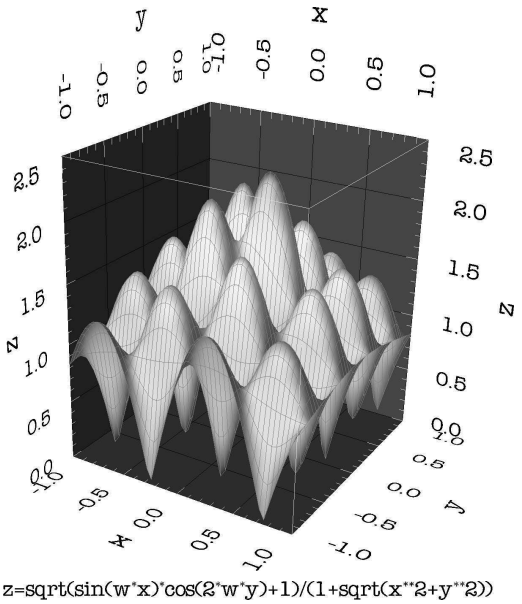


Figure 10: OpenDX-annotated figure.

So, we instead use the *LaTeXCaption* macro. In addition, to make the mathematics easier to read on-screen, we anti-alias the output using the *TextAlias* macro. The resulting visual program is shown in figure 11. The main argument to *LaTeXCaption* is the following piece of LaTeX code:

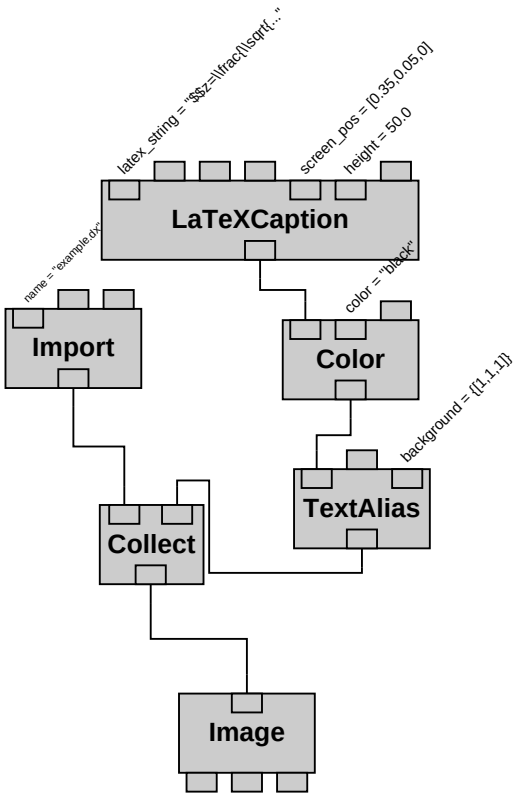


Figure 11: Modified visual program using *LaTeXCaption*.

```
$$z=\\frac{\\sqrt{\\sin(\\omega x)
   \\cos(2\\omega y)+1}}{1+\\sqrt{x^2+y^2}}$$
```

where, as mentioned earlier, the backslashes have been doubled because backslash is an escape character in OpenDX. *LaTeXCaption*'s other arguments include an orientation vector, a position vector and optional LaTeX preamble text. Standard OpenDX modules can be used to make other modifications, such as colour changes.

8 Special Effects with *LaTeXText*

LaTeXCaption provides flat 2D screen annotation within a 3D OpenDX space. *LaTeXText* has similar functionality except that it operates in 3D and the text can be oriented and positioned arbitrarily in 3D space. In this section we show how OpenDX can be used as a tool to produce 3D special effects. All the examples which follow were typeset with LaTeX via the *LaTeXText* macro (or its equivalent *LaTeXFileText*) but the effects described can all be applied to standard OpenDX text objects no matter how they were created. These examples really just scratch the surface of what can be done with special effects in OpenDX — the possibilities are almost limitless.

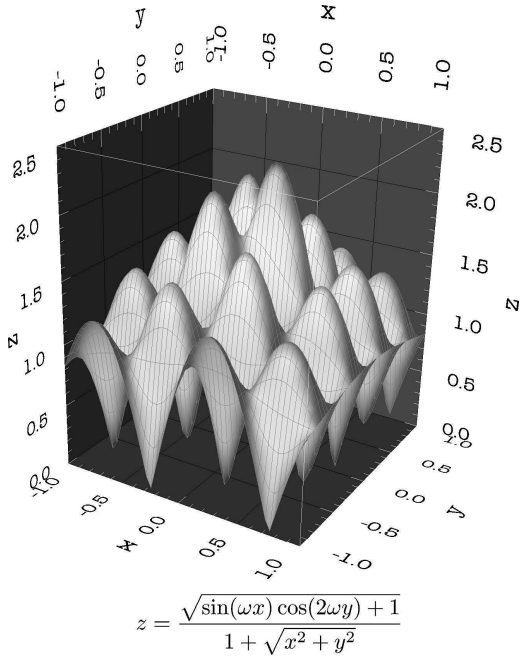


Figure 12: LaTeX-annotated figure.

8.1 Warped Text

Many of the operations that can be done on OpenDX objects can also be done on text. So, for example, we can *warp* a piece of text by transforming its coordinates as shown in figure 13. Note that the warped text has a distinct 'sheen' due to reflected light. The properties of the lighting can be precisely controlled within OpenDX, as can the reflective properties of the surfaces of objects.

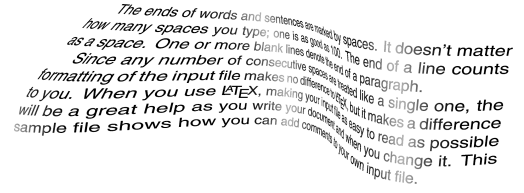


Figure 13: Warping text.

8.2 Texture Mapping

Texture mapping, i.e. the overlaying of a 2D image onto a 2D surface, is a common technique in computer graphics. It is most useful when the surface itself has a low resolution. Overlaying a high resolution image then produces an impressive visual effect but with an underlying simplicity allowing fast geometric transformations. In figure 14 we overlay an image (texture) onto the font characters. Note that as well as being texture mapped, the text has

also been *extruded* to give it a thickness. We show another example of extrusion later. At present, texture mapping in OpenDX relies on OpenGL hardware rendering and there are some technical limitations on the quality of hard-copy output one can obtain.

Figure 14: Texture mapping.

8.3 Text Boundaries

It is easy within OpenDX to embellish the character boundaries of text in many different ways. In figure 15, spherical glyphs are used to define boundaries but almost anything is possible — and often easy to set up. The density and size of the glyphs can be adjusted within OpenDX by first extracting the outline information for each character and populating the outlines with graphical glyphs.

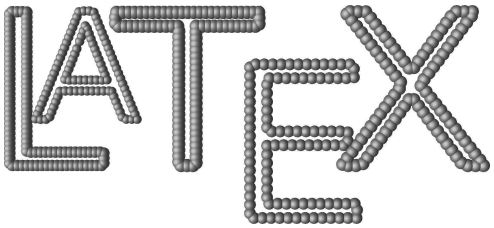


Figure 15: Glyph boundaries: spheres in this case.

8.4 Exploiting Transparency

The opacity of the 3D text can be changed to make it semi-transparent. For example, a stained glass effect can be achieved by wrapping a tube around the boundary of each character (to simulate the lead) and reducing the opacity of the text to some suitable value (< 1). This is illustrated in figure 16.

Figure 16: Stained glass effects — note the transparency of the 'glass'.

8.5 Extrusion

In figure 17, we illustrate *extruded* 3D text — i.e. the text has a thickness as well as width, height and depth!

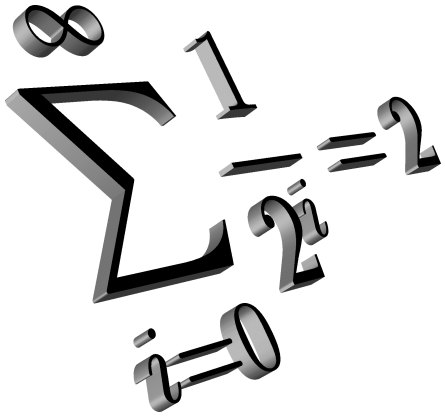


Figure 17: Extruded 3D text.

It should be emphasized at this point that as far as OpenDX is concerned, this is just an ordinary 3D object. In the image display window, you can rotate, zoom and even walk through the zero in the subscript on the $\sum_{i=0}^{\infty}$ sum! Such manipulations are possible with all the objects we have described. Note that in figure 17, the text has been rotated and the image generated with perspective. These properties can be controlled interactively from the OpenDX image window.

8.6 Foreign Languages

LaTeX has excellent support for many of the world's languages, including an increasing number of non-Latin languages such as Arabic, Japanese, Chinese and Hebrew, through packages such as ArabTeX, CJK and babel. Many of these packages have been

successfully applied to OpenDX captioning using *DXfontutils*.

9 Getting *DXfontutils*

The complete *DXfontutils* system consists of:

- three Perl scripts, font2dx, dvidx and latex2dx;
- five OpenDX macros: *TextAlias*, *LaTeXText*, *LaTeXCaption*, *LaTeXFileCaption* and *LaTeXFileText*;
- a set of TEX fonts in OpenDX format: Computer Modern, AMS Euler and a selection of others converted from outlines in the TEX Live 2004 distribution;
- several example OpenDX networks showing how to achieve the various effects described in this document and a sample dvidx map file.

The complete system can be downloaded from:

`http://www.njph.f2s.com/dxfontutils`

10 Summary

We have shown one way that LATEX can be used as a back-end typesetting engine: to enhance the limited annotation facilities provided in the core OpenDX system. To facilitate this, a font conversion program was written, allowing native OpenDX fonts to be utilized throughout for maximum efficiency. Additional requirements were an OpenDX `dvi` driver and a set of macros to be used in the visual programming editor of OpenDX.

We have also demonstrated how OpenDX can be used as a powerful tool for producing special effects on LATEX generated typeset material. LATEX and OpenDX are therefore complementary, each able to enhance the output from the other.

The main problem to date has been the speed with which the various Perl scripts, external programs, and OpenDX macros link together. For example, the standard OpenDX *Text* macro is much, much faster than the *LaTeXText* macro. The bottleneck is primarily the dvidx script.

OpenDX has a caching mechanism which means that for a given piece of LATEX formatted text, the dvidx and latex2dx scripts are run just once. Subsequent operations such as rotation or shading are done on the cached object. Of course, if the LATEX commands are changed, then the scripts are automatically re-executed. Generally, re-execution happens only if any of the inputs to *LaTeXText* or *LaTeXCaption* are changed.

Finally, there are several other improvements which could be made, such as:

- modifying font2dx to produce OpenDX fonts in the more compact binary format;
- modifying dvidx to read binary format OpenDX fonts (at present it works only with text format fonts);
- adding "SlantFont" transformation directives to the dvidx map file following a similar scheme to that used by dvips;
- adding `\special` support within dvidx for the inclusion of native OpenDX objects;

Currently, *DXfontutils* is more a proof-of-concept system than a well-tuned production software product. However, it is a proof-of-concept with considerable functionality. We have illustrated its use with LATEX, but there is no reason why plain TEX or other formats could not be used instead — all that would be required are minor edits to the latex2dx script.

References

[1] *IBM Visualization Data Explorer User's Reference*. `http://opendx.npaci.edu/docs/html/refguide.htm`, 1999–2004.

[2] *OpenDX*. `http://www.opendx.org`, 1999–2004.

[3] *comp.graphics.algorithms FAQ, §2.06*. `http://www.faqs.org/faqs/graphics/algorithms-faq`, 2004.

[4] *Gnuplot*. `http://www.gnuplot.info`, 2004.

[5] P. Bourke. *Determining whether or not a polygon (2D) has its vertices ordered clockwise or counter-clockwise*. `http://astronomy.swin.edu.au/~pbourke/geometry/clockwise`, 2004.

[6] W. Glunz. *pstoedit*. `http://www.pstoedit.net`, 2004.

[7] J. Pazdziora. `TEX::DVI::PARSE`, `Font::TFM`. `http://www.cpan.org`, 2004.

[8] J. Schewchuk. *Triangle*. `http://www.cs.cmu.edu/~quake/triangle.html`, 2004.

[9] D.L. Thompson, J.A. Braun, and R. Ford. *OpenDX: Paths to Visualization*. Visualization and Imagery Solutions Inc., 2001.

[10] G. Williams. *Fontforge*. `http://fontforge.sourceforge.net`, 2004.

◇ J. P. Hagon
Physics Centre
School of Natural Sciences
University of Newcastle upon Tyne
NE1 7RU
United Kingdom
`jerry.hagon@newcastle.ac.uk`

LaTeX

dramatist: Another package for typesetting drama with LaTeX

Massimiliano Dominici

Abstract

As the name plainly says, dramatist is a package designed to handle all the typographical specialities which arise in the edition of a dramatic work. It was originally designed to support a private edition of a mid-19th century Italian tragedy in verse: G. B. Niccolini's *Arnaldo da Brescia*. Being a package, it can be used with any class, from standard LaTeX classes to the more specialized ones, such as from the KOMA-script bundle, memoir (this is the actual class I've used the package with, for my work) and the like.

The package provides a general environment, drama, and specific commands for handling stage oriented document divisions (acts, scenes), characters lists and speakers' name appearance, stage directions. Both plays in prose and in verse are supported; for the latter, however, not in an explicit way, but relying instead on the verse environment and facilities provided by either the main class or packages like verse.

1 Introduction

The edition of a dramatic work needs specific support for several features, from special document divisions (acts and scenes, rather than chapters and sections) to special typographical treatment concerning the characters. This kind of support cannot be given by the standard LaTeX classes without redefining a large number of macros, a quite undesirable approach. Hence the need for a class or package expressly designed for this purpose, handling all the specialities involved in the job

The following packages dealing with stage plays and dramatic works are at present available on the CTAN archives and in most TeX distributions:

- plari (Kaijanaho, 2003): a small class that replaces, in the standard LaTeX report class, the set of usual document divisions with another one more suited for stage scripts, and adds some other minor features (the sides package is a new update of plari which we unfortunately did not have time to investigate);
- play (Kilfiger, 1999): coming with both a class covering most of the basic features of a stage play and a package designed for supporting insertion of small drama citations in the middle of a document of a different kind;
- drama (Swift, 1996): being part of the ambitious Frankenstein bundle, whose unusual philosophy it shares;
- dramatist (Dominici, 2003): the package to be introduced here.

The aforesaid packages are characterized by different approaches to the basic *class vs. package* decision and by different degrees of user configurability. It's a matter of opinion, and at last of taste, whether it is better to rely, for anything concerning layout adjustments, placement of floats, indexing, page numbering, appearance of headers and table of contents and so on, upon the facilities provided by a sophisticated class like memoir or scrbook, or to make use of the many packages which cover the same facilities. I found the first way easier, and that's why I wrote a package rather than a class.

From this point of view, a classification can be made which divides the four packages in two groups: on the one hand, plari and play come as a class, while on the other, drama and dramatist come as a package.

In the first group, the use of plari class can be recommended only for simple documents: the class lacks some basic features, such as a proper command for defining and typesetting a "Dramatis personæ" list, gives poor support for plays in verse and offers only a low degree of configurability. As it disables all the standard LaTeX sectioning commands, the class prevents the author or editor from inserting any material standing outside the stage play script itself, such as a preface, foreword, or introduction.

A more complete class is play. Except for the "Dramatis personæ" list, it covers all the major features relating to dramatic works, offers full internal support for plays in verse via a dedicated environment, and shows a reasonable degree of configurability. However, as the definition of the basic `\act` and `\scene` relies on the standard LaTeX macros `\chapter` and `\section`, the user cannot redefine single portions of these commands in a simple way: the entire macro must be redefined. Together with the class, a small package is also distributed, which provides a few basic features to be used for printing short citations from a play in the body of a document of a different kind.

As for drama, the argument is more complicated. drama is part of the Frankenstein bundle and shares its unusual philosophical lines. This sort of object-oriented interface to LaTeX, while remarkable from many points of view, makes difficult any attempt at customization by a user who is not en-

tirely familiar with it. However, it is quite usable as it comes, and it shows itself a powerful tool, covering almost all features (though no specific support is given for typesetting plays in verse), notably including a continuation message when a single speech of one character is broken across two pages and divided by a stage direction.

A closer examination of the dramatist package is the subject of the next section.

2 Overview of the dramatist package

2.1 The drama environment

The first task of a package designed for typesetting drama is to provide an environment which may work as a wrapper for the text to be formatted. In the case of dramatist, this is the drama environment.

The package provides two versions: the normal version, to be used when typesetting plays in prose, and a starred version for plays in verse. This is due to the different tasks to be performed in the two cases, mainly regarding text arrangement and speech tags' definitions. In the case of a play in prose, the dialogue is arranged in a description-like environment, where the item label is the speaking character's name. In the case of a play in verse, drama* calls the verse environment (provided by the main class used or by a package like verse) to arrange the text and simply prints the speech tag above the dialogue lines. If line numbering is allowed (see section 2.5 below), drama* also handles the features concerned with this.

In both cases, nothing, except for the dialogue and short directions within the dialogue, should be enclosed within the drama environment. Acts, scenes, stage directions and definitions of characters should be given *outside* the environment itself.

2.2 Document divisions

In a drama, the ordinary document division into chapters and sections is replaced by a division into acts and scenes. dramatist provides an interface for such a scheme with the commands `\act` and `\scene`. They start a new act or a new scene (the act on a new page, by default), respectively, take no arguments and by default print in small caps the name *Act* or *Scene*, followed by a roman numeral. An internal counter is charged to hold this numeral, increasing it every time `\act` or `\scene` is called. Like the standard `\chapter` and `\section` commands, `\act` and `\scene` have a starred form, which does not make an entry for the table of contents.

An optional argument can be specified; it is meant only for insertion of footnotes and endnotes, like this:

```
\act[\footnote{Content of the footnote}]
```

If a title for the act or scene is needed, the `\Act` or `\Scene` commands are available:

```
\Act[Short Title]{Title}
\Scene[Short Title]{Title}
```

dramatist provides no **oneact/multiact** switch (as the drama package does), which avoids printing the act number for one-act plays, because I think it unnecessary. To achieve such a result, the user need only redefine the single command `\printscenenum`, whose default is printing the act number, an endash, and the scene number, like this:

```
\renewcommand{\printscenenum}{%
          \scenenumfont \thescene}
```

As another example of customization, something other than the English word "Scene" may well be required. This is controlled by the command `\scenename`. To redefine it as the Italian word "Quadro", for example, this is all that is needed:

```
\renewcommand{\scenename}{Quadro}
```

2.3 Characters

Dealing with the characters of the play is another important task when typesetting drama. This involves many aspects and features of the document:

- Usually a "Dramatis Personæ" list is placed before any other material.
- One may want the name of the characters to appear in some particular typographical shape within the stage directions.
- Finally every part of the dialogue should be introduced by the name of the speaker printed in a standard recognizable form.

One can find support for all these features in the dramatist package.

The typographical appearance of the characters is defined once and for all by calling the command `\Character`. It takes up to three arguments: the first, optional, argument specifies what is to appear in the "Dramatis Personæ" list; the second the name of the character throughout the document, both in the stage directions and as a speaker; and the third provides the commands for calling the name specified by the second argument. In short, if ⟨*arg2*⟩ and ⟨*arg3*⟩ are respectively the second and the third argument, `\Character` creates a pair of commands `\`⟨*arg3*⟩ and `\`⟨*arg3*⟩`speaks` which can be used for printing ⟨*arg2*⟩ in a stage direction or as a speaker (see the examples below).

The "Dramatis Personæ" list is produced by the command `\DramPer`. It prints only those entries defined by `\Character` with the first, optional, argument specified. This can be useful to omit some

speakers from the "Dramatis Personæ" list, or when two or more individual characters act simultaneously as a single speaker (and thus, no related entry exists in the "Dramatis Personæ" list).

However, a `\speaker` command is also provided to deal with these occurrences: it takes one argument, the name of the character to print, but it does not define any command for printing the name in a stage direction.

Characters listed in the *Dramatis Personæ* may need to be grouped and given a common designation. For this occurrence the package provides an environment, CharacterGroup, taking a mandatory argument specifying the designation for the characters in the following group.

Inside this environment, the characters have to be defined by `\GCharacter`, whose syntax is the same of `\Character`, except that the first argument is here, obviously, mandatory. In the output, the characters will be grouped by a big parentheses with the common designation printed, centered, to the right. The user can define the amount of space reserved for the characters' names, the parentheses, and the designation by means of `\CharWidth`, `\ParenWidth` and `\GroupWidth`, respectively.

2.4 Stage directions and settings

Finally, the package provides support for printing stage settings and small indications in the body of the dialogue. The user can issue a `\StageDir` command (or the equivalent stagedir environment for longer stage directions) in the first case or a `\direct` command in the second case.

When working with a verse play, the `\direct` command also takes a starred form, to be used when the command itself occurs at the end of a stanza. This works only with the verse package and the verse environment provided by the memoir class.

2.5 Support for plays in verse and line numbering

When dealing with a play in verse, the author must take into consideration not only the specific features of a play but those of verse, too. Of course, the specialities he will deal with will be, generally speaking, different from those encountered when strictly typesetting verse. He will seldom make use of stanzas, for instance, while he will often break a verse line over several physical lines — every time a new character begins to speak in the middle of the verse line itself. Line numbering also might be required.

So, a package for typesetting drama should provide some kind of support for these features. But, in my opinion, there is no need for full internal support. I thought, indeed, it would be better to rely on the facilities provided by the many extant classes and packages, leaving it to the author to choose from among them the one he finds most suited to the work at hand.

dramatist, then, does not define a verse environment, but supposes it is already provided by the class (and very usually this supposition is true), or by loading a package such as verse or poemscol. Since dramatist does not deal directly with the specific features of verse, no conflict can arise with a specific verse environment,[1] although the author will of course be restricted to the facilities provided by the chosen environment.

As for line numbering, dramatist defines three options, provided the memoir class or the verse package has been loaded. The default is to number verse lines consecutively throughout the entire drama, with no regard to acts and scenes. The other two possibilities are to number verse lines per act or per scene; these can be specified by loading the package with the option lpna or lpns, respectively. If the required package or class has not been loaded, the option is simply ignored.

Of course, all this is meant only for the drama* environment. If the code defined by one of these options is called inside the drama environment (i.e., a passage in prose) a warning message will be written to the log file and the option ignored. Generally speaking, line numbering is not very useful for a play in prose; if this is needed, the lineno package may provide a workable solution.

2.6 User customization

Everything in the package has been made as customizable as possible by means of user definable commands. In particular, everything concerning the typographical appearance, such as spacing, fonts, and so on, can be adjusted to the user's taste by means of simple `\renewcommand` or `\setlength` commands.

For example, the default behaviour of the package is to print the act name and number in small caps. Suppose the author wants them printed boldface; also, he wants to insert a large amount of space before the start of the act. The following lines, inserted in the preamble, perform this task.

```
\renewcommand{\actnamefont}{\bfseries}
\renewcommand{\actnumfont}{\actnamefont}
\setlength{\beforeactskip}{50pt}
```

Or, if the author wants, as for a play in verse, the character name to have a negative indentation:

```
\setlength{\speaksskip}{-1em}
```

[1] There is one exception to this statement; see section 2.4.

Moreover, the user may store his preferred settings in a `dramatist.cfg` file, placed either in the working directory or any directory where TEX looks for style files, and reuse them in future documents.

3 Examples

Figure 1 shows the output for a play in prose, from Schiller's *The Robbers*; figure 2 shows the input.

Another example is given for a play in verse, taken from the tragedy *Arnaldo da Brescia*. Figure 5 shows the input, while figure 3 shows the output for "Dramatis Personæ" list, and figure 4 the first page of the first act.

References

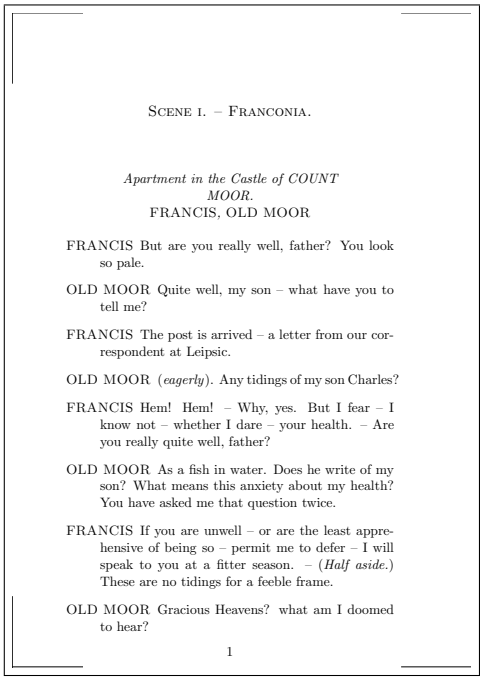
Dominici, Massimiliano. "`dramatist.sty`". Available from CTAN, `macros/latex/contrib/dramatist`, 2003.

Kaijanaho, Antti-Juhani. "The plari document class. Typesetting stageplay scripts with LaTeX 2ε". Available from CTAN, `macros/latex/contrib/plari`, 2003.

Kilfiger, James. "Typesetting drama with LaTeX". Available from CTAN, `macros/latex/contrib/play`, 1999.

Swift, Matt. "The drama LaTeX package". Available from CTAN, `macros/latex/contrib/frankenstein/unsupported`, 1996.

◇ Massimiliano Dominici
Pisa, Italy
`mlgdominici@interfree.it`



Scene i. – Franconia.

Apartment in the Castle of COUNT MOOR.
FRANCIS, OLD MOOR

FRANCIS But are you really well, father? You look so pale.

OLD MOOR Quite well, my son – what have you to tell me?

FRANCIS The post is arrived – a letter from our correspondent at Leipsic.

OLD MOOR (*eagerly*). Any tidings of my son Charles?

FRANCIS Hem! Hem! – Why, yes. But I fear – I know not – whether I dare – your health. – Are you really quite well, father?

OLD MOOR As a fish in water. Does he write of my son? What means this anxiety about my health? You have asked me that question twice.

FRANCIS If you are unwell – or are the least apprehensive of being so – permit me to defer – I will speak to you at a fitter season. – (*Half aside.*) These are no tidings for a feeble frame.

OLD MOOR Gracious Heavens? what am I doomed to hear?

1

Figure 1: First page of first act of *The Robbers*

```
\documentclass[a5paper,showtrims,11pt]{memoir}
\usepackage{dramatist}

%% Layout
\settrimmedsize{18,5cm}{13cm}{*}
\setlength{\trimedge}{\stockwidth}
\addtolength{\trimedge}{-\paperwidth}
\settrims{0pt}{\trimedge}
\settypeblocksize{*}{22pc}{1.71}
\setlrmargins{*}{*}{1.5}
\setulmargins{*}{*}{1}
\setlength{\footskip}{20pt}
\checkandfixthelayout
\ifpdf
\setlength{\pdfpageheight}{\stockheight}
\setlength{\pdfpagewidth}{\stockwidth}
\fi

\renewcommand{\printscenenum}{%
    \scenenumfont \thescene}
\setlength{\beforesceneskip}{20pt}
\pagestyle{plain}

\begin{document}

\Character{MAXIMILIAN, COUNT VON MOOR.}{OLD MOOR}
          {moor}
\Character{FRANCIS, his Sons.}{FRANCIS}{fran}

[...]

\scene[. -- Franconia.]

\StageDir{\begin{center} Apartment in the Castle of
COUNT MOOR.\\\fran, \moor\end{center}}

\begin{drama}
\franspeaks  But are you really well, father?
You look so pale.

\moorspeaks  Quite well, my son -- what have
you to tell me?

\franspeaks  The post is arrived -- a letter
from our correspondent at Leipsic.

\moorspeaks \direct{eagerly}. Any tidings of my son
Charles?

\franspeaks  Hem!  Hem! -- Why, yes.  But I fear
-- I know not -- whether I dare -- your health.
-- Are you really quite well, father?
[...]
\end{drama}
[...]
\end{document}
```

Figure 2: A play in prose: input code for *The Robbers*

P

ARNALDO da Brescia.
ADRIANO IV; pontefice.
GIORDANO PIERLEONI.
LEONE FRANGIPANI.
ANNIBALDO; nobile Romano.
GUIDO; cardinale di Santa Pudenziana.
OTTAVIANO; cardinale di Santa Cecilia.
Un CARDINALE di Santa Maria in Portico.
Alcuni altri CARDINALI.
SENATORI ROMANI.
POPOLO ROMANO.
LEGATI della Repubblica Romana.
PIETRO; prefetto di Roma.
Un SACERDOTE che annunzia la scomunica al Popolo Romano.
ALCUNI DEL CLERO.
OSTASIO; conte di Campania e seguace di Arnaldo.
ADELASIA; sua moglie.
DONNE ROMANE devote e penitenti del cardinal Guido.
Un Monaco; Mandato di un cardinale.
Un CAMERIERE segreto del papa.
Un ARALDO del papa.
CAPITANI E SOLDATI SVIZZERI, seguaci di Arnaldo.
CAPITANI E SOLDATI della Repubblica Romana.
GALGANO E FERONDO, soldati di Giordano.

Figure 3: The "Dramatist Personæ" list from *Arnaldo da Brescia*

A

Piazza vicina al Campidoglio.

S –

GIORDANO, LEONE, POPOLO

GIORDANO
Destatevi... sorgete... il nostro sangue
Si tra ca nel tempio; e son raccolti,
Tenebrosa congrega, i cardinali
A vestir del gran manto un altro lupo
Che pastore si chiami. Un dì sceglieste,
O Romani, il pontefice : gli antichi
Dritti il fero Innocenzo appien vi tolse,
E compi l'opra d'Ildebrando audace.
Cesare colla stola, ei far volea
Del mondo un tempio onde l'amor fuggisse,
Uno il pensiero, uno il volere, ed uno
Tiranno a un tempo, e sacerdote, e Dio.
Mirate l'opra sua! Roma deserta
Dal Laterano al Colosseo : guidava
Il normando furore e il saracino;
Fremea la sua preghiera, e maledisse
Colui che non insanguina la spada .
Imprecando morì: così perdonano
I vicari di Cristo ai lor nemici.
Barbari cardinali alzan dall'are
Colle man sanguinose un Dio di pace,
E coi rifiuti delle mense opime
Dopo i veltri ci pascono. Latino
Sangue gentile, sopportar saprai
Servitù così vile? ognor costoro

Figure 4: First page of first act of *Arnaldo da Brescia*

```
\documentclass[a5paper,showtrims,11pt]{memoir}
\usepackage[T1]{fontenc}
\usepackage[latin1]{inputenc}
\usepackage[italian]{babel}
\usepackage{dramatist}
\settrimmedsize{18,5cm}{13cm}{*}
\setlength{\trimedge}{\stockwidth}
\addtolength{\trimedge}{-\paperwidth}
\settrims{0pt}{\trimedge}
\settypeblocksize{*}{22pc}{1.71}
\setlrmargins{*}{*}{1.5}
\setulmargins{*}{*}{1}
\setlength{\footskip}{20pt}
\checkandfixthelayout
\ifpdf
\setlength{\pdfpageheight}{\stockheight}
\setlength{\pdfpagewidth}{\stockwidth}
\fi
\frenchspacing
\pagestyle{plain}

\Character{ARNALDO da Brescia.}{ARNALDO}{arn}
\Character{ADRIANO IV; pontefice.}{ADRIANO}{adr}
\Character{GIORDANO PIERLEONI.}{GIORDANO}{gior}
[...]
\DramPer

\begin{drama*}
\act

\StageDir{Piazza vicina al Campidoglio.}

\scene

\StageDir{\gior, \leo, \popo}

\begin{drama*}
\giorspeaks
Destatevi\dots\  sorgete\dots\  il nostro sangue\\
Si traffica nel tempio; e son raccolti,\\
Tenebrosa congrega, i cardinali\\
A vestir del gran manto un altro lupo\\
Che pastore si chiami. Un dì sceglieste,\\
[...]
\end{drama*}
[...]
\end{document}
```

Figure 5: A play in verse: input code for *Arnaldo da Brescia*

Variable width boxes in LaTeX

Simon Law

Seasoned LaTeX users are familiar with the default box commands: `\makebox`, `\framebox` and `\parbox`. They are the building blocks for page layout, and are commonly used. After all, being able to create boxes allows a typesetter great flexibility in positioning objects on a page. Figure 1 illustrates a simple use of `\parbox`.

Hello brave world Goodbye cruel world

Figure 1: Using `\parbox` to position text

1 Traditional `\parbox`

As you can see, I was able to align the two boxes so that each would be aligned. Looking at the source code in Figure 2, you'll see that I had to manually specify the box widths.

```
\parbox[t]{1cm}{Hello\\brave\\world}
\parbox[b]{1.5cm}{Goodbye\\cruel\\world}
```

Figure 2: `\parbox` source code

Of course, guessing the width of the longest line gets tedious. You can try using `\settolength` on the longest line, but that might change as your text changes.

2 Using `\pbox`

In order to automatically determine the width of the box, we will use the pbox[1] package. It provides the `\pbox` command, which is analogous to the `\mbox` command. In Figure 3, I typeset the same text using `\pbox` instead.

```
\pbox[t]{\textwidth}{Hello\\brave\\world}
\hspace{0.1cm}
\pbox[b]{\textwidth}{Goodbye\\cruel\\world}
```

Figure 3: `\pbox` source code

The syntax for `\pbox` is quite similar to that of `\parbox`. You must provide the maximum width of the box (*max-width*) and the contents (*text*):

`\pbox[`*pos*`][`*height*`][`*inner-pos*`]{`*max-width*`}{`*text*`}`

By default, the centre of each box will be vertically aligned. However, the three optional arguments allow you to align the `\pbox` as necessary. These options work exactly like their `\parbox` counterparts.

3 Now with `minipage`

This works well for simple paragraphs, where environments need not be embedded. However, once you start needing the features of the `minipage` environment, you begin to run into the same problems. David Arseneau has solved this problem with his varwidth[2] package.

An example use would be to centre a `verbatim` environment. This is normally done in a `minipage` because the `verbatim` environment left-flushes all its text against the left margin. In order to use the `minipage`, you still have to figure out the width of its contents and specify it manually.

```
#include <stdio.h>
int main()
{
  printf ("Hello world!\n");
  return 0;
}
```

Figure 4: Centered source code example

Figure 4 shows a snippet of source code that is representative of a sample in an article or a textbook. The code in Figure 5 illustrates how to typeset this without manually determining the width.

```
\centering
\begin{varwidth}{\columnwidth}
\begin{verbatim}
#include␣<stdio.h>
int␣main()
{
␣␣printf␣("Hello␣world!\n");
␣␣return␣0;
}
\end{verbatim}
\end{varwidth}
```

Figure 5: `varwidth` source code

4 Conclusion

Both the pbox and varwidth packages are useful extensions to standard LaTeX 2ε. They allow typesetters to place boxes and minipages throughout their documents without the need for guessing widths.

◇ Simon Law
sfllaw@law.yi.org
http://www.law.yi.org/~sfllaw/

[1] http://www.ctan.org/tex-archive/macros/latex/contrib/pbox/

[2] http://www.ctan.org/tex-archive/macros/latex/contrib/misc/varwidth.sty

Macros

xkeyval — new developments and mechanisms in key processing

Hendri Adriaens and Uwe Kern

Abstract

This article introduces the xkeyval (LA)TEX package, an extension of the well-known keyval package. The new package provides more flexible commands, syntax enhancements, and a new option processing mechanism for class and package options using the *key=value* syntax.

1 Introduction

The keyval package [2] written by David Carlisle is widely used by package authors to provide the means for users to easily specify numerous optional arguments for macros. The main advantages of using keyval are that (1) the number of optional arguments is no longer limited to 9 and that (2) the arguments are named, and hence there is less chance of confusion about the syntax of a macro.

The package provides ways to define so-called "key macros" which handle the input of the user. These key macros end up defined with the form `\KV@family@keyname`, where the `KV` is a literal prefix to avoid collisions. They should take one argument to handle user input. A macro to handle the key `pi` can, for instance, be defined by

```
\define@key{myfam}{pi}{\setlength{\parindent}{#1}}
```

This defines a macro named `KV@myfam@pi`. Such key macros are called when `\setkeys` is invoked to set the keys. In our case, when `pi` is used, the key macro will set `\parindent` to the given value. Here is a typical example of its use:

```
\setkeys{myfam}{pi=10pt,pn=Page~\thepage}
```

The packages keyval and xkeyval are mainly directed to class and package authors. The various `\define@key` commands usually go into the document preamble or the package and the main interface for users is given by `\setkeys`.

The `xkeyval.tex` can be used with plain TEX; all the functionality described here is available, with the exception of the '`X`' macros listed in section 3.

Editor's note: This article was first published in *MAPS* 31 (2004), and is reprinted, with additions, by kind permission of the authors and editor.

2 Why a new package?

When working on another package, the need arose to have multiple families in the package. Each family would provide keys for a particular macro or environment. This provided the means to block the use of illegal keys in a macro argument, which could have a destructive effect on the rest of the document. However, it would also be nice to be able to allow the user to set specific keys of each macro or environment globally in the preamble. One could, for instance, think of allowing the user to set the markup of all `example` and `exercise` environments in the document in the preamble, but disallowing changing the markup of `example` environments locally in `exercise` environments and vice versa. In more complicated settings, specifying keys in macros which are not designed to handle those keys can easily lead to almost untraceable errors. That was the start of the xkeyval package [1].

However, in the process of generalizing keyval, we noticed that a lot of packages had already tried extending the features, all in their own way. Quite a few packages, for instance, provide a system to allow the use of keys and values in `\usepackage` commands. The most famous examples are the hyperref, geometry and beamer packages. All of these approaches differ in details and are not portable to other packages without reprogramming. This called for a unified approach.

Another extra feature, found for instance in the hyperref package, is the availability of boolean keys which can only be true or false. hyperref actually implements this within the ordinary key system, using `\define@key`. However, since (part of) the function to be executed on the use of the key is known in advance (namely, set an "if" command to true or false depending on the input), the system can be simplified.

A final motivation for the new package is based on the fact that the development of the keyval package seemed to have paused since 1999 and that fundamental changes and improvements to the system could more easily be made with a new package. Among the improvements, we find macros for creating package options that can take values, new types of keys, the use of multiple families in `\setkeys`, the pointer syntax, the preset system, robust input parsing and support for the PSTricks family of packages. The remaining sections of this article will discuss these new developments.

3 Keys and values in package options

First of all, the package supplies macros to declare

class or package options, execute them and process them. The macros are available under the usual LaTeX names, but all with the suffix `X`, namely

```
\DeclareOptionX
\DeclareOptionX*
\ExecuteOptionsX
\ProcessOptionsX
```

These commands allow the user to assign a value to an option just like when using `\setkeys`. The first macro is based on `\define@key` and the final two are based on `\setkeys`. Supposing that a package mypack is set up with these commands, a user could for instance do

```
\usepackage[textcolor=red,font=times]{mypack}
```

These macros are fully integrated with the LaTeX option system. This, for instance, allows packages to copy global options specified in the `\documentclass` command, to pass options to other classes or packages and to update the list of unused global options that will be displayed by LaTeX in the log file.

However, key values like `author=\textit{Me}` in class or package options are not allowed, although they could easily be processed by `\setkeys`. This restriction results from the design of LaTeX's option processing mechanism, which expands the entire option list (keys and values) completely, causing obvious trouble.[1]

To avoid these premature expansions, several kernel macros need to be redefined. xkeyval includes the xkvltxp package which contains these new definitions. Loading this package before loading the class or package which uses xkeyval for option processing will allow class and package options to contain expandable macros. This file will not be included in the LaTeX 2_ε kernel since it might introduce compatibility conflicts for those using an old kernel with new packages which might depend on this new functionality.

4 Prefixes, families, keys and pointers

The package provides extended syntax for all of the commands provided by keyval.[2] The syntax for defining keys has been extended with an optional argument to set the prefix of the key macro. It is good practice for package authors to use a package specific prefix for all internal macros so as to avoid possibly redefining a macro of another package. Moreover, this optional argument allows for defining and setting keys in specialized systems such as implemented in the PSTricks package. More details about this system will be discussed later, in the section about the pst-xkey package.

The syntax for setting keys using `\setkeys` has been adjusted accordingly. Also, one can specify a list of families which should be scanned when setting keys, as discussed in the introduction. For instance,

```
\setkeys{font,page}{fs=10pt,pn=Page~\thepage}
```

The package also provides new types of keys. These are choice keys, which allow for a limited number of possible input values, and boolean keys, which are a special type of choice key and only take the values `true` and `false`. An example is below.

```
\define@choicekey{fam}{keya}{\fbox,\mbox}{#1{text}}
\define@boolkey{fam}{keyb}{%
  \ifKV@fam@keya we continue\else we stop here\fi
}
\setkeys{fam}{keya=\mbox,keyb=false}
```

These keys generate an error when the user specifies a value that is not allowed.[2] The package provides a viewer utility in xkvview to generate tables with information about defined keys.

Part of the new syntax is also the possibility of using pointers to keys. Pointers allow assigning to `keyb` the value that has been assigned to `keya`, irrespective of what that value is. For example

```
\setkeys{family}{\savevalue{keya}=red,%
  keyb=\usevalue{keya}}
```

Here, `\savevalue` will make xkeyval save the value submitted to `keya`. `\usevalue` will use this value again. (One can use the `\savekeys` command to avoid typing `\savevalue` every time.) If, in this example, `red` is changed to `blue` no changes are necessary to the value of `keyb` to assign it `blue` as well. This is an obvious similarity to TeX's behaviour in the macro case `\def\cmdb{\cmda}`.

This pointer system can be used as well in the default value system. This system submits a default value to the key macro in case the user has used the particular key, but didn't assign a value to it. One could, for example, define the keys

```
\define@key{fam}{keya}{keya: #1 }
\define@key{fam}{keyb}[\usevalue{keya}]{keyb: #1 }
```

Then the following use of `\setkeys`

```
\setkeys{fam}{\savevalue{keya}=test,keyb}
```

would result in typesetting

```
keya: test keyb: test
```

[1] Note that `author=\protect\textit{Me}` is *not* a solution for this problem.

[2] Please refer to the documentation of the xkeyval package to learn about further syntactical details which are not discussed in this article.

We will discuss some technical details regarding the pointer syntax. First of all, the control sequences \savevalue and \usevalue are not defined! Instead, the package uses them as delimiters. A simple parsing step determines if \savevalue has been used in the key name part. Parsing is also used to substitute occurrences of \usevalue by the saved value. When a pointer is replaced, its replacement will also be scanned again for pointers. This allows for nested pointers in key values. Moreover, it ensures that, once the value is submitted to a key macro, this value does not contain pointers anymore.[3]

The replacement process is a little trickier when the user did not submit a value to the key. In this case, the default value of a key (if present) should be scanned for pointers. Default value macros are set up like this:

```
\def\prefix@fam@key@default{%
  \prefix@fam@key{the default value}%
}
```

The macro \prefix@fam@key@default will be executed when the user did not supply a value to the key.

This system has been introduced by keyval and many packages use it. However, some packages do not use it in the way intended by keyval. For instance, the fancyvrb package defines default value macros to execute arbitrary code rather than the standard \prefix@fam@key. To retain compatibility with existing packages, we must support this; otherwise, we could do something much cleaner, e.g., define \prefix@fam@key@default as 'the default value' in the first place, without the extra macro invocation.

This is an important restriction for the pointer system since we want to retrieve the default value from the default value macro and scan it for pointers. So, xkeyval proceeds as follows. It first checks whether the default key macro starts as expected, namely with a key macro \prefix@fam@key. If that is the case, it locally redefines the key macro to save the value to a temporary macro and then executes the key macro. The temporary macro then contains the default value which can be scanned for pointers. If the default value macro is not of the expected form, as with fancyvrb, then xkeyval just executes it without attempting to retrieve the default value or replace pointers.

[3] Unless the pointer is hidden to xkeyval inside a group.

5 Preset system

The default value system operates when users specify keys, but no value for the keys. But the keyval package does not provide a way to assign values to keys that have not been used at all by the user. In many applications, one would like to implement default values for keys when they are not used. For instance, 'scale this figure with factor 1 unless specified otherwise by the user'. One could go ahead and call the key macro with a preset value and afterwards, submit the user input to \setkeys and possibly overwrite the values that you have just set. This is possible (but quite cumbersome when there are many keys) in cases where keys do not generate material themselves, but, for instance, only set a length.

But what happens if we apply this scheme to keys which are defined as follows?

```
\define@key{fam}{keya}{Your input was: #1}
\define@key{fam}{keyb}{\edef\list{\list,#1}}
```

If we follow the scheme in the first example, both our preset value as well as the user input (if present) will be typeset. In the second example, both the preset value and the user input will be added to the list contained in \list.

To avoid this, xkeyval introduces the preset system. First one declares the keys that should always be assigned and their values using \presetkeys, for instance

```
\savekeys{fam}{head}
\presetkeys{fam}{head=red}{tail=\usevalue{head}}
```

The reason to have two arguments containing key presets in the \presetkeys macro will become clear in a moment.

Now, when submitting user input for keys in the family fam, the macro \setkeys will determine which keys will be set by the user and will avoid setting them again with the preset values. Keys that are not set by the user will be set by the values specified in \presetkeys.

However, when pointers are used, there is one thing about this system that we should keep in mind. If the pointer points to a key which is assigned a value afterwards, the pointer cannot know this value yet and errors will occur. Hence, it is best (in most situations) to execute preset pointers at the very end as done in the example above.

A similar discrepancy can occur when keys without pointers in the values are preset after setting the user input. Users then can't use pointers to these presets as they are preset in a later stage of execution. Hence, for keys without pointers in the value,

it is best to execute them at the very beginning, before setting user input.

That is why the `\presetkeys` macro has two arguments: the first one (usually containing keys and values without pointers) will be inserted before setting user input keys, the second one (containing pointers to preset values or user input) afterwards.

This system is especially useful when you can't rely on key values remaining local to a macro or environment since the preset system will, at every use of your macro or environment, reset key values to the preset value unless overwritten locally by the user. This needs some more explanation. `\def` definitions (for instance made by key macros) will be destroyed by TEX when leaving a group or environment. Hence the values will remain local. However, if your keys do not always use `\def`, but for instance, `\gdef`, such global definitions will escape the group or environment and might distort all following macros or environments. Hence, you will have to take care to reinitialize the key values at every use of the macro or environment.

This is, however, not necessary anymore with the preset system. Once the preset keys have been defined for a specific family, each time this family is used in the `\setkeys` command, the preset values will be taken into account together with the user input.

The following example will demonstrate the power of the preset system in combination with pointers. Below the example, you can find its output and the explanation. Let's assume we want to create a simple frame/shadow box command with the following *default* behaviour:

- a shadow will be drawn if and only if the box is framed;
- the shadow color should be a 40% tint of the frame color, thus being clearly discernible;
- the shadow size (or width) should be 4 times the width of the frame.

Certainly, the user should be able to overrule each of these default parameter relations when the box command is actually applied.

```
\documentclass{article}
\usepackage{xkeyval}
\usepackage{calc,xcolor}

\makeatletter
\newdimen\shadowsize
\define@boolkey{Fbox}{frame}[true]{}
\define@boolkey{Fbox}{shadow}[true]{}
\define@key{Fbox}{framecolor}%
 {\def\Fboxframecolor{#1}}
\define@key{Fbox}{shadowcolor}%
 {\def\Fboxshadowcolor{#1}}
\define@key{Fbox}{framesize}%
 {\setlength\fboxrule{#1}}
\define@key{Fbox}{shadowsize}%
 {\setlength\shadowsize{#1}}
\savekeys{Fbox}{frame,framecolor,framesize}
\presetkeys{Fbox}%
 {frame,framecolor=black,framesize=0.5pt}%
 {shadow=\usevalue{frame},
  shadowcolor=\usevalue{framecolor}!40,
  shadowsize=\usevalue{framesize}*4}
\newcommand*\Fbox[2][]{%
 \setkeys{Fbox}{#1}%
 {\ifKV@Fbox@frame\else\fboxrule0pt\fi
  \ifKV@Fbox@shadow\else\shadowsize0pt\fi
  \sbox0{\fcolorbox{\Fboxframecolor}{white}{#2}}%
  \hskip\shadowsize
  \color{\Fboxshadowcolor}%
  \rule[-\dp0]{\wd0}{\ht0+\dp0}%
  \llap{\raisebox{\shadowsize}%
   {\box0\hskip\shadowsize}}}%
}
\makeatother

\begin{document}
\Fbox{demo1}
\Fbox[framecolor=gray]{demo2}
\Fbox[shadow=false]{demo3}
\Fbox[framesize=1pt]{demo4}
\Fbox[frame=false,shadow]{demo5}
\end{document}
```

demo1 demo2 demo3 demo4 demo5

First of all, lines 7 to 16 define the keys to be used in the example. The `\presetkeys` command in line 18 defines the presets: the frame will be set to true, its color to black and the frame size to 0.5 pt, unless the user provides different specifications for these keys. The requirements listed above are then covered by the pointer expressions in the next argument.

The first box application now shows the default box without additional user input. We see a frame and a shadow, based on the color black. The second box shows that the user input for the frame color will overwrite the preset values and turn the box gray. But since the shadow color equals the frame color by default, the shadow is light gray. In the third example, we have a frame, but no shadow. Notice that the frame color has returned to black, the preset value. The fourth box has an increased frame size and hence an increased shadow size as well due to the pointer use when presetting the keys. The last example shows that it is possible to overwrite the preset behaviour of linking shadows to frames: it displays a shadow without a frame.

6 Robust parsing

Just as with the pointer delimiters `\savevalue` and `\usevalue`, keyval and xkeyval treat the comma and

the equality sign as delimiters. In the past, this has led to problems. A well known incompatibility exists between the Turkish language version of the babel package and all packages using keyval. Since Turkish babel changes the catcode of the equality sign for shorthand notation, the parsing macros of keyval cannot detect these characters anymore and will generate errors.[4]

xkeyval solves this by sanitizing (i.e. setting the catcode to 12) all characters necessary to parse the input properly. This is done using the macro `\@selective@sanitize`, which can sanitize one or more different characters in a single run. Moreover, the sanitize group depth can be controlled. xkeyval implements the macro such that only commas and equality signs appearing in the top level of a key value will be sanitized, since that is all that's needed for input parsing. Characters inside groups are left untouched and can hence contain even babel shorthand notation without causing errors:

```
\usepackage[turkish]{babel}
...
\setkeys{fam}{key={some =text}}
```

In this example, the first '=' will be sanitized for parsing, whereas the second '=' will remain untouched and thus keeps its original meaning.

7 Redefining macros?

Obviously, redefining existing macros is dangerous in general. Nevertheless, the xkeyval package redefines the two major keyval macros `\define@key` and `\setkeys`. The reason is that this avoids any confusion of having several systems running next to each other, doing approximately the same things.

Although xkeyval supports all of the syntax allowed by the original keyval package, we still had to check the packages using keyval before we could make the decision to redefine the macros. Three major issues came up in that process.

First of all, we found that some packages were using keyval internals directly instead of the user interface formed by `\define@key` and `\setkeys`. To avoid any errors of undefined control sequences in these packages, xkeyval loads the keyval internals if keyval hasn't been loaded before.

Secondly, certain packages implemented a creative use of the default value system as has been discussed in the section about the pointer syntax. The solution in xkeyval has also been discussed there.

[4] See for more information concerning this problem of keyval and babel: `http://www.latex-project.org/cgi-bin/ltxbugs2html?pr=babel/3523`

Finally, we found that the pst-key package was redefining `\define@key` and `\setkeys` itself to provide the means of setting PSTricks keys. After discussing this with the PSTricks maintainer Herbert Voß, we agreed that xkeyval would develop a unified approach to keys and values and that the pst-key package would be abandoned. More information on the development related to PSTricks is provided in the final section of this article.

After redefining the necessary macros, xkeyval will make sure that the keyval package cannot be loaded subsequently, in order to avoid again redefining the xkeyval macros. This was the final step necessary in safely redefining the keyval macros and providing a system to which all package authors can convert their package without too much effort.

8 The pst-xkey package

An important stream of packages will be using xkeyval in the near future. These are the PSTricks packages [3, 4]; for key and value processing, they currently rely on a combination of private definitions in `pstricks.tex` and pst-key, the latter being a modification of the keyval package.

Due to the popularity and flexibility of the PSTricks package, several people have contributed extensions to the original distribution. Unfortunately, all PSTricks keys used to have the same form, namely `\psset@somekey`; thus, PSTricks authors have needed to check all existing packages to be sure not to redefine an existing key.

The PSTricks maintainer Herbert Voß has recognized this problem and soon the work on xkeyval started to provide a way to define and set PSTricks keys via this package. The major advantage would be the possibility for individual package authors to nest their keys in a well chosen family (for instance, the package name) and avoid the need to check other packages for existing keys.

In order to make this possible, `\define@key` and `\setkeys` needed to be adjusted so that the standard keyval prefix `KV` could be changed, for instance to `psset`. Further, the `\psset` macro needed to be redefined to use the new `\setkeys` and let this scan all families available. When a PSTricks package is loaded, it adds all families used in the package to a list and this list will be used in `\setkeys`. Since all separate packages will use different families, reusing key names is not a problem anymore. The redefinition of `\psset`, along with some other macros necessary to do the job, is available in the `pst-xkey` package which comes with the xkeyval package.

Due to the vastness of the PSTricks collection of packages, the conversion of all packages to use pst-

xkey instead of pst-key will take some time, but has already started and should be finished in the near future.

References

[1] Hendri Adriaens. xkeyval package, v2.4, 2005/03/31. CTAN:/macros/latex/contrib/xkeyval.

[2] David Carlisle. keyval package, v1.13, 1999/03/16. CTAN:/macros/latex/required/graphics.

[3] Herbert Voß. PSTricks web site. http://www.pstricks.de.

[4] Timothy Van Zandt et al. PSTricks package, v1.04, 2004/06/22. CTAN:/graphics/pstricks.

◇ Hendri Adriaens
hendri[at]uvt.nl
http://stuwww.uvt.nl/~hendri

◇ Uwe Kern
tex[at]ukern.de
http://www.ukern.de

A non-expert looks at a small TEX macro

David Walden

Introduction

I use TEX a lot, but I seldom dig deeper into how TEX works than I must in order to address the immediate writing project I am working on. However, once I think I have figured out something new, I like to write it up to help me be sure I understand it. In this piece I describe a simple LATEX macro I wrote, how the macro evolved, and what I learned along the way. Perhaps other intermediate users who have a similar incremental approach to increasing their capabilities to use TEX will find reading my account a short cut to understanding of their own.

My problem

In some documents I write, I use an extra blank line and an extra large letter on the first character of the first word of a paragraph to indicate a thought break.

Here is an example.

A couple of years ago, I wrote a simple LATEX macro to accomplish this:

```
\newcommand{\newthoughtgroup}[1]{%
  \bigskip\noindent{\Large #1}}
```

It was called as follows:

```
\newthoughtgroup{H}ere is an example.
```

However, I didn't like having the first word of the paragraph in my LATEX file being split as in the above line. I wished the macro call could be

```
\newthoughtgroup{Here} is an example.
```

but still only make the first character of the first word larger.

Search and discovery

Therefore, I looked around for a way to have the whole first word be part of the macro argument — I had to look around since I didn't understand TEX macros well enough to be able to figure it out myself.

First approach. I discovered the following pair of macros on comp.text.tex (April 6, 1994) in a posting by Victor Eijkhout, who was answering a question about making the first letter of a word be upper case:[1]

```
\def\CapString#1{%
  \CapFirstLetter#1$} %assumes no $ in arg 1
\def\CapFirstLetter#1#2$%{
  \uppercase{#1}#2}
```

Without fully comprehending how Eijkhout's macros worked, I changed them as follows to accomplish my purpose:

```
\def\newthoughtgroup#1{%
  \BigFirstLetter#1$}
\def\BigFirstLetter#1#2$%{
  \bigskip\noindent{\Large #1}#2}
```

I suspect I am not alone among TEX user in blindly copying or converting something that already exists without much understanding of how it works.

Learning more. After using my version of Eijkhout's macros for a while, I decided to try to understand them in detail. So, I looked at chapter 20 of Knuth's *The TEXbook*;[2] in particular, I tried to understand from the first dangerous bend signs on page 203 to the first dangerous bend signs on page 204. The following is what I think I learned.[3]

First, I noted the difference between LATEX macro definitions and TEX macro definitions. My original LATEX macro listed above might be written as a TEX macro as follows:

[1] I've suddenly jumped to TEX style macro definitions instead of the LATEX form of macro definitions because that is what I found searching comp.text.tex, and for another reason that may become apparent.

[2] Addison Wesley, Reading, MA, 1986.

[3] I am not going to repeat the full explanation of a macro definition or how a macro finds its arguments when called; I'll just use what I learned to explain the macros I was working with.

```
\def\newthoughtgroup#1{%
  \bigskip\noindent{\Large #1}}
```

The TEX form of macro definition includes `\def`, followed by the new macro name (`\newthoughtgroup` in our case), followed by what Knuth calls the *parameter text* which in this case is `#1` indicating the macro has one *undelimited parameter*, and ending with the *replacement text* (`\bigskip\noindent{\Large #1}`). The call-time argument of an undelimited parameter is the first non-blank *token*,[4] or the tokens enclosed in matched braces, after the macro name.

This same format of TEX macro definition is used for the first macro below.

```
\def\newthoughtgroup#1{%
  \BigFirstLetter#1$}
\def\BigFirstLetter#1#2$%
  \bigskip\noindent{\Large #1}#2}
```

The parameter text is `#1`, and the replacement text is `\BigFirstLetter#1$`. Thus, when the first macro is called with

```
\newthoughtgroup{Here}
```

the macro is *expanded* into its replacement text, which thus becomes `\BigFirstLetter Here$`.[5]

But the second macro's parameters specify a slightly different form of macro call. The first parameter (`#1`) is undelimited and, thus, the macro call's first argument is the first (non-blank) token or tokens enclosed in braces (as with the first macro). The second parameter, however, is *delimited* by the following `$` and, thus, the macro call's second argument is all the tokens from the end of the first argument to the `$`, i.e., to the delimiter.

Thus, when the first macro calls the second macro, that macro call (`\BigFirstLetter{Here$}`) finds its first argument to be `H` and its second argument to be `ere` with the `$` being discarded after matching. In turn, the call to `\BigFirstLetter` is replaced by

```
\bigskip\noindent{\Large H}ere}
```

producing the desired vertical space, no indentation, a big `H`, and normalsize `ere`.

Second approach. I happily used these macro definitions for a long time until I discussed them one day recently with Karl Berry. He pointed out that my version of Victor's formulation can be changed to remove that restriction on including `$` in the argument. He explained that the second argument's delimiter doesn't have to be a character; it can be an arbitrary control sequence (even an undefined control sequence), and he wrote down the following for me:[6]

```
\def\newthoughtgroup#1{%
  \BigFirstLetter#1\enddavesmacro}
\def\BigFirstLetter#1#2\enddavesmacro{%
  \bigskip\noindent{\Large #1}#2}
```

Third approach. That sounded like a good improvement, but then Karl said, "Personally, I would be inclined to a different approach, that has the benefit of being called without braces — which thus addresses your original reason for moving from a macro called with `\newthoughtgroup{H}ere`." He showed me the following definition for `\newthoughtgroup`:

```
\def\newthoughtgroup#1{%
  \bigskip\noindent {\Large #1}}
```

When called, for example, as

```
\newthoughtgroup Here is an example.
```

the argument that replaces the parameter (`#1`) is the `H`, i.e., the first non-blank token.[7]

Conclusion

As I started drafting this conclusion, it gradually dawned on me that the Third Approach TEX macro is the same as the TEX transliteration of my original LATEX macro ("Learning more" section), and perhaps my original LATEX macro ("My problem" section) also worked when called without braces:

```
\newthoughtgroup Here is an example.
```

[4] Tokens are described between exercises 7.2 and 7.3 on pages 38–39 of *The TEXbook*. As what the user typed is read into TEX, the letters, numbers, command names, etc., are stored as *tokens*. Tokens are internal representations of the characters in the input stream, with the notable exception that *control sequences* (e.g., `\bigskip`, `\def`, `\newthoughtgroup`) are each stored as single tokens. Macro definitions are stored as tokens, and macro calls are processed in terms of tokens.

[5] My macros are usually so simple that I can just think of the literal characters of the macro definition replacing the literal characters of the macro call in the sequence of characters that TEX reads, and so the definition `\newthoughtgroup` in this section originally looked funny to me. I wondered why the replacement text for `\newthoughtgroup{Here}` wasn't `\BigFirstLetterHere$` and then wondered why TEX didn't report that as an undefined control sequence. The answer, I believe, is that, as noted in footnote 4, TEX processes macros in terms of tokens, and the replacement text, `\BigFirstLetter#1$`, is manipulated as three distinct tokens: `\BigFirstLetter`, `#1`, and `$`.

[6] Victor also showed me a different formulation — one optimized for efficiency — that I will not try to explain in this note.

[7] Karl was not quite done yet. His final note was that if I was willing to stop trying to figure out macros like these, the "lettrine" package has support for many variations along the lines I desired. See `http://www.tex.ac.uk/cgi-bin/texfaq2html?label=dropping` for mention of the package and `http://www.tex.ac.uk/tex-archive/macros/latex/contrib/lettrine/doc/demo.pdf` for a demonstration document.

It does — a bit of a startling conclusion for me.

There are two possible lessons here. Perhaps I originally should have posed my real problem to `comp.text.tex` rather than searching for "first letter of a string"; I might have been pointed in the right direction of understanding how TEX macro calls find their arguments. Or perhaps it paid to wander in some less-than-optimal directions; my journey of discovery was enlightening and relatively painless, and trying to explain it in writing definitely consolidated my knowledge — and I hope helped you.

Acknowledgements

I appreciate Victor Eijkhout's deep understanding of how the TEX program processes the TEX language (his book *TEX by Topic* has a comprehensive discussion of how TEX processes macros, `http://www.eijkhout.net/tbt/`) and also the deep understanding of Karl Berry and his suggestions as I prepared this paper.

Biographical note

David Walden is retired after a career as an engineer, engineering manager, and general manager involved with research and development of computer and other high tech systems. These days he does a lot of writing.

◇ David Walden
East Sandwich, MA
`www.walden-family.com/dave`

Hints & Tricks

Glisterings

Peter Wilson

> All that glisters is not gold —
> Often have you heard that told.
>
> *Merchant of Venice*, Act II scene 7
> William Shakespeare

The aim of this column is to provide odd hints or small pieces of code that might help in solving a problem or two.

Corrections, suggestions, and contributions will always be welcome.

An issue that has cropped up recently on the `comp.text.tex` (`ctt`) newsgroup is what can be done when two packages clash by defining the same macro.

> And we are here as on a darkling plain
> Swept with confused alarms of struggle and flight,
> Where ignorant armies clash by night.
>
> *Dover Beach*
> Alfred, Lord Tennyson

1 Package/package clashes

A very simple method of undefining a macro, perhaps `\amacro`, is to let it be undefined, as:

```
\let\amacro\undefined
```

Of course, `\undefined` must never be defined. You might feel safer if instead you used, say

```
\let\amacro\uNdEFiNed
```

or some other unlikely name.

If two packages are being used, say packA and packB, which both create `\amacro` then, provided the second has used `\newcommand` and not the TEX `\def` macro which will silently replace any prior definition, it will complain that `\amacro` is already defined. If the definitions in packA and packB are identical then the following resolves the problem.

```
\usepackage{packA}
\let\amacro\undefined
\usepackage{packB}
```

Life being what it is, the definitions are usually different. In this case both definitions can be used but the name of the first definition has to be altered.

```
\usepackage{packA}
\let\Aamacro\amacro
\let\amacro\undefined
\usepackage{packB}
```

Following this, you use `\Aamacro` when you want packA's version and `\amacro` for the packB version.

Of course, life gets even more awkward if packA uses `\amacro` as part of another macro that you might use, in which case you have to hope that the author of at least one of the packages will change it to eliminate the clash.

2 Class/package clashes

A slightly different version of the same problem is when there is some clash between the code in a class and the code in a package. I came across this when I was developing the memoir class [3] which incorporates code from many[1] packages. In some cases I

[1] Mostly written by me.

needed to make sure that a particular package was not used with the class. I came up with this macro that fooled LaTeX into thinking that a package had been loaded, even though it hadn't been. The argument to the macro is the package name.

```
\newcommand*{\@memfakeusepackage}[1]{%
  \@namelet{ver@#1.sty}\@empty}
\newcommand*{\@namelet}[1]{%
  \expandafter\let\csname #1\endcsname}
```

(The code must be put where @ is treated as a letter.)

The LaTeX kernel has two useful macros for composing and using macro names which do not necessarily consist only of letters, namely:
`\@namedef`{⟨*text*⟩}{⟨*def*⟩}, and
`\@nameuse`{⟨*text*⟩}.
The first of these lets you define a macro called \⟨*text*⟩ and the second lets you call a macro called \⟨*text*⟩ As an example, the result of the next piece of code is shown afterwards; note that you can't directly call a macro whose name includes analphabetic characters.

```
\makeatletter
\newcommand*{\ru}{are you}
\@namedef{ru4me}#1{#1, are you for me?}
‘\ru4me{Fred}’ he asked. \\
‘\@nameuse{ru4me}{Fred}’ he asked.
\makeatother
```

‘are you4meFred’ he asked.
‘Fred, are you for me?’ he asked.

In the same vein the macro
`\@namelet`{⟨*text*⟩}
defined above is for `\let`ing. Thus, calling

```
\@memfakeusepackage}{pack}
```

effectively expands to

```
\let\ver@pack.sty\@empty
```

which appears to be the magic incantation to make LaTeX believe it has already used the pack package.

The memoir class includes code very similar, but not identical, to the array, dcolumn, delarray and tabularx packages and I used `\@memfakeusepackage` to make sure these were not loaded again.

The memoir class also includes code corresponding to Heiko Oberdiek's ifpdf package [1] but I did not do anything to prevent loading the package. This resulted in a thread on `ctt` where the poster was using

```
\documentclass{memoir}
\usepackage{ps4pdf}
```

only to be told that `\ifpdf` was already defined. It turns out that the ps4pdf package uses the ifpdf package which defines `\ifpdf` which was also defined in memoir.

Heiko Oberdiek [2] gave the simple 'let to undefined' solution and the following more complex one:

```
\documentclass{memoir}
     %% memoir defines \ifpdf
\makeatletter
     %% save memoir's \ifpdf
\let\saved@ifpdf\ifpdf
     %% then undefine it
\let\ifpdf\@undefined
     %% use ifpdf package (defines \ifpdf)
\usepackage{ifpdf}
     %% is \ifpdf undefined?
\@ifundefined{ifpdf}{%
     %% yes, used the saved memoir version
  \let\ifpdf\saved@ifpdf
}{%
     %% no, check for matching definitions
  \ifx\ifpdf\saved@pdf
  \else
     %% mismatch, write error message
    \latex@error{Different meaning
              of \@backslash ifpdf}\@ehc
  \fi
}
\makeatother
     %% use ps4pdf which uses \ifpdf
\usepackage{ps4pdf}
```

This scheme can be applied to similar situations. Note that it produces an error if the second and first definitions are different, which could very well be useful.

References

[1] Heiko Oberdiek. The ifpdf package, July 2001. Available on CTAN in `latex/macros/contrib/oberdiek`.

[2] Heiko Oberdiek. Re: memoir, ps4pdf and `\ifpdf`. Post to `comp.text.tex` newsgroup, 3 September 2004.

[3] Peter Wilson. The memoir class for configurable typesetting, 2004. Available on CTAN in `latex/macros/contrib/memoir`.

◇ Peter Wilson
18912 8th Ave. SW
Normandy Park, WA 98166
USA
herries.press@earthlink.net

The Treasure Chest

This is a selected list of the packages posted to CTAN from January 2004 through December 2004, with descriptive text pulled from the announcement or researched and edited for brevity. Please inform us of any errors.

This installment, like the last, lists entries alphabetically within CTAN directories, rather than by date. We've also omitted some packages which had only minor updates, again for brevity.

Hopefully this column and its companions will help to make CTAN a more accessible resource to the TEX community. Comments are welcome, as always.

◇ Mark LaPlante
109 Turnbrook Drive
Huntsville, AL 35824
laplante@mac.com

biblio

`babelbib` in `biblio/bibtex/contrib`
: Generates multilingual bibliographies in cooperation with `babel`.

`bib-fr` in `biblio/bibtex/contrib`
: French translations of classical BibTeX styles.

`bib2xhtml` in `biblio/bibtex/utils`
: Convert BibTeX files into XHTML.

`bibtool` in `biblio/bibtex/utils`
: Manipulation of BibTeX files, including: sorting and merging, pretty-printing, syntax checks with error recovery, semantic checks, generation of uniform reference keys, controlled rewriting with regular expressions, collection of statistics, and more. Includes documentation. C source only (no binary).

`ebib` in `biblio/bibtex/utils`
: BibTeX database manager for GNU Emacs.

`IEEEannot` in `biblio/bibtex/contrib`
: Unofficial style for an annotated bibliography in the IEEE citation format.

`spain` in `biblio/bibtex/contrib`
: The traditional bibliographic style in Spain.

`vancouver` in `biblio/bibtex/contrib`
: BibTeX style to meet the "Uniform Requirements for Manuscripts Submitted to Biomedical Journals" (the Vancouver style).

dviware

`dvipng` in `dviware`
: Convert `.dvi` files to `.png` images.

fonts

`accfonts` in `fonts/utilities`
: Programs to generate accented fonts.

`antt` in `fonts/psfonts/polish/antt`
: Antykwa Toruńska is a two-element typeface designed by Zygfryd Gardzielewski, a Polish typographer. A great variety of characters, including many mathematical symbols, in many weights, are included.

`aurical` in `fonts`
: Calligraphic font resembling handwriting.

`bera` in `fonts`
: New fonts Bera Serif, Bera Sans, and Bera Mono, based on the Vera fonts made freely available by Bitstream. (Renamed due to the license conditions.)

`cirth` in `fonts`
: Tolkien's Cirth font.

`cm-lgc` in `fonts/ps-type1`
: Type 1 fonts converted from METAFONT sources of the Computer Modern font families.

`courier-scaled` in `fonts/ps-fonts`
: Sets the default typewriter font to Courier with a possible scale factor.

`dictsym` in `fonts`
: Symbols commonly used in dictionaries.

`esint` in `fonts/ps-type1`
: Eddie Saudrais's font `esint10` in Adobe PostScript Type 1 format.

`fc` in `fonts/jknappen`
: Fonts for African languages.

`fourier-GUT` in `fonts`
: Fourier-GUTenberg is a math complement for Adobe Utopia.

`fpl` in `fonts`
: Small caps and oldstyle digits for URW Palladio L.

`frcursive` in `fonts`
: This is the French Cursive font, a cursive handwriting font family in the style of the French academic running-hand, written with METAFONT.

`greektex` in `fonts/greek`
: Fonts for processing LaTeX files written in a mixture of Greek and English.

`hfoldsty` in `fonts`
: Provides virtual fonts for using oldstyle figures with the European Computer Modern fonts.

`ibygrk` in `fonts/greek`
: A collection of fonts and macros to typeset ancient Greek.

`kerkis` in `fonts/greek`
: Kerkis font family, based on URW Bookman, with complete Greek support.

`lm` in `fonts/ps-type1`
: The massive Latin Modern font collection, in Type 1 format.

`metatype1` in `fonts/utilities`
: The tool used to build the excellent Latin Modern

fonts, among others. Has example source for an extended version of Knuth's `logo` font.

`MnSymbol` in `fonts`
Math symbol font for Adobe MinionPro.

`nkarta` in `fonts`
A corrected version of `karta`, containing map symbols.

`pandora` in `fonts/ps-type1`
Type 1 versions of the Pandora fonts.

`psethiop` in `fonts/ps-type1`
For typesetting Ethiopian languages.

`sanskrit` in `fonts/ps-type1`
Type 1 version of Charles Wikner's 'skt' font series for the Sanskrit language.

`sauter` in `fonts/cm`
An update to the Sauter parameter package for Computer Modern fonts.

`skaknew` in `fonts/chess`
For typesetting chess games.

`t1infos` in `fonts/utilities`
Two tiny tools for studying Type 1 fonts: `t1area` gives information about the black area of a glyph, and `t1extremes` tells if the "extremes" of Bezier curves are at the right place.

`tapir` in `fonts`
A simple geometrical font mostly created from line and circular segments with constant thickness.

`tipa` in `fonts`
An update of the TIPA typefaces.

`tt2001` in `fonts/ps-type1`
Type 1 EC fonts generated by TEXtrace.

graphics

`3DLDF` in `graphics`
Three-dimensional (batch) drawing program with METAPOST output.

`a2ping` in `graphics`
Unix command line utility written in Perl that converts many raster image and vector graphics formats to EPS, PDF, and other formats.

`epix` in `graphics`
Utility for mathematically accurate, camera quality plots and line figures.

`expressg` in `graphics/metapost/contrib/macros`
Facilities to assist in drawing diagrams that consist of boxes, lines, and annotations, such as IDEF or UML. Particular support is provided for creating EXPRESS-G diagrams.

`featpost` in `graphics/metapost/macros`
Three-dimensional drawing with METAPOST.

`metaplot` in `graphics`
Plot-manipulation macros for METAPOST.

`pst-3dplot` in `graphics/pstricks/contrib`
Plot 3D math functions.

`pst-bar` in `graphics/pstricks/contrib`
Produce bar charts.

`pst-fr3d` in `graphics/pstricks/contrib`
Draw 3D framed boxes.

`pst-func` in `graphics/pstricks/contrib`
Draw some special mathematical functions.

`pst-geo` in `graphics/pstricks/contrib`
Draw geographical projections.

`pst-infixplot` in `graphics/pstricks/contrib`
Macro commands for converting natural mathematical expressions to PostScript syntax. That is, allows PSTricks plotting with infix expressions rather than RPN.

`pst-jftree` in `graphics/pstricks/contrib`
Draw trees.

`pst-light3d` in `graphics/pstricks/contrib`
Draw a 3D shadowing effect on characters and PSTricks graphics.

`pst-math` in `graphics/pstricks/contrib`
Enhancement of PostScript math operators for use with PSTricks.

`pst-poly` in `graphics/pstricks/contrib`
Draw various polygons.

`pstricks-add` in `graphics/pstricks/contrib`
A collection of code from the pstricks mailing list.

`sparklines` in `graphics`
Sparklines are intense, simple, wordlike graphics. This tool allows drawing sparklines using the `pgf` package.

`texcad32` in `graphics`
TEXCad32 is a clone of the DOS program Texcad running under Windows.

`tpic2pdftex` in `graphics`
An `awk` bridge from `tpic` to PDFTEX.

help

`uk-tex-faq` in `help`
Major English-language FAQ, with information on virtually all TEX-related topics. Available on the web at `http://www.tex.ac.uk/faq`.

indexing

`cooridx` in `indexing`
Preprocessor for `MakeIndex` to sort chemical names in an index.

info

`beginlatex` in `info`
A thorough manual on getting started with LATEX — *Formatting Information: A Beginner's Guide to LATEX*, by Peter Flynn.

`chroma` in `info/colour`
A reference book of LATEX colors.

`fontinstallationguide.pdf` in `info/Type1fonts`
A comprehensive guide to installing Type 1 PostScript fonts.

`guia-atx` in `info/spanish`
A guide to writing LaTeX documents with Emacs and AucTeX.

`l2tabu` in `info`
Mark Trettin's guide to common problems with LaTeX. Originally in German with translations available in English, French and Italian.

`lshort` in `info`
"The Not Short Introduction to LaTeX 2_ε", by Tobias Oettiker. Available in Bulgarian, Dutch, Englsh, Finnish, French, German, Italian, Japanese, Korean, Mongolian, Polish, Portuguese, Russian, Spanish, Slovak, Thai, and Ukrainian.

`rgb` in `info/colour`
X11 color swatches.

`tex-references` in `info`
Reference information for TeX and friends.

`tlc2` in `info/examples`
Examples from *The LaTeX Companion, Second Edition*.

`ttb` in `info/biblio`
A manual about bibliographies, especially BibTeX.

`voss` in `info/math`
Articles on math-related topics by Herbert Voss, originally published in *Die TeXnische Komödie*.

language

`CBcoptic` in `language/coptic`
Typeset Coptic philological text with proper fonts and hyphenation.

`cjhebrew` in `language/hebrew`
Hebrew typesetting package including fonts.

`eehyph` in `language/hyphenation`
Estonian hyphenation patterns.

`elhyph` in `language/hyphenation`
Greek hyphenation patterns.

`gahyph` in `language/hyphenation`
Irish hyphenation patterns.

`ibycus-babel` in `language/greek/package-babel`
Allows usage of the Ibycus 4 font for ancient Greek with Babel.

`ithyph` in `language/hyphenation`
Italian hyphenation patterns.

`MNT` in `language/mongolian`
MnTTeX provides tools for typesetting *The Secret History of the Mongols*.

`pecha` in `language/tibetan`
Print Tibetan text in the classic 'pecha' layout style.

`ushyph` in `language/hyphenation`
Extended US English hyphenation patterns.

`velthuis` in `language/devanagari`
Velthuis Devanagari for TeX.

macros/context

`bib` in `macros/context/contrib`
ConTeXt bibliography module.

`t-amsl` in `macros/context/contrib/maths`
Provides some environments and commands that AMS-LaTeX users expect.

`t-nath` in `macros/context/contrib/maths`
Provides for ConTeXt the same functionality as the `nath` package for LaTeX.

macros/latex/contrib

`12many` in `macros/latex/contrib`
Provides generic way of writing "1, 2, many", as in $\{1, 2, \ldots, m\}$.

`anufinalexam` in `macros/latex/contrib`
LaTeX document shell to produce the standard formatting of final exams at The Australian National University.

`arcs` in `macros/latex/contrib`
Place an arc over or under a short piece of text.

`assignment` in `macros/latex/contrib`
Class for writing homework and lab assignments.

`beamer` in `macros/latex/contrib`
Create slides and presentations for a projector; now with much-improved support for verbatim code, and many other bug fixes.

`bigfoot` in `macros/latex/contrib`
Critical edition work in progress; for now, just the `suffix` package making it easy to define command variations like `\macro*` and `\macro\!`.

`caption` in `macros/latex/contrib`
Very general customization of captions in floating environments such as `figure` and `table`; cooperates with many other packages.

`clefval` in `macros/latex/contrib`
Defines macros `\TheKey` and `\TheValue` to support (the semblance of) hash tables.

`cmap` in `macros/latex/contrib`
Create PDF files with searchable and copyable text; now also works with the `CJK` package.

`contour` in `macros/latex/contrib`
Generates a colored contour around a given text, enabling printing text over a background without the need for a color box around the text. Vector outlines are now enabled when supported by the backend driver, such as with `dvips` and `pdftex`.

`cvsty` in `macros/latex/contrib`
Another style file for preparation of curricula vitæ.

`dramatist` in `macros/latex/contrib`
Typeset dramatic works (plays), either prose or verse, with support for 'dramatis personæ' lists, stage directions, and more.

`egameps` in `macros/latex/contrib`
Formatting extensive games, a kind of specification in game theory, using PSTricks.

`elmath` in `macros/latex/contrib`
Direct support for Greek characters on Greek keyboards.

`empheq` in `macros/latex/contrib`
A visual markup extension to `amsmath` for emphasizing equations, including extensible symbols, nondelimiter scaling, and column alignments.

`engrec` in `macros/latex/contrib`
Enumerate lowercase and uppercase Greek letters, with assorted variants.

`eskd` in `macros/latex/contrib`
Producing text documents in accordance with Russian (probably post-USSR) design standards.

`europcv` in `macros/latex/contrib`
Class for the standard model for curricula vitæ as recommended by the European Commission.

`exercise` in `macros/latex/contrib`
Helps in typesetting exercises and lists of exercises.

`extract` in `macros/latex/contrib`
Extract arbitrary content, possibly conditionally, from a source document and write it to a target document; for instance, extract exercises from lecture notes, or specific slides from a presentation. Also provides an environment for sharing code, such as a preamble, between the source and target files.

`figbib` in `macros/latex/contrib`
Supports organizing figures in BibTeX databases, for general List of Figures formatting, simplifying inclusion of figures, and more.

`floatrow` in `macros/latex/contrib`
Support for many float and caption layouts.

`fontspec` in `macros/latex/contrib`
Automatic and unified interface to feature-rich AAT and OpenType fonts in XeLaTeX.

`functan` in `macros/latex/contrib`
Macros for functional analysis, notably Sobolev spaces, and PDE theory.

`glossary` in `macros/latex/contrib`
Assist in generating a glossary with `makeindex`.

`ha-prosper` in `macros/latex/contrib`
Extends the `prosper` class for slide presentations with tables of contents, portrait slides, and more.

`hepparticles` in `macros/latex/contrib`
Typesetting high energy physics particle names in or out of math, including in section titles and other bold contexts.

`IEEEconf` in `macros/latex/contrib`
Format documents according to the IEEE Computer Society Press guidelines; this package replaces `latex8.sty` and `latex8.bst`.

`juraabbrev` in `macros/latex/contrib`
Handling abbreviations in German law, including making a list of those actually used.

`juramisc` in `macros/latex/contrib`
Collection of packages and classes for typesetting German juridical documents.

`jurarsp` in `macros/latex/contrib`
BibTeX style for citations of judgements and other official documents in German law.

`kerntest` in `macros/latex/contrib`
Print tables and generate `mtx` files to help in adjusting font kerning tables.

`koma-script` in `macros/latex/contrib`
A versatile bundle of document classes and packages, aiming to be a replacement for the standard LaTeX 2_ε classes.

`layaureo` in `macros/latex/contrib`
Support wide page layouts for documents using the A4 paper size, including binding offsets.

`ledmac` in `macros/latex/contrib`
Typeset critical editions; a LaTeX port, and extension, of the plain `edmac`, `tabmac`, and `edstanza` macros.

`ledpar` in `macros/latex/contrib`
Extension of `ledmac` enabling parallel typesetting, in columns or on facing pages.

`logpap` in `macros/latex/contrib`
Draws logarithmic/linear graph paper, in all combinations.

`ltabptch` in `macros/latex/contrib`
Fixes bugs in `longtable.sty`.

`ltxindex` in `macros/latex/contrib`
Making indices in LaTeX using GNU `texindex` instead of `makeindex`.

`makebox` in `macros/latex/contrib`
Defines a `\makebox*` command, same as `\makebox` but with the width given by a sample text instead of an explicit length.

`maybemath` in `macros/latex/contrib`
Provides math commands `\maybebm` and `\maybeit` which typeset their arguments in bold or italic, respectively, if the surrounding context is appropriate, such as a section title.

`memoir` in `macros/latex/contrib`
Peter Wilson's flexible LaTeX class for typesetting general fiction, non-fiction, and mathematical works; support for customized designs, trim marks, various document sizes, and much more.

`mentis` in `macros/latex/contrib`
Adjustment for publishing at Mentis Publishers, Paderborn, Germany.

`mhchem` in `macros/latex/contrib`
Support for typesetting chemical molecular formulae, and chemical equations with these formulae. Also includes the `rsphrase` package with the text, in English and German, of all official Risk and Safety Phrases used to label chemicals.

`microtype` in `macros/latex/contrib`
LaTeX interface to the pdfTeX micro-typographic features: character protrusion and font expansion.

`movie15` in `macros/latex/contrib`
Package for multimedia inclusion, for use with PDF version 1.5.

`msg` in `macros/latex/contrib`
Aims to localize any document class or package, so

that messages may be reported in the end-user's preferred language.

`nature` in `macros/latex/contrib`
Unofficial support for preparing articles and letters to the journal *Nature*.

`ncctools` in `macros/latex/contrib`
Many packages for general LaTeX use, including cropmarks, watermarks, hyphenation of compound words, poor man's blackboard bold, and more.

`ofs` in `macros/latex/contrib`
Olšak's Font System, containing plain and LaTeX macros for managing large font collections, including support for many font encodings.

`osa` in `macros/latex/contrib`
Latest LaTeX, REVTeX, and BibTeX tools for journals of the Optical Society of America.

`pagenote` in `macros/latex/contrib`
Supports tagged notes on a separate page, a.k.a. end notes.

`paresse` in `macros/latex/contrib`
Abbreviations for typesetting of Greek letters in math mode, using a new active character.

`pbsheet` in `macros/latex/contrib`
Typesetting of problem sheets including mathematics, programs, and graphics.

`pclnfss` in `macros/latex/contrib`
Support for selecting and using the standard 45 scalable fonts built into most PCL laser printers.

`perltex` in `macros/latex/contrib`
Allows defining LaTeX macros with Perl code, thus combining LaTeX's typesetting power with Perl's programmability.

`pict2e` in `macros/latex/contrib`
A picture-drawing package, described in the second edition of *LaTeX: A Document Preparation System*.

`pittetd` in `macros/latex/contrib`
A class that formats documents for submission as electronic theses and dissertations (ETD) to the University of Pittsburgh, including bookmarking. It also has some generic features potentially useful for ETD classes at other institutions.

`probsoln` in `macros/latex/contrib`
Generate new problem sheets, and answers, including random selection of problems.

`ptptex` in `macros/latex/contrib`
Official class files for the journal *Progress of Theoretical Physics*.

`rotfloat` in `macros/latex/contrib`
Bridges the `sidewaysfigure`, `sidewaystable` and `float` packages to allow floats that are both rotated and (for example) ruled or boxed.

`rrgtrees` in `macros/latex/contrib`
Producing linguistic tree diagrams suitable for Role and Reference Grammar (RRG); allows the construction of trees with crossing lines, as required by this theory for many languages.

`sauerj` in `macros/latex/contrib`
Miscellaneous styles by Jonathan Sauer, including: `optparams` supports creating macros with multiple optional parameters; `parcolumns` supports typesetting in two or more columns in parallel; `processkv` supports calling a user-defined macro for each key/value pair in a list; `zahl2string` formats numbers as German words.

`scientificpaper` in `macros/latex/contrib`
A simple, generic scientific paper format.

`sgame` in `macros/latex/contrib`
Formats strategic games, a game theory specification.

`SIstyle` in `macros/latex/contrib`
Typesets physical units following the rules of the International System of Units (SI).

`splitbib` in `macros/latex/contrib`
Split a bibliography into categories and subcategories; does not depend on BibTeX.

`stdpage` in `macros/latex/contrib`
Produce standard pages of n lines with at most m characters each, for translations, proofreading, etc.

`struktex` in `macros/latex/contrib`
Support for Nassi Shneidermann structure charts in algorithm development.

`subfig` in `macros/latex/contrib`
Almost-compatible replacement for `subfigure`, using the `caption` package and with a keyword/value interface.

`switchcml` in `macros/latex/contrib`
Obfuscates an email address so that it prints correctly but cannot be harvested.

`umich-thesis` in `macros/latex/contrib`
Produce a University of Michigan dissertation according to the Rackham dissertation handbook.

`underbracket` in `macros/latex/contrib`
Draws brackets to underline text, especially but not exclusively with `musictex` and `musixlyr`.

`wallpaper` in `macros/latex/contrib`
Easy addition of background images to LaTeX documents, including tiling.

`xarrow` in `macros/latex/contrib`
Extensive arrows: `\xlongequal`, `\xleftrightarrow`, `\xlongleftrightarrow`, `\xlongleftarrow`, `\xlongrightarrow`, and `\xLong` variants for all.

`xcolor` in `macros/latex/contrib`
Driver-independent color extensions, including shading, color masking, color separation, and conversion between color models such as RGB and CMYK.

`xkeyval` in `macros/latex/contrib`
A notable extension of the `keyval` package.

macros/latex/exptl

`mem` in `macros/latex/exptl`
An experimental environment for multilingual and

multiscript typesetting with LaTeX in the Aleph typesetting system.

`xfrac` in `macros/latex/exptl`
Produce visually pleasing split level fractions for arbitrary fonts.

support

`autoconf` in `support`
Autoconf macros to test for the presence of LaTeX.

`bibex` in `support`
Automates the extraction of bibliographic references from BibTeX databases.

`bmeps` in `support`
A program to convert from PNG, TIFF, JPEG, and NetPBM to EPS.

`chktex` in `support`
Finds typographic errors in LaTeX.

`easylatex` in `support`
Turns "ASCII math" into LaTeX source.

`eukleides` in `support`
A Euclidean geometry drawing language.

`gellmu` in `support`
GELLMU is an acronym for "Generalized Extensible LaTeX-Like MarkUp", which is the author's concept for using LaTeX-like markup to write consciously for SGML document types such as HTML, DocBook, TEI, or GELLMU's own didactic LaTeX-like article format.

`latexdiff` in `support`
A Perl script for finding the differences between two LaTeX files as another LaTeX file, with various output format options.

`latexrender` in `support`
Use LaTeX in PHP programs.

`ldiff` in `support`
A Python script for reporting the differences between two LaTeX files, as a PostScript document.

`maketable` in `support`
Convert Word or Excel tables to TeX tabular structures.

`mimetex` in `support`
Parses well-formed LaTeX math expressions, emitting either GIF images or MIME xbitmaps.

`orderrefs` in `support`
Reorder the bibliography in a LaTeX document by order of citation.

`pdfcrop` in `support`
Takes a PDF as input, calculates the BoundingBox for each page with the help of Ghostscript and generates an output PDF without margins.

`png2pdf` in `support`
Convert PNG images to PDF.

`preview-latex` in `support`
A system for displaying inline images of selected parts of a file in Emacs source buffers. The style file is independently useful for extraction of selected text elements as images.

`pydocstrip` in `support`
An alternative to TeX `docstrip`.

`references` in `support`
Bibliographic software for authors of scientific manuscripts and for management of bibliographic data of journal articles, books, book chapters, etc.

`shortcuttool` in `support`
Enables file import to the input tool Shortcut and provides a shortcut file.

`tex4ht` in `support`
A complete system for translating (La)TeX and ConTeXt sources into HTML, XML, MathML, etc.

`texconverter` in `support`
Windows front-end to various LaTeX to HTML converters.

`tif2eps` in `support/pstools`
Convert TIFF images to EPS.

`vpp` in `support/viewprintpspdf`
A command line utility to view and print PostScript and PDF documents.

systems

`epmtfe` in `systems/os2`
The "EPM TeX Front End", a module for the OS/2 "Enhanced Editor" EPM. It turns the EPM into an integrated TeX environment, providing (La)TeXing, previewing and executing of auxiliary programs from the editor menu.

`latexpix` in `systems/win32`
A drawing program for Windows which generates LaTeX pictures.

`oztex` in `systems/mac`
OzTeX is a standalone Mac implementation of TeX; also can be used as a front-end to teTeX on OS X.

`pdftex` in `systems`
An extension of TeX that can create PDF directly from TeX source files. It also contains many new features and extensions to TeX.

`TeXmacs` in `systems/unix`
GNU TeXmacs is a free scientific text editor, which was inspired by both TeX and GNU Emacs.

`WinShell` in `systems/win32`
A graphical user interface for easily working with TeX. It is *not* a TeX system itself, so requires a system such as MikTeX or TeX Live.

`vtex` in `systems`
VTeX/Free for OS/2 and Linux (x86).

Abstracts

Editor's note: This issue of *TUGboat* contains abstracts and summaries from recent publications by several other TEX user groups, translated to English where needed. For a complete list of all user group publications, see `http://tug.org/pubs.html`.

***Zpravodaj* 13(1)–14(2), 2003–2004**

Zpravodaj is the bulletin of CSTUG, the TEX user group for the Czech and Slovak languages. Their web site is `http://www.cstug.cz`, and the *Zpravodaj* web site is `http://bulletin.cstug.cz`.

***Zpravodaj* 13(1), 2003**

Petr Olšák, Úvodníček [Introduction]; p. 1–2

Jiří Kosek, Sazba XML [Typesetting XML]; p. 3–6

This article summarizes methods suitable for processing XML documents by the TEX system — direct typesetting (xmltex, ConTEXt), conversion to TEX (XSLT) and TEX based stylesheet language implementations (XSL, DSSSL). The article acts as an introduction for more detailed articles about processing XML with TEX.

Jiří Kosek, Použití parseru XML v TEXu [Use of an XML parser in TEX]; p. 6–14

This article shows how to use xmltex — an XML parser written in pure TEX — to directly typeset XML documents. Special interest is devoted to correct processing of localized Czech/Slovak documents.

Jiří Kosek, JadeTEX; p. 15–26

JadeTEX is a TEX macro package which is able to process SGML and XML documents according to a DSSSL stylesheet in conjunction with (Open)Jade DSSSL processor. This article briefly describes basic principles of the DSSSL language and its usage for formatting XML documents. Complete working example of a DSSSL stylesheet is shown in the article.

Jiří Kosek, PassiveTEX; p. 26–38

PassiveTEX is a TEX-based XSL-FO processor which is able to process XML documents according to an XSL stylesheet in conjunction with any XSLT processor. This article briefly describes basic principles of the XSL language and its usage for formatting XML documents. Complete working example of an XSL stylesheet is shown in the article.

Zdeněk Wagner, Fraktální obrazce v PostScriptu [Fractal Images in PostScript]; p. 45–53

The picture used on the cover of this issue is an example of a fractal image. The article describes the PostScript macro by means of which the picture was created.

***Zpravodaj* 14(1), 2004**

Petr Olšák, Úvodníček [Introduction]; p. 1–2

Zdeněk Wagner, Anatomie virtuálních fontů [Anatomy of Virtual Fonts]; p. 3–16

The article is a brief introduction to the concept of virtual fonts. It is first explained how TEX works with fonts. Afterwards a simple tool for building a virtual font, namely qdTEXvpl, is presented. Finally usage of virtual fonts is demonstrated by typesetting spaced and underlined text. The macros and Perl scripts described in this article are available from the web page of the Bulletin.

Aleš Pavelka, Wordové plug-iny související s TEXem aneb Možnosti a schopnosti produktů Word2TEX a TEX2Word [Word Plug-Ins for TEX: Possibilities and Abilities of the Word2TEX and TEX2Word Products]; p. 16–28

The article describes two MS Word plug-ins which allow conversion from and to TEX. The documents illustrating the results of conversion are available from the web page of the Bulletin.

Ladislav Bittó, TEX and PostScript in Graphics of Programming Languages; p. 28–38

The article describes possibility of generating PostScript graphics by means of a library of subroutines written in FORTRAN. The speed of the program is compared to that of METAPOST. Examples of pictures created by the mentioned program are available from the web page of the Bulletin.

***Zpravodaj* 14(2), 2004**

Petr Olšák, Úvodníček [Introduction]; p. 45–46

Vít Zýka, Používáme pdfTEX IV: mikrotypografické rozšíření [Using pdfTEX IV: micro-typographic extensions]; p. 47–53

This article describes two micro-typographic extensions being implemented by Hàn Thế Thành in pdfTEX: character protruding and font expansion. Expanded font metric preparation is also addressed.

Miroslav Balda, Výpočty a diagramy v LATEXu [Calculations and diagrams in LATEX]; p. 54–110

The article deals with the title problem from the point of view of a common user of LATEX. It describes a way of using the standard packages `fp.sty`

and `curves.sty`, along with their new extensions `fp-contrib.sty` and `diagram.sty` with an auxiliary package `support.sty`. The suite allows solving rather complicated tasks in one run of the LaTeX compiler. A solution for processing fatigue data into SN-curve, bands of confidence intervals, plots and a table of results is presented as an example. The system is also suitable for presentation purposes.

[Received from Zdeněk Wagner]

Die TEXnische Komödie
Contents of Issues 1–4/2003

Die TEXnische Komödie (DTK) is the publication of DANTE, the German language TEX user group. Their web site is `http://www.dante.de`.

15. Jahrgang, Heft 1/2003 (Februar 2003)

15. Jahrgang, Heft 2/2003 (Mai 2003)

15. Jahrgang, Heft 3/2003 (September 2003)

◦ *Leserbrief(e)* [Letters]; pp. 64–66:

Moriz Hoffmann-Axthelm, Zu Torsten Brongers Artikel "Einfaches Setzen von Texten in Fraktur mittels `blackletter1`" [On Torsten Brongers' article "Simple typesetting of texts in Gothic using `blackletter1`"]; pp. 64–66

◦ *Spielplan* [Repertory]; pp. 67–70:
The international and national calendar.

◦ *Adressen* [Addresses]; pp. 70–72:

15. Jahrgang, Heft 4/2003 (November 2003)

Gerd Neugebauer, [Editorial]; p. 1

◦ *Hinter der Bühne : Vereinsinternes* [Backstage : Club matters]; pp. 4–31:

Volker RW Schaa and Klaus Höppner, Grußwort [Introduction]; pp. 4–7

Günter Partosch, Protokoll der 29. Mitgliederversammlung von DANTE e.V. am 9. September 2003 in Rauischholzhausen [Program of the 29th general meeting of DANTE e.V. on 9 September 2003 in Rauischholzhausen]; pp. 5–12

Sebastian Waschik, Bericht von der Herbsttagung von DANTE e.V. [Report of the Fall meeting of DANTE e.V.]; pp. 12–14

Thomas Lotze, [EuroTEX 2003 in Brest/ Bretagne]; pp. 15–23

Blandyna Bogdol, LaTeX ist auch weiblich [LaTeX is for women too]; pp. 24–27

Volker RW Schaa, TEX Collection: Fehler und Updates [TEX Collection: Errors and updates]; pp. 27–28

Volker RW Schaa, Danksagung [Thanks]; pp. 29–30

Holger Grothe and Volker RW Schaa, Einladung zur TEX-Tagung DANTE 2004 in Darmstadt — 15 Jahre DANTE e.V. [Announcement of DANTE 2004 in Darmstadt — 15 years of DANTE e.V.]; pp. 30–31

◦ *Bretter, die die Welt bedeuten* [The stage is the world]; pp. 32–65:

David Kastrup and Markus Kohm and Torsten Krüger and Michael Niedermair and Rolf Niepraschk, $\epsilon\chi$TEX — ein Überblick [$\epsilon\chi$TEX — an Overview]; pp. 32–38

In December 2002 a small group of developers met to develop TEX further, based upon NTS. Consisting at the beginning of only vague ideas, the studying of the sources of ϵ-TEX, pdfTEX and Ω led to the decision to develop a new system written in Java — $\epsilon\chi$TEX. This article is about the current status of the project.

Harald Harders, Mehrsprachige Literaturverzeichnisse: Anwendung und Erweiterung des Pakets `babelbib` [Multilingual bibliographies: Application and enhancement of `babelbib`]; pp. 39–63

The package `babelbib` provides two extensions over most existing BibTEX styles: it is possible to change the keywords according to the language, and it is possible to change some typographical elements in the bibliography without changing the whole BibTEX style.

Rolf Niepraschk, Tipps und Tricks: Eine `minipage`, die mitdenkt [Tips and Tricks: A 'thinking' `minipage`]; pp. 63–65

Usually one has to know the width of a `minipage` in advance to define it. The article describes the package `varwidth` which circumvents this.

◦ *Rezensionen* [Reviews]; pp. 66–68:

Carsten Heinisch, TeX2Word und Word2TeX [TeX2Word and Word2TeX]; pp. 66–68

Review of two file conversion tools from Chikrii Softlab.

◦ *Von fremden Bühnen* [On other stages]; pp. 69–70:

Martin Schröder, Der `\year=2004` TEX Kalendar [The `\year=2004` TEX Calendar]; pp. 69–70

◦ *Spielplan* [Repertory]; pp. 71–74:
The international and national calendar.

◦ *Adressen* [Addresses]; pp. 74–75:

(Compiled by Wolfgang Huber and Barbara Beeton)

Biuletyn GUST 20–21, 2004

Biuletyn GUST is the publication of GUST, the Polish language TEX user group. The group's web site is `http://www.gust.org.pl`.

Biuletyn GUST 20, 2004

John Plaice and Paul Swoboda, Moving Omega to a C++-based Platform; pp. 3–5

The code for the Omega Typesetting System has been substantially reorganised. All fixed-size arrays implemented in Pascal Web have been replaced with interfaces to extensible C++ classes. The code for interaction with fonts and Omega Translation Processes (OTPs) has been completely rewritten and placed in C++ libraries, whose methods are called by the typesetting engine. The Pascal Web part of Omega no longer uses change files. The overall Omega architecture is now much cleaner than that of previous versions.

Marcin Woliński, I my tak składamy? Rzecz o parametrze `topskip` [So we do typeset like this? The case of `topskip`]; pp. 6–8

When using the default Plain TEX value of the `topskip` parameter, upper edges of some columns can look unaligned. In this paper the problem is illustrated and proposals for selecting other values for `topskip` are given.

Janusz S. Bień, Standard Unicode 4.0. Wybrane pojęcia i terminy [Unicode 4.0 — basic notions and terminology]; pp. 9–14

Selected features of Unicode are presented and the standard is compared with earlier text encoding approaches. The paper contains proposals for Polish translations of the original English language terms used in the Unicode standard.

David Kastrup, The `bigfoot` bundle for critical editions; pp. 15–20

The LATEX package `bigfoot` and supporting packages solve many of today's problems occurring in the contexts of single and multiple blocks of footnotes, and more. The main application is with philological works and publications, but simpler problems can be solved painlessly as well without exercising all of the package's complexities. For other problems not yet tackled in this area, a solid framework is provided.

Jean-Michel Hufflen, A Tour around MlBibTEX and Its Implementations(s); pp. 21–28

This article describes the components of mlBibTEX, a new implementation of BibTEX including multilingual features. We justify our choices and show why our use of XML eases most operations performed by MlBibTEX. Also, there are two implementations of MlBibTEX, a prototype developed in Scheme, and a more robust program in C. We also explain how we take advantage of this approach.

Janusz M. Nowacki, Antykwa Toruńska wersja 2.0 [The new embodiment of Antykwa Toruñska]; pp. 29–33

The paper features extended version of the Antykwa Torunska family of fonts.

Szymon Zioło, Cocoon – środowisko publikacyjne oparte na XMLu [Cocoon — an XML based publishing environment]; pp. 34–38

Cocoon is an XML-based, open source application for developing WWW sites and other web applications. It uses a clever transformation mechanism, which enables separation of graphical layout design tasks from site structure and information management.

Radosław Tryc, SVG z TEX-em [SVG for TEX]; pp. 39–43

SVG is a publicly available, well documented and easily extensible format used in the Internet and multimedia. It is argued that SVG is useful for TEX users as well. In the paper selected tools for producing and processing SVG graphics are presented (Sodipodi, Scribus, Apache Batik).

Włodzimierz Bzyl and Tomasz Przechlewski, Wykorzystanie TEX4ht i XSLT do konwersji plików LATEXa [LATEX-to-XML conversion with `tex4ht` and XSLT]; pp. 44–47

The TEX4ht system is generally considered to be the best application for converting TEX files to HTML/XML format. TEXht consists of three parts: style files which enhance existing macros with HTML, or DocBook, or TEI like features; the `tex4ht` processor which extracts HTML (or DocBook/TEI) files from DVI files produced by TEX; and the `t4ht` processor which is responsible for translating DVI code fragments which need to be converted to pictures; for this task, the processor uses tools available on the current platform. Out of the box, TEX4ht is configured to translate roughly from `plain`, LATEX, `ltugboat`, `ltugproc`, Lecture Notes in Computer Science (`llncs`) formats to HTML/XML. However, the conversion from a visual format to information oriented one cannot be done automatically; usually prior configuration of TEX4ht is needed. Instead of configuring TEX4ht — which is not easy — we could use an XSLT style-sheet to remap elements to reference XML format. The paper introduces the TEX4ht

system. Selected problems of configuring the system and converting TEX/LATEX files to XML with TEX4ht are discussed.

Halina Wątróbska and Ryszard Kubiak, Wykorzystanie Emacsa, Haskella i TEXa w pracach nad słownikiem języka staro-cerkiewno-słowiańskiego [Emacs, Haskell and TEX; cooperating on an Old-Church to Slavonic dictionary]; pp. 48–53

The paper describes how TEX, the Haskell programming language and the Emacs editor are used for authoring of the Old-Church Slavonic to Polish dictionary in the Slavic Department at the Gdańsk University.

Tomáš Hála, The Implementation of Nested Quotation Marks; pp. 54–56

In a lot of languages, quotation marks are set using characters. In some styles, e.g. `czech.sty` and `slovak.sty`, a special macro command is used. However, none of these methods allow for correct typesetting of nested quotation marks. This contribution describes a solution to this problem in LATEX. A set of macros in a special TEX-style has been composed and settings for various languages have been created. The presented solution is user-friendly and general. In addition, the standard settings can be configured by the user.

Biuletyn GUST 21, 2004

Jerzy Ludwichowski, Cicer cum caule – aktualności stare i nowe [Cicer cum caule — old and fresh news]; pp. 3–4

A chronicle of the most important, recent, TEX-related events and achievements.

Antonina Liedtke, By kod giętki wyraził, co wymyśli głowa [So that the code expresses all the mind invents]; pp. 5–13

Using LATEX in a publishing house differs from personal usage. Some packages are used infrequently, e.g., BIBTEX is not required if bibliographies are included in the text of the manuscripts. On the other hand some packages are vital for pre-press (for example the Crop package) but rather unnecessary for authoring. For publishers the typographical correctness, i.e. conformity to publishing standards is of great importance. The paper deals with the graphical layout design, an issue important for publishers. It is argued that designing graphically appealing documents in LATEX is not only possible but also easy — very often it is sufficient to include some two or three additional packages. This is demonstrated with code examples for designing chosen graphical layouts originating from real books.

Jean-Michel Hufflen, Making mlBIBTEX Fit for a Particular Language. Example of the Polish Language; pp. 14–26

The mlBIBTEX project aims to provide a multilingual bibliography program. In this article, we show how to make mlBIBTEX's Version 1.3 fit for a particular language. In particular, we explain how bibliographical keywords such as 'and', 'chapter', ... should be defined in this particular language. We also show how to refine bibliography styles. For the BachoTEX conference, we chose the Polish language; nevertheless, reading this paper should be useful for anyone who would like to adapt mlBIBTEX.

Jacek Kmiecik and Marek A. Walenta, O przetwarzaniu dużych dokumentów – duże też może być piękne... [Processing large documents — big can be beautiful...]; pp. 27–30

The main task of the BPP AGH (Bibliographic List of Staff Publications) application is accumulating, processing and giving on-line accessibility to all kinds of data relating to the publications authored by the staff of the AGH University of Science and Technology. The project is based on open software: Linux, Apache, PHP, MySQL and TEX. Processing of the database content into a PDF file is done with ConTEXt.

Robin Fairbairns and Jim Hefferon and Rainer Schöpf and Joachim Schrod and Graham Williams and Reinhard Zierke, CTAN – plany [CTAN plans]; pp. 31–34

The readers of TUGboat likely know the Comprehensive TEX Archive Network as a great pile of TEX stuff. That is, it is full of TEX materials and it is great, but it is also perhaps a pile — a bit of a mess. We will sketch some plans for improving CTAN. As part of that, we will outline its architecture, history, and some issues.

Tomasz Łuczak, Małe marzenie [A small dream]; pp. 35–36

A short description of SlaX-TL, a CD bootable, TEX dedicated Linux distribution based on SlaX.

Stanisław Wawrykiewicz, TEX Live 2004; pp. 37–39

A short introduction to the forthcoming TEX Live 2004 distribution.

Andrzej Borzyszkowski, 14th European TEX Conference, 24–27 czerwca 2003, Brest [14th

European TEX Conference, 24–27 June 2003, Brest]; p. 40

A report from the 14th European TEX conference in Brest, France.

Tomasz Przechlewski, XII Ogólnopolska Konferencja TEX-owa, 30.04.–2.05. 2004, Bachotek [XII annual GUST TEX conference, 4/30–5/2, 2004, Bachotek]; pp. 40–42

A report from BachoTEX 2004, the 12th annual GUST TEX conference.

Włodzimierz Bzyl and Tomasz Przechlewski, Konferencja TUG 2004, Xanthi, Grecja [TUG 2004 conference, Xanthi, Greece]; pp. 42–44

A report from the TUG 2004 TEX conference in Xanthi, Greece.

Bogusław Jackowski and Jerzy Ludwichowski, Konferencja TUG 2003, Waikoloa Beach Resort, Big Island, Hawaje, USA [TUG 2003 conference, Waikoloa Beach Resort, Big Island, Hawai'i, USA]; pp. 45–52

A report from the TUG 2003 conference, Waikoloa Beach Resort, Big Island, Hawai'i, USA.

[Received from Tomasz Przechlewski]

***Les Cahiers GUTenberg* 43, December 2003**

Les Cahiers GUTenberg is the publication of the French language TEX user group, GUTenberg. Their web site is `http://www.gutenberg.eu.org`, and articles from *Cahiers* issues are available at `http://www.gutenberg.eu.org/publications`.

Jacques André, Éditorial: un siècle et demi d'imprimerie [Editorial: A century and a half of printing]; pp. 3–4

The editor begins by pointing out that, while for many TEX users document composition ends with the file output either displayed on a screen or printed on paper, in the commercial world, such files still have a long life, with many additional applications. And, going in the other direction, before such output options as laser printers and monitor screens, older technologies were the norm, all the way back to shaping lead to form letters.

This history of printing technology over the past 150 years forms the basis of this issue, with the articles presented in reverse chronological order, moving from things which some of us are quite familar with, to those which predate personal experience.

Maurice Laugier (the current GUTenberg president) writes about his career at Louis Jean, a company which has, over the past half century, become a large producer of French scientific publications, especially mathematics. His article, spanning the interval between lead and laser, is as much about the company history of composition equipment as about methods developed there. Three words sum it up: quality, cost, speed of production.

These three words also appear in Éric Le Ray's article on Marinoni, who was in some ways the French equivalent of Citizen Kane, and an inventor of numerous machines. The article itself is less in the *Cahier* style, and more in that of the humanities, with lots of dense paragraphs, figures of different sizes, many footnotes (some lengthy). At some 60 pages, its very production was a test of TEX's abilities to pleasantly surprise in the face of such variability.

[Translated summary of editor's text]

Maurice Laugier, La composition des mathématiques. Évolution des techniques au travers d'une expérience professionnelle [Math composition: The evolution of techniques during a professional career]; pp. 5–32

During the past forty years, composition and printing techniques have undergone significant upheavals which have entailed conversions and the questioning of traditions. The composition of mathematical texts has naturally been deeply affected by these changes.

Techniques have changed, but if typographic knowledge is an art which evolves, its fundamentals still have a reason for existing. The confusion between the tools and the knowledge to use those tools has often led to results detrimental to the entire endeavour, leading to an overall reduction in quality.

[Translated extracts of author's abstract]

Éric Le Ray, Histoire de l'imprimerie et de la presse, en marge d'un centenaire: Hippolyte-Auguste Marinoni (1823–1904) [History of Printing and the Press, on the eve of a centennial: Hippolyte-Auguste Marinoni (1823–1904)]; pp. 33–99

Marinoni was born in Paris in 1823, orphaned early on in life, becoming an apprentice at the age of 12. In 1837 he earned his machinist's certificate and began working on typography equipment the next year — his career yielded many advances in the field of printing. In 1882 he became head of *Le Petit Journal*, which made it possible for this "Napoleon of the Press" to become an influential figure in the world of information and the press. He died of tuberculosis in 1904. As for the Marinoni Company, it underwent various changes in ownership, eventually becoming Heidelberg Web Systems in 1999.

[Translated extracts of author's abstract]

[Compiled by Christina Thiele]

MAPS 29–31, 2003–2004

MAPS is the publication of NTG, the Dutch language TEX user group. Their web site is `http://www.ntg.nl`.

MAPS 29, Spring 2003

Philipp Lehman, The font installation guide; pp. 65–160

This guide is an unmodified printout of Philip Lehman's original guide, which is available from CTAN. [Editor's abstract]

MAPS 30, Spring 2004

Wybo Dekker, Redactioneel [From the editor]; p. 1

Overview of the issue's contents.

Siep Kroonenberg, The MAPS style; pp. 2–4

This paper introduces the renewed MAPS classfile and includes some usage notes. [Author's abstract]

Piet van Oostrum, Een uittreksel uit de recente bijdragen in het CTAN archief [Extracts from recent contributions to the CTAN archive]; pp. 5–7

This article describes a number of recent contributions to the CTAN archive. The selection is based on what I find interesting and what I think others will find interesting. It is thus a personal choice. There is no intention of giving a complete overview. Consider this a kind of menu to whet the appetite of the curious. [Translation of author's abstract]

Siep Kroonenberg, Schatgraven op TEX Live [TEX Live treasure chest]; pp. 8–9

This piece brings to the attention of the reader the rich contents of the TEX Live CD. [Translation of author's abstract]

Hans Hagen, TEX Live Collection; pp. 10–12

Past and future of the TEX Live Collection is described. [Author's abstract]

Taco Hoekwater, De CXTEX distributie [The CXTEX distribution]; pp. 13–20

The aim of the CXTEX project is to be able to execute a complete texexec call from beginning to end within a single, as efficient as possible, system process. The first components of this distribution are presented in this article: traditional as well as C-language versions of texexec, texutil and pdfetex. [Translation of author's abstract]

Hans Hagen, The SciTE-TEX integration; pp. 21–24

Text editors are a sensitive, often emotional subject. Some editors have exactly the properties a software designer or a writer desires and one gets attached to it. Still, most computer experts such as TEX users often use three or more different editors each day. SciTE is a modern programmer's editor which is very flexible, very configurable, and easily extended. We integrated SciTE with TEX, ConTEXt, LATEX, METAPOST and viewers and succeeded, in that it is now possible to design and write your texts, manuscripts, reports, manuals and books with the SciTE editor without having to leave the editor to compile and view your work. The article describes what is available and what you need with special emphasis on highlighting commands with lexers. [Author's abstract (edited)]

Wybo Dekker, Introducing oldstyle figures in existing virtual fonts; pp. 25–32

This paper describes a *Ruby* script `osf` that can be used to make a copy of a virtual font with its figures replaced with old style figures. [Author's abstract]

Adam T. Lindsay, Apple symbols; pp. 33–41

This Mac-specific article documents some fonts available exclusively on Mac OS X 10.3, 'Panther', and makes them available to Mac users with fairly minimal installation effort. I do not distribute the fonts themselves. [Author's abstract (edited)]

Adam T. Lindsay, Unicode symbols; pp. 42–48

The Unicode standard includes a number of signs, symbols, dingbats, bullets, arrows, graphical elements, and other miscellaneous glyphs. Prompted by finding a font dedicated to many such Unicode symbols on Mac OS X systems, this article documents some ways of enabling these symbols on your own system. [Author's abstract (edited)]

Wybo Dekker, Woordafbreking op ë en ï [Hyphenation at ë and ï]; p. 49

LATEX has issues hyphenating words that contain ë. This article shows how to solve that problem: use `\"e` or `\"{e}` instead of ë for a unitary ë, and `"e` for all others. Analogously for ï. [Translation of author's abstract]

R. F. Smith, LATEX uitvoer genereren vanuit C programma's [Generating LATEX output from C programs]; pp. 50–51

This article describes a simple way to generate LATEX output from C programs. [Author's abstract]

Willi Egger, Help! — the typesetting area; pp. 52–59

Typesetting (large) documents presents significant challenges that have to be resolved before a satisfactory printed result is achieved; e.g. the internal structure of the document should be clear, and the document's typographical layout should match its content. This article, based on a presentation

given at the NTG meeting in Arnhem on 13 November 2003, describes a traditional design technique known as the harmonic proportion.
[Author's abstract (edited)]

Siep Kroonenberg, TEX and prepress; pp. 60–65
This article discusses preparing documents for professional printing with TEX and pdfTEX, including color printing and prepress standards.
[Author's abstract]

Piet van Oostrum, Een tutorial over het gebruik van BibTEX [A tutorial on the use of BibTEX]; pp. 66–86
This article describes the use of BibTEX, with particular emphasis on aspects that present problems to inexperienced users. It is based on a presentation the author gave at the NTG meeting in Arnhem on 13 November 2003.
[Translation of author's abstract]

Siep Kroonenberg, De TEX flyer: doe er wat mee! [The TEX flyer: Do something with me!]; pp. 87–89
On the following two pages we present once more our printed TEX flyer. The front describes the strong points of TEX, and the back contains all the necessary information to give people a quick introduction to TEX.
[Translation of author's abstract]

MAPS Editors, Foto's van de NTG-dag [Photos from the NTG meeting]; pp. 90–91

MAPS 31, Fall 2004

Wybo Dekker, Redactioneel [From the editor]; p. 1
Overview of the issue's contents.

Piet van Oostrum, Een uittreksel uit de recente bijdragen in het CTAN archief [Extracts from recent contributions in the CTAN archive]; pp. 2–4
This article describes a number of recent contributions to the CTAN archive. The selection is based on what I find interesting and what I think others will find interesting. It is thus a personal choice. There is no intention of giving a complete overview. Consider this a kind of menu to whet the appetite of the curious.
[Translation of author's abstract]

Hans Hagen, The state of ConTEXt; pp. 5–7
In this article I will describe the current state of the ConTEXt macro package and the forces that play a role in its evolution. I will also indicate the directions in which we look for further developments.
[Author's abstract]

Taco Hoekwater, METAPOST developments; p. 8
This item on the current status of METAPOST was reprinted in *TUGboat* 25(1), p. 105.

Giuseppe Bilotta, The Aleph project; pp. 9–11
A brief introduction to the Aleph project, a TEX extension providing most of Omega and ε-TEX features.
[Author's abstract]

Maarten Sneep, Producing graphs with METAPOST; pp. 12–18
Karel Wesseling described in MAPS 29 how several METAPOST graphs can be aligned relative to each other, by including them in a ConTEXt command `\startcombination[1*2]`. Here I describe a different approach to the same problem: aligning multiple graphs in a single figure. As a bonus, a description is added on how to create error-bars in a METAPOST generated graph.
[Author's abstract (edited)]

Dwight Aplevich, Circuit_macros; pp. 19–24
The evolution of the Circuit_macros package is described, with some of the conventions for drawing circuit elements and some of the lessons learned.
[Author's abstract]

Frans Goddijn, Een briefhoofd maken [Making a letterhead]; pp. 25–31
Shortly after successfully compiling my first TEX document, I wanted to switch over as many documents as possible to TEX. And the notion of being able to typeset the letterhead at the same time as the text of the document seemed to me to be very nice. It seemed best not to clutter the individual letter files with code, so I put all the necessary commands into a separate style file. I also created a simpler letterhead to put on following pages if the letter is longer than one page. Thanks to the tips of Henk de Haan, I have been able to help others in the course of time to make their own letterhead.
[Translation of author's abstract]

Brooks Moses, MetaPlot, MetaContour, and other collaborations with METAPOST; pp. 32–39
Most methods of creating plots in METAPOST work by doing all of their calculations in METAPOST, or by doing all of their calculations in a preprocessing program. There are advantages to dividing the work more equitably by doing the mathematical and data-visualization calculations in a preprocessing program and doing the graphical and layout calculations in METAPOST. The MetaPlot package

provides a standard, flexible, interface for accomplishing such a collaboration between programs, and includes a general-purpose set of formatting macros that are applicable to a wide range of plot types. Examples are shown of linear plots with idiosyncratic annotation and two-dimensional contour plots with lines and filled contours on a non-Cartesian mesh.
[Author's abstract]

Willi Egger and Hans Hagen, Support for typesetting Greek in ConTEXt; pp. 40–45

There are situations where one needs to typeset pieces of text in Greek. Until recently there was no direct support to do this in ConTEXt. With the integration of the module greek this has changed. The basics were built by Giuseppe Bilotta (Italy). The module uses a subset of the cb-greek fonts. The article describes the module and the way Greek text is coded. Several examples of Greek text are given.
[Author's abstract (edited)]

Steve Grathwohl, A simple book design in ConTEXt; pp. 46–51

I show how a simple book design can be implemented in ConTEXt.
[Author's abstract]

Adam Lindsay, OpenType in ConTEXt; pp. 52–58

This is a summary of issues encountered and solutions implemented in order to support some advanced OpenType features in ConTEXt. This article describes an accompanying set of support files that address installation (using TEXfont), accommodating extended optical families, and some "pro" font features. The extended character set afforded by pro fonts enables support for comprehensive small caps and old-style figures. Although the typescripts and commands are described together, certain features (like variant encodings for TEXfont and optical typescripts) can be used independently of the other features described. [Author's abstract (edited)]

Hans Hagen, Fontgebruik [Font usage]; pp. 59–61

Hans Hagen presents a very extravagant title: a back page from an 1899 handbook. The editors of MAPS offer a prize for the best and most elegant TEX-recreation of this layout.
[Translation of editor's abstract]

Frans Goddijn, Conversies [Conversions]; pp. 62–66

A look back at 12 years with a software package that in one way or another has made friendships.
[From the author's introduction]

Siep Kroonenberg, Exact layout with LATEX; pp. 67–70

This article describes several techniques useful for implementing a professionally designed layout such as a letterhead. [Author's abstract]

Wybo Dekker, Boekdrukken en valkuilen [Book printing and pitfalls]; pp. 71–76

To set a book, that's one thing, but then to also get it nicely printed... I'd like to take you along the pitfalls... learn and have (malicious) fun.
[Translation of author's abstract]

Eckhart Guthöhrlein, Object-oriented graphics with MetaObj; pp. 77–86

MetaObj is a macro package for METAPOST, a programming language for graphics producing PostScript output, based on the well-known METAFONT. MetaObj is written and maintained by Denis B. Roegel. It has been released under the LPPL and is available from CTAN. MetaObj provides very high-level object-oriented macros, which simplify the construction of complicated drawings by defining objects of arbitrary complexity and combining them into larger structures. This is already reflected in the name of the package: MetaObj is short for "METAPOST Objects".
[Author's abstract (edited)]

Patrick Gundlach, contextgarden.net; pp. 87–90

The project contextgarden.net was started to enhance the documentation of ConTEXt. It consists of several web services that provide the technical base for documentation. A large amount of the content is provided by the visitors to the web site.
[Author's abstract (edited)]

Hans Hagen, Fonts, more than a sample; pp. 91–94

Some time ago the NTG members received a colorful little booklet showing a lot of fonts. Since these fonts come with TEX Live, a ConTEXt user may be tempted to use them. The bad news is that fonts are always a bit troublesome in TEX distributions and recent changes in the TEX directory structure haven't made life easier. However, the good news is that it is doable to get these fonts working for you. Here I will present a few recipes, but I avoid discussing the 'dirty details'. These are covered in the manuals. [From the author's introduction]

Willi Egger and Frans Goddijn, Bloei der decadence [Flowering of decadence]; pp. 95–98

The book *Flowering of decadence* by Johan Polak has been out of print for many years, but it is now available as a PDF, freely downloadable via

the Internet. For those who want to read on the computer screen there is an interactive screen version, and another version is suitable for print. Both new editions of the book have been created via ConTEXt. This article describes some aspects of setting up this project. It was a complicated matter, due to the huge quantity of references to books, magazines, persons, place names and other terms. Our goal was to keep the process as simple as possible. Therefore we used no Plain TEX hacks, but rather simple methods typical of ConTEXt. Also we wanted a screen version of the book with a relatively small number of navigation files that could also be compiled as a paper version. A particular challenge was a piece of Greek text in a footnote.

[Translation of author's abstract]

Hendri Adriaens and Uwe Kern, Keys and values; pp. 99–103

This article introduces the xkeyval package as an extension of the well-known keyval package. The package provides more flexible commands, syntax enhancements, and a new option processing mechanism for class and package options using the general `key=value` syntax. [Author's abstract]

Taco Hoekwater, Boekbespreking vormwijzer [Book review]; pp. 104–105

Book review of *Display: A guide to creating and (re)producing printed matter*, by K. F. Treebus.

[From the author's introduction]

[Compiled by Steve Peter]

Calendar

2005

Apr 30 – May 3 BachoTEX 2005, 13th annual meeting of the Polish TEX Users' Group (GUST), "The Art of TEX Programming", Bachotek, Brodnica Lake District, Poland. For information, visit http://www.gust.org.pl/BachoTeX/2005/.

May 10 – Jul 17 In Flight: A traveling juried exhibition of books by members of the Guild of Book Workers. University of Texas, Austin, Texas. Sites and dates are listed at http://palimpsest.stanford.edu/byorg/gbw.

May 11 – Jun 16 From chisel to pen: inscriptional letterforms from early Christian Wales. An exhibition at the St. Bride Printing Library, London, England. For information, visit http://www.stbride.org/events.html.

May 22 – 27 Book History at A&M: The Fourth Annual Texas A&M Workshop on the History of Books and Printing. Texas A&M University, College Station, Texas. For information, visit http://lib-oldweb.tamu.edu/cushing/bookhistory/2005.html.

May 24 – 27 XTech Conference, "XML, the Web and Beyond", Amsterdam RAI Centre, Netherlands. For information, visit http://www.xtech-conference.org/.

May 25 – 28 CIDE.8, Conférence Internationale sur le Document Electronique, "Multilingualism", Beirut, Lebanon. For information, visit http://www.certic.unicaen.fr/cide8/.

Jun 1 – 3 Society for Scholarly Publishing, 27th annual meeting, "Expanding the World of Scholarly Publishing", Boston, Massachusetts. For information, visit http://www.sspnet.org.

Jun 6 – 9 Seybold Seminars, Amsterdam, Netherlands. For information, visit http://www.seybold365.com/2005/.

Jun 6 – Jul 29 Rare Book School, University of Virginia, Charlottesville, Virginia. Many one-week courses on topics concerning typography, bookbinding, calligraphy, printing, electronic texts, and more. For information, visit http://www.virginia.edu/oldbooks.

Jun 8 – Nov 13 70 Years of Penguin Design: Exhibition, Room 74, Twentieth Century Gallery, Victoria & Albert Museum, London, England.

Practical TEX 2005
Friday Center for Continuing Education, Chapel Hill, North Carolina.

Jun 14 – 17 Workshops and presentations on LATEX, TEX, ConTEXt, and more. For information, visit http://www.tug.org/practicaltex2005/.

Jun 15 – 18 ALLC/ACH-2005, Joint International Conference of the Association for Computers and the Humanities, and Association for Literary and Linguistic Computing, "The International Conference on Humanities Computing and Digital Scholarship", University of Victoria, British Columbia. For information, visit http://web.uvic.ca/hrd/achallc2005/ or the organization web site at http://www.ach.org.

Jun 24 – 26 NTG 35th meeting, Terschelling, Netherlands. For information, visit http://www.ntg.nl/bijeen/bijeen35.html.

Status as of 1 June 2005

For additional information on TUG-sponsored events listed here, contact the TUG office (+1 503 223-9994, fax: +1 503 223-3960, e-mail: office@tug.org). For events sponsored by other organizations, please use the contact address provided.

An updated version of this calendar is online at http://www.tug.org/calendar/.

Additional type-related events are listed in the Typophile calendar, at http://www.icalx.com/html/typophile/month.php?cal=Typophile.

Jul 14 – 17 SHARP Conference (Society for the History of Authorship, Reading and Publishing), "Navigating Texts and Contexts". Dalhousie University, Halifax, Canada. For information, visit http://sharpweb.org/ or http://www.dal.ca/~sharp05/.

Jul 20 – 24 TypeCon2005, "Alphabet City", Parsons School of Design, New York City. For information, visit http://www.tdc.org/news/2004typecon2005.html.

Jul 22 – 25 "The Changing Book: Traditions in Design, Production and Preservation", University of Iowa Libraries, Iowa City, Iowa. For information, visit http://www.lib.uiowa.edu/book2005/.

Jul 31 – Aug 4 SIGGRAPH 2005, Los Angeles, California. For information, visit http://www.siggraph.org/s2005/.

Aug 1 – 5 *Extreme* Markup Languages 2005, Montréal, Québec. For information, visit http://www.extrememarkup.com/extreme/.

TUG 2005
Wuhan, China.

Aug 23 – 25 The 26th annual meeting of the TEX Users Group. For information, visit http://www.tug.org/tug2005/.

Aug 26 – 28 Celebrating Johnson's *Dictionary* (1755–2005), Pembroke College, Oxford, England. For information, visit http://www.pmb.ox.ac.uk/pembroke_college/johnson_index.

Sep 7 – Oct 6 The Graven Image Press: lettercutting and visual metaphor in the work of Stan Greer. An exhibition at the St. Bride Printing Library, London, England. For information, visit http://www.stbride.org/events.html.

Sep 7 – 9 28th Internationalization and Unicode Conference, "Unicode 4.1 — Multilingual Challenges and Solutions for 2006", Orlando, Florida. For information, visit http://www.global-conference.com/iuc28/.

Sep 11 – 13 The Third International Conference on the Book, "Publishing, Libraries, Literacy and the Information Society", Oxford International Centre for Publishing Studies, Oxford Brookes University, Oxford, UK. For information, visit http://book-conference.com/.

Sep 11 – 14 Seybold Seminars, Chicago, Illinois. For information, visit http://www.seybold365.com/2005/.

Sep 15 – 18 Association Typographique Internationale (ATypI) annual conference, "On the Edge", Helsinki, Finland. For information, visit http://www.atypi.org/.

Sep 22 – 23 American Printing History Association conference, "[r]Evolution in Print: New Work in Printing History & Practice", Mills College, Oakland, California. For information, visit http://www.printinghistory.org/htm/conference/.

Oct 10 – 12 Fourth Annual St. Bride Conference, "Temporary Type", London, England. For information, visit http://www.stbride.org/conference.html.

Nov 2 – 4 ACM Symposium on Document Engineering, Bristol, UK. For information, visit http://www.documentengineering.org/.

Nov 3 – 5 "Reaching the Margins: The Colonial and Postcolonial Lives of the Book, 1765–2005", The Open University and the University of London, London, England. For information, visit http://www.sas.ac.uk/ies/Conferences/Open_University.htm.

Nov 14 – 18 XML 2005 Conference, Atlanta, Georgia. For information, visit http://2005.xmlconference.org/.

Nov 29 – Dec 2 Seybold Seminars, San Francisco. For information, visit http://www.seybold365.com/2005/.

2006

Jul 12 – 14 SHARP Conference (Society for the History of Authorship, Reading and Publishing). The Hague, Netherlands. For information, visit http://sharpweb.org/.

Jul 30 – Aug 3 SIGGRAPH 2006, Boston, Massachusetts. For information, visit http://www.siggraph.org/s2006/.

Sep 29 – 30 American Printing History Association conference, "The Atlantic World of Print in the Age of Franklin", Philadelphia, Pennsylvania. For information, visit http://www.printinghistory.org/htm/conference/.

A brief report on the first GUIT meeting

Onofrio de Bari*
Maurizio Himmelmann†

The first public meeting of GUIT (Gruppo Utilizzatori Italiani di TEX, `http://www.guit.sssup.it`) was held on 9th October 2004 in Pisa, Italy, at the Sant'Anna School of Advanced Studies.

We started our work in 2000 with just a small website built around a web forum, strongly emphasizing the idea of virtual community. The simple idea of gathering people in Italy interested in LATEX and its future is something that some years ago could have not been imagined, because of the absence of a gathering place in the TEX Italian world.

From our point of view the meeting was a great success. We needed some months to organize everything, but it was a great satisfaction for us to have about 50 people attend the conference.

After the talks were introduced, Klaus Hoeppner from DANTE e.V. spoke about upcoming TEX events worldwide, followed by talks about the status of LATEX in Italian universities, and common mistakes in LATEX syntax and font installation. It was then time for a nice coffee break (appreciated indeed by people attending there) followed by a talk about page layout and another about the generation of STATA (statistics software) tables to be embedded in a LATEX document.

The afternoon session started with a lecture about critical editions in LATEX, followed by multilingual bibliographies and XML (many thanks to our other foreign guest, Jean-Michel Hufflen), graphics and diagrams with XY-pic, `tex4ht` and a short discussion about the future goals of the Italian TUG, mainly focused on the shifting from our present *virtual* community to a *real* community.

The GUIT meeting was conceived to involve more people from Italy in the staff and organization affairs, and to have more members to reach further effectiveness in our activities. As we achieved this goal, the next GUIT meeting will be focused on boosting LATEX knowledge across Italy and developing stronger ties with other TEX user groups. For these and many other reasons we would be very glad of your presence! It will be held in October 2005 in Pisa. More information is available on the web, at `http://www.guit.sssup.it/guitmeeting/2005/`.

Last but not least, we are extremely grateful for the valuable DANTE e.V. support, and look forward to having more foreign guests next time.

* GUIT Public Relations Officer
† President of GUIT

TUG 2005
International Typesetting Conference
Announcement and Call for Participation

TUG 2005 will be held in Wuhan, China from August 23–25, 2005. CTUG (Chinese TEX User Group) has committed to undertake the conference affairs.

Wuhan is close to the birthplace of Taoism and the Three Gorges Reservoir. China is also the birthplace of typography in ancient times, and is simply a very interesting place to go.

For more information, see the conference web page at `http://tug.org/tug2005`, or email `tug2005@tug.org`.

Conference program

The keynote speaker will be Wai Wong, from the Chinese University of Hong Kong, on "Typesetting Chinese: A personal perspective".

Other speakers include Nelson Beebe, Jin-Hwan Cho, Hong Feng, Eitan Gurari, Hans Hagen, Yannis Haralambous, Jonathan Kew, Ross Moore, Karel Píška, Chris Rowley, Karel Skoupý, Philip Taylor, and Suki Venkat. A complete list of presentations and tutorials are available on the conference web site.

Conference registration

The conference fees and deadlines for members of any TEX user group (in US dollars):

Normal registration	July 1, 2005	$220
Late registration	August 1, 2005	$380

In all cases, non-user group members add $20.

Hope to see you there!

TUG Business

TEX Development Fund 2003–05 Report

Karl Berry and Kaja Christiansen

The TEX Development Fund was created by the TEX Users Group in 2003, under the aegis of the TUG Technical Council, to foster growth of TEX-related technical projects. The first set of grants was awarded in March 2003, with more grants awarded on a rolling basis after that. This report covers all projects, both completed and pending, as of April 2005. "Completed" refers to the work for the grant; they are generally ongoing development efforts, rather than projects which are done once-and-for-all.

We are especially appreciative of the ongoing support from individuals. Since its inception, more than 200 donations have been received, allowing us to make several more grants than would otherwise have been possible. Thank you, everyone!

For application information, the complete list of donors, and more, please see the development fund web site.

◇ Karl Berry and Kaja Christiansen
devfund@tug.org
http://tug.org/tc/devfund/

Completed projects

1 Latin Modern extensions

Applicants: Boguslaw Jackowski, Janusz Nowacki, Poland,
http://www.ctan.org/tex-archive/fonts/lm.
Amount: US$2000; acceptance date: 19 May 2004.

Continuing enhancement of the Latin Modern family of fonts.

This was completed by 20 April 2004, with the Latin Modern 0.98.3 release. The related article "Latin Modern: Enhancing Computer Modern with accents, accents, accents", was presented at TUG 2003 and published in *TUGboat* 24(1).

2 pdfTEX extensions

Applicant: Hàn Thế Thành, Vietnam,
http://www.pdftex.org.
Amount: US$1500; acceptance date: 26 March 2004.

1. New primitives to provide more control over the quality of typesetting complex documents (feedback as well as manipulating the result of breaking paragraphs into lines).
2. A primitive to ease the use of font expansion with pdftex, so one can use font expansion having expanded TFM's (which are complicated to generate for an average user).

This was completed by 14 October 2004, and the pdftex 1.20a release includes this work. The related article "Micro-typographic extensions of pdfTEX in practice" was presented at Practical TEX 2004 and published in *TUGboat* 25(1).

3 Source release of i-Installer v2

Applicant: Gerben Wierda, The Netherlands,
http://sourceforge.net/projects/ii2.
Amount: US$1500; acceptance date: 3 November 2003.

Make source release of new version of i-Installer, the engine used for installing and configuring the applicant's gwTEX distribution for Mac OS X.

This was completed on 28 February 2004, and the new version is available online. Article forthcoming.

Projects underway

4 TEXmuse

Applicant: Federico Garcia, USA.
Amount: US$1000; acceptance date: 11 April 2005.

Design of algorithms and code implementation for the first stage of the TEXmuse project for musical typesetting.

The 'first stage' consists of code that is able to typeset the basic musical text of Bach's 15 inventions. These pieces are for piano and only two voices: two staves and one voice per staff. Being from the Baroque, they feature interpretative notation (slurs, articulations, etc.) only in a very limited way. All of this makes these pieces a good first stage in the development of TEXmuse.

5 Baskerville

Applicant: Hrant Papazian, USA,
http://themicrofoundry.com.
Amount: US$3000; acceptance date: 19 October 2004.

Design and implementation of a Baskerville typeface revival to high standards of typographic quality, historical sensitivity, and usability.

The typeface family will include two weights (regular and bold), each with a true italic. The fonts will cover the character ranges Basic Latin, Latin-1 Supplement, and Latin Extended-A, as defined by Unicode.

6 Using Omega to generate XML and MathML from TEX documents

Applicant: John Plaice, Australia.
Amount: US$2000; date: April 2003.

Since 1998, the Omega Project has been capable of generating MathML and XML directly from the typesetting engine. In this project, we propose to further develop the XML- and MathML-generation capabilities of the Omega Project.

The Omega approach to generating markup languages from TEX input consists of two parts:

- modifying the mathematics part of the typesetting engine so that MathML can be automatically generated;
- adding new macro primitives so that XML opening and closing tags can be produced by the programmer.

In this project, we propose to comprehensively cover the high-level LATEX and $\mathcal{AMS}$-LATEX macros and define a matching DTD/schema, and ensure that Omega can correctly translate a correct LATEX document with mathematics into XML and MathML. High-level macros will be written, new macro primitives will be defined, and modifications will be made to the typesetting engine.

Although this work is not complete, the related article "XLATEX, a DTD/schema which is very close to LATEX" was presented at EuroTEX 2003 and published in *TUGboat* 24(3).

7 Combining the extensions of TEX into one system

Applicant: John Plaice, Australia.
Amount: US$2000; date: April 2003.

There are currently three large extensions to TEX:

- Omega has focused on extensions supporting multilingual typesetting;
- ε-TEX has focused on extensions to the macro language and its tracing;
- pdfTEX has focused on producing PDF rather than DVI.

ε-TEX and pdfTEX have already been combined into pdf-ε-TEX, and more recently Giuseppe Bilotta has created e-Omega (now named Aleph).

In this project, we propose to combine the key elements of ε-TEX and pdfTEX with Omega. In addition to combining several Pascal Web change files and integrating the associated C/C++ code, an important objective will be to harness the power of Omega's Translation Processes and context manipulation code to generate high-quality PDF files.

Although this work is not complete, the related article "Moving Omega to a C++-based platform" was presented at TUG 2004 and published by Springer-Verlag in the conference proceedings, *TEX, XML, and Digital Typography*.

8 CTAN release of critical edition support for LATEX

Applicant: David Kastrup, Germany.
Amount: US$1500; date: April 2003.

The project described here is very large. Only a small part is funded through this grant: making it possible for the main work to be included on CTAN and integrated into the main LATEX sources. For background information, the full description is at `http://tug.org/tc/devfund/grants.html`.

Although this work is not complete, the related article "The `bigfoot` bundle for critical editions" was presented at TUG 2004 and published in the conference preprints distributed to TUG members.

Accommodation of the footnote apparatus Critical editions usually contain multiple footnote apparatus. A typical set for an edition of a commentary would be

1. Footnotes of the original commentator to the basic text, numbered sequentially.
2. Footnotes pointing out variations of various editions or manuscripts of the original publication. Those would typically be indicated with letters starting from "a" on each side.
3. Footnotes containing comments of the current editor.

Of course, this is just a simple example: much more contrived apparatus can be seen, too. In the first stage of the project, a separate footnote style will be designed that overrides only small parts of the standard LATEX 2_ε output routine, probably building upon the ncctools package.

Other issues While LATEX provides for margin notes and paragraphs, the mechanism is not versatile enough to cater for either margin notes in footnotes or multiple levels of margin notes.

The possibilities for editions of course are limitless, nevertheless there are basic building blocks from which a page layout may be built up. The current LATEX output routine does not accommodate such formats, nor would it be useful to accommodate it in the base class. However, there are a lot of elements that can be systematically tackled and given interfaces, so that the average document designer could merely aggregate boxes, insertions and their processing in a reasonably easy way.

TUG financial statements for 2004

Robin Laakso

This financial report for 2004 also includes numbers from 2003, for purposes of comparison. As usual, the accounts have been reviewed by TUG's accountant but have not been audited.

Revenue highlights

Overall, the TEX Users Group suffered an 11 percent drop in income in 2004 compared to 2003. TUG membership dues were $94.5K at the end of 2004, compared to $105.5K in 2003. This represents a decline in membership of approximately 200: from about 1800 TUG members in 2003 to just over 1600 in 2004. Joint member dues from NTG (the Dutch group) and UK-TUG dropped 15 percent and 8 percent, respectively.

Interest income was down 29 percent in 2004 compared to 2003, almost entirely due to the 24 month CD coming due in May of 2004, which renewed (for 12 months) at half the previous rate, in turn due to prevailing economic conditions.

TUG also realized some income-producing successes in 2004:

- TUG store sales increased from $2040 (the store opened in April, 2003) to $5640 in 2004, primarily due to a full year of operation, and an increase in software sales.
- The Pearson Publishing Group (which includes Addison-Wesley) and TUG affiliated to offer TEXnical books to members and non-members alike at a 30 percent discount via the TUG web site. TUG receives 15 percent of the gross sales, which resulted in over $800 increase in royalties compared to 2003.
- In the last quarter of 2004 TUG partnered with WinEdt to offer a discount on their licenses for TUG members. Members receive a 25 to 30 percent discount on WinEdt licenses, depending on which category is purchased.
- General contribution income increased 17 percent from 2003 to 2004, largely due to a contribution and matching endowment received from an individual TUG member.
- A LATEX3 donation line item was added to the membership forms in 2004 resulting in over $1000 in contributions for that purpose.
- The Practical TEX 2004 conference held in San Francisco essentially broke even (as was budgeted).

The total dollar increase from the above was $6736.

Expense highlights

Payroll, software production and mailing, and *TUGboat* production and mailing continue to be the major expense items.

Payroll was down 1 percent in 2004 from 2003.

Software production and mailing was down 12 percent, from $10.2K in 2003 to $9K in 2004. The savings is mostly due to having some of the software manufactured locally rather than overseas, and because the lesser weight of the TEX Collection in 2004 resulted in lower postage costs.

TUGboat production and mailing at $26.2K in 2004 consists of three publications, the first one of which (the special non-*TUGboat* "preprints" publication) was produced and mailed at a cost of $10.2K; the remaining two issues are booked at the accrued amount of $8K each.

Notable contributions and allocations made by TUG in 2004:

- TEX Development Fund: $5000
- TUG Bursary: $2000
- Adobe/Apple Technical Group: $1000
- Apple developer membership: $500
- Miscellaneous donations: $950

If you have any questions about TUG's finances, or if you would like to help with any TUG-related activities, please contact the TUG office.

◇ Robin Laakso
TUG Executive Director
office@tug.org

TeX Users Group

Balance Sheet Prev Year Comparison

As of December 31, 2004

	Dec 31, 04	Dec 31, 03
ASSETS		
Current Assets		
Checking/Savings		
OregonTelco PrimeShare	5	133,750
OregonTelco 12 Mo CD	101,056	
OregonTelco Mmarket	10,406	
BofA 9 Mo CD	10,058	
BofA Maximizer	19,645	28,902
BOA Checking		
Paypal	1,511	
BOA Checking	1,080	-7,527
Total BOA Checking and PayPal	2,591	-7,527
BOA Money Mkt Bursry	1,202	1,711
Petty Cash	10	10
Total Checking/Savings	144,973	156,846
Accounts Receivable		
Accounts Receivable	525	300
Total Accounts Receivable	525	300
Other Current Assets		
Deferred PracTeX expense	790	
Deferred Intl conf expense	250	
Deposits	10	10
Total Other Current Assets	1,050	10
Total Current Assets	146,548	157,156
Fixed Assets		
Equipment	44,895	44,625
Accumulated Depreciation	-42,605	-40,300
Total Fixed Assets	2,290	4,325
Total Fixed Assets	2,290	4,325
TOTAL ASSETS	**148,838**	**161,481**
LIABILITIES & EQUITY		
Liabilities		
Current Liabilities		
Accounts Payable		
Accounts Payable	23,574	31,104
Total Accounts Payable	23,574	31,104
Other Current Liabilities		
Deferred conference donations		100
Deferred conference income	265	
Deferred contributions	200	
Deferred member income	680	
AMS Prepaid Memberships		1,800
Payroll Liabilities		
Federal P/R Taxes Payable	875	885
State P/R Taxes Payable	192	195
Total Payroll Liabilities	1,067	1,080
Total Other Current Liabilities	2,212	2,980
Total Current Liabilities	25,786	34,084
Total Liabilities	25,786	34,084
Equity		
Restricted DevFund as of 12/31	4,058	3,433
Restricted Bursary as of 12/31	1,202	1,711
Restricted LaTeX3 as of 12/31	1,074	-76
Unrestricted as of 1/1	121,064	122,588
Net Income	-4,346	-259
Total Equity	123,052	127,397
TOTAL LIABILITIES & EQUITY	**148,838**	**161,481**

TeX Users Group

Profit & Loss Prev Year Comparison

January through December 2004

	Jan - Dec 04	Jan - Dec 03
Ordinary Income/Expense		
Income		
Membership Dues	101,631	113,597
Product Sales	8,259	4,656
Contributions Income	7,453	5,743
Practical TeX Conference	259	4,915
Conference Classes	-555	
Interest Income	4,295	6,064
Advertising Income	950	400
Bursary	-1,009	381
TeX Development Fund	625	3,433
LaTeX 3	1,149	-234
Miscellaneous Income		0
Total Income	123,057	138,955
Cost of Goods Sold		
TUGboat Prod/Mailing	26,242	22,500
Software Production/Mailing	8,962	10,207
Postage/Delivery - Members	5,111	3,684
Conf Exp, office + overhead	1,115	3,698
Member Renewal		469
Copy/Printing for members	389	67
Total COGS	41,819	40,625
Gross Profit	81,238	98,330
Expense		
Contributions made by TUG	8,449	21,100
Office Overhead	12,788	13,233
Payroll Exp	59,768	60,091
Contract Labor		735
Professional Fees	2,016	1,505
Depreciation Expense	2,305	3,334
Total Expense	85,326	99,998
Net Ordinary Income	-4,088	-1,668
Other Income/Expense		
Other Income		
Prior year adjust	-4,292	-3,592
Other Income	4,034	5,000
Total Other Income	-258	1,408
Net Other Income	-258	1,408
Net Income	**-4,346**	**-260**

TUG 2005 Election Report

Barbara Beeton
for the Elections Committee

The deadline has come, the ballots have been counted, and the results are in.

Karl Berry has been elected TUG president for the term that ends with the 2007 annual meeting. The following votes were counted:

Karl Berry, 183
Lance Carnes, 177

Both candidates made a good showing, although the total number of voters was only a third of those eligible.

Of 1080 members as of the ballot closing date (May 17), 360 valid ballots were received. Ten ballots postmarked after the closing date, and three received with no return address were not opened or counted. (Although there may be a concern for privacy, the return address is the only way to tell that a ballot is coming from an eligible member. The count was made by a teller with no stake in the outcome, who was supplied with a list of eligible voters; every envelope was checked against this list before the envelopes were opened, and care was taken to make sure that only the envelope with the latest postmark was processed from any one voter.)

As previously announced, the number of candidates for open board positions was fewer than positions, so these board candidates were declared duly elected for a term ending with the 2009 annual meeting: Steve Grathwohl, Jim Hefferon, Klaus Höppner, Ross Moore, Arthur Ogawa, Steve Peter and David Walden. Continuing board members with terms ending in 2007, are: Barbara Beeton, Kaja Christiansen, Susan DeMeritt, Gerree Pecht, Cheryl Ponchin, Sam Rhoads and Philip Taylor. Also, Lance Carnes has been appointed to the board, for a term ending in 2007.

In this issue, statements for all the candidates, both for President and for the Board, are appended (in alphabetical order). They are also available through the election web page.

The Committee gratefully acknowledges the diligent work of the TUG executive director, Robin Laakso, in receiving, organizing, and validating the membership of nominees and their respective nominators.

This was the first contested election since 1991. Thanks to everyone for their participation.

◇ Barbara Beeton
for the Elections Committee

Karl Berry

Biography:

I have served as TUG president since 2003, and was a board member for two terms prior to that. During my term as president, we've enacted some new initiatives, notably: expanded the availability of the special reduced membership rate (to past graduates and citizens of countries with modest economies); increased the memberships available to our institutional supporters; joined with Addison-Wesley in making their TEX (and other) books available at a substantial discount; and opened the online TUG store.

As president, I also serve on the conference committee, and thus was (and am) one of the organizers for all TUG-sponsored conferences, including TUG 2004 in Greece, TUG 2005 in China, and the new Practical TEX conference series, so far in San Francisco (2004) and Chapel Hill (2005).

I have also been on the TUG technical council for many years. I co-sponsored the creation of the TEX Development Fund in 2002, and act as one of the system administrators and webmasters for the TUG servers. I'm also one of the production staff for the *TUGboat* journal. I've administered TEX installations at many universities and companies over the years.

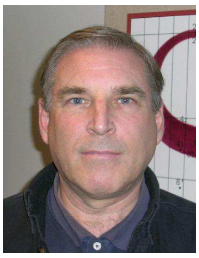
On the TEX development side, I'm currently co-editor of TEX Live, the largest free software TEX distribution. Previously, I maintained Unix TEX (Web2c) for several years. Along with Web2c, I developed Kpathsea, a freely redistributable library for path searching, and modified Dvips, Xdvi, and other drivers to use it; Eplain, a macro package extending plain TEX; a naming scheme for fonts; and other projects. I am also the maintainer of GNU Texinfo, the standard TEX-based documentation format for the GNU Project.

I am a co-author of *TEX for the Impatient*, an early comprehensive book on TEX, which is now freely available. I've also produced a number of books, articles, collections, and ephemera with and about TEX, studied typeface design, and co-written several articles on reading research and mathematical analysis of type. I first encountered and installed TEX in 1982, as a college undergraduate.

Personal statement:

I believe TUG can best serve the TEX community by working in partnership with the other TEX user groups worldwide, and sponsoring projects and conferences that will increase interest in and use of TEX. The quality of TEX's output remains unsurpassed, even now. It is our challenge to bring that quality to an even broader audience.

Lance Carnes



Biography:

- Involved with the TEX Users Group since its beginning in 1980
- Served on the Board of Directors from 1981 to 1991
- Proposed and helped organize the PracTEX 2004 conference
- Headed the Editorial Board to launch *The PracTEX Journal* in 2005 (see `http://www.tug.org/pracjourn`).

The main reason for submitting my name as a candidate for TUG President is to put an emphasis on Users in the TEX Users Group. For the past 25 years TUG has focused mostly on TEX software developers and power-users, while often forgetting the needs of day-to-day LaTEX and TEX users. From my experiences with PracTEX conference attendees and *PracTEX Journal* readers, it is clear they have many needs which TUG is not currently fulfilling. I feel TUG should shift its priorities to concentrate more on user education and training, and to provide more practical information in print and on-line.

In addition, there are challenges facing TUG, and I am ready to work with the Board to address them. Three areas I feel need attention are:

1. Membership is down. Some ideas for correcting this:
   - Reduce membership fees. By making *TUGboat* and TEX-Live optional benefits, the price of a membership could be about one-half what it is now.
   - Increase membership privileges. For example, give members exclusive on-line access to *TUGboat* and other resources.
   - Promote Institutional Memberships. Change to a sliding membership fee scale based on institution size, and provide membership privileges to everyone at the institution.
2. There are not enough training classes and materials.

   I think TUG should be the leader in providing training guidelines and curricula, and in offering classes and workshops. I would propose forming an Education Committee, composed of Board members and others in the community, which will design courses and materials for LATEX and TEX training.
3. Conference attendance is low.

   Some possibilities to boost conference attendance: require all TUG-sponsored conferences to have a mix of beginning, intermediate, and high-level presentations, possibly in parallel sessions. Require conferences to include classes and workshops appropriate for beginning and intermediate users.

I look forward to working with the Board and with all TUG members to continue TUG's traditional activities while putting several of the above ideas into practice. Together we can make this an organization which responds to the wishes and needs of both TEXperts and practical LATEX and TEX users.

Steve Grathwohl

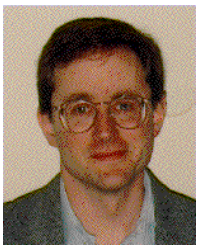
Biography:

I began using TEX in 1986 when a friend gave me his copy of the TEXbook and a pre-release version of Textures, which I tried with mixed success to run on my old Mac512K with only a single floppy drive. In a short time I had tossed off Word, WordPerfect, and other word processing systems; but it wasn't until I typeset my wife's dissertation (600pp, Middle English, Old French, multi-page tables) and began work at Duke University Press that I began using TEX in a serious, systematic way.

For the Press, in 1993, TEX was a peculiar dialect that mathematicians spoke, not really useful for production. Now, in 2005, TEX is used to produce seven of our journals, only one of them a mathematics journal. I am very pleased that I was able to demonstrate to the Press that TEX was more than capable of being a dependable production platform.

I think I bring to the TUG board the sensibilities of both an enthusiastic user of TEX and a reasonably hard-headed journals production guy who has to make decisions about what works within tight scheduling constraints.

Jim Hefferon



I've been involved with TEX for years, lately by maintaining the TUG branch of CTAN. I've been serving in an appointed position on the board, and I hope I can continue to help out.

Klaus Höppner

Biography:

I got a PhD in Physics in 1997. After some post-doctoral fellowships I have been working working in the Control Systems group of an accelerator center in Darmstadt, Germany, since 2002. My first contact to LATEX was in 1991, using it frequently since then.

I was preparing the CTAN snapshot on CD, distributed to the members of many user groups, from 1999 until 2002. I was heavily involved in the organization of several DANTE conferences and EuroTEX 2005. Since 2000, I am a member of the DANTE board, acting as vice president since 2002.

Personal statement:

In the years since Karl Berry's presidency the cooperation of TUG and European user groups improved a lot. My candidacy is in the hopes of helping to continue this trend. Projects like TEXLive and CTAN owe their success to the work of active volunteers, but also to the support and cooperation of the user groups.

I appreciate the start of *The PracTEX Journal*, the first online journal about TEX. I wish it could become a part of a future pool for TEX articles where authors can give their permission for translations and publishing these in the journals of other user groups.

Arthur Ogawa

The most important issue facing TUG today is its declining membership. I am running for membership on the TUG Board of Directors because I take this issue seriously.

I have served on the TUG Board from 1997 to the present and have served in the past as Secretary and Vice President, involving myself in the business of TUG as a member of its Executive Committee. During this time, I have watched as TUG's membership first staged a modest recovery from the lows of 1997 and then leveled off. The current trend is a slight yearly decline. Attendance at TUG conferences has also declined during this time. TUG's continued existence was greatly imperiled by the meager membership numbers of the late 1990s, and the current situation does not bode well.

While I do not feel that I possess the only answer to the problem, my committment is to address the matter and to find a solution, by working with the TUG Board, its Executive Director, and TUG's membership.

At the present time, TUG is a vigorous and vital organization. Its day-to-day operations are competently served by our office, staff and volunteers, and its Board of Directors and President work together effectively. I am convinced that TUG provides its members with valuable services and products, and that TUG supports important software efforts that most certainly benefit TEX users, whether or not members of TUG or any other user group.

Now it is time for TUG to ensure that its efforts to support and benefit TEX users will continue. How this is to be done is not clear at present, but I firmly believe that the TEX Users Group, which has been helping TEX users for over 25 years, can continue to do so. TEX is a free, popular, and robust software, and it continues to benefit people all over the world. It is our opportunity as TUG members to help with its further development, its dissemination, and its use by the many people who have embraced it.

I hope that you agree with me on the importance of TUG in this effort.

Steve Peter

Biography:

I am a linguist and publisher originally from Illinois, but now living in New Jersey. I first encountered TEX as a technical writer documenting Mathematica. Now I use TEX and friends (these days, lots of ConTEXt) for a majority of my publishing work, and occasionally consult on it. I am especially interested in multilingual typography and finding a sane way to typeset all of those crazy symbolisms linguists create. As if that weren't bad enough, I've recently begun studying typeface design.

I got involved in TUG via translations for *TUGboat*, where I also work on the production team. This past year, I was on the organizing committee for PracTEX San Francisco, co-edited the TUG 2004 conference pre-proceedings, and was appointed to the TUG Board (thanks, Karl!). Working with and for the community has been so rewarding that I've decided to run for a regular term on the board.

Personal statement:

The future of TEX and TUG lies in communication and working together to promote and sustain the amazing typographic quality associated with TEX and friends. I am especially interested in having TUG support various projects (technical and artistic) that will serve to bolster TEX and TUG's visibility in the world at large.

David Walden

Biography:

I was supposed to be studying math as an undergraduate at San Francisco State College; but, from my junior year I was hacking on the school's IBM 1620 computer. While working as a computer programmer at MIT's Lincoln Laboratory, I did the course work for a master's degree in computer science at MIT. Most of my career was at Bolt Beranek and Newman Inc. (BBN) in Cambridge, Massachusetts, where I was, in turn, a computer programmer, technical manager, and general manager. At BBN, I had the good fortune to be part of BBN's small ARPANET development team. Later I was involved in a variety of high tech professional services and product businesses, working in a variety of roles (technical, operations, business, and customer oriented). For more about me, see `http://www.walden-family.com/dave`.

Personal statement:

Throughout my business career and now during my so-called retirement years, I have always done considerable writing and editing. This led to my involvement since the late 1990s with TEX and as a member of TUG and now as a TUG volunteer (*The PracTEX Journal* editorial board, TUG Interview Corner, etc.). I am interested in serving on the TUG Board for three reasons:

1. To more explicitly serve the community that has so generously served me via comp.text.tex, CTAN, *TUGboat*, etc.
2. As a way of helping maintain the viability for years to come of TEX and the TEX world, entities I would call "national treasures" except for their world wide nature.
3. Because rubbing shoulders more closely with various TUG members will help me learn more about TEX faster.

As a TUG Board member, my frame of mind would be to get things done quickly and pragmatically with enough generality so evolution is possible.

Institutional Members

American Mathematical Society,
Providence, *Rhode Island*

Banca d'Italia,
Roma, *Italy*

Center for Computing Science,
Bowie, *Maryland*

Certicom Corp.,
Mississauga, *Ontario Canada*

CNRS - IDRIS,
Orsay, *France*

CSTUG, *Praha*, *Czech Republic*

Florida State University,
School of Computational Science and Information Technology,
Tallahassee, *Florida*

IBM Corporation,
T J Watson Research Center,
Yorktown, *New York*

Institute for Advanced Study,
Princeton, *New Jersey*

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

KTH Royal Institute of Technology, *Stockholm*, *Sweden*

Masaryk University,
Faculty of Informatics,
Brno, *Czechoslovakia*

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*

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

Uppsala University,
Uppsala, *Sweden*

Vanderbilt University,
Nashville, *Tennessee*

TEX Consultants

Ogawa, Arthur
40453 Cherokee Oaks Drive
Three Rivers, CA 93271-9743
(209) 561-4585
Email: arthur_ogawa@teleport.com
Bookbuilding services, including design, copyedit, art, and composition; color is my speciality. Custom TEX macros and LATEX2ε document classes and packages. Instruction, support, and consultation for workgroups and authors. Application development in LATEX, TEX, SGML, PostScript, Java, and C++. Database and corporate publishing. Extensive references.

Veytsman, Boris
2239 Double Eagle Ct.
Reston, VA 20191
(703) 860-0013
Email: boris@lk.net
I provide training, consulting, software design and implementation for Unix, Perl, SQL, TEX, and LATEX. I have authored several popular packages for LATEX and `latex2html`. I have contributed to several web-based projects for generating and typesetting reports.
For more information please visit my web page:
http://users.lk.net/~borisv.

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 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.

The TUG office mentions the consultants listed here to people seeking TEX workers. If you'd like to be included, or place a larger ad in *TUGboat*, please contact the office or see our web pages:

TEX Users Group
1466 NW Naito Parkway, Suite 3141
Portland, OR 97208-2311, U.S.A.

Phone: +1 503 223-9994
Fax: +1 503 223-3960
Email: office@tug.org
Web: http://tug.org/consultants.html
http://tug.org/TUGboat/advertising.html

TUGBOAT Volume 25 (2004), No. 2 / 2004

Introductory

Intermediate

Intermediate Plus

Advanced

Contents of publications from other TEX groups

Reports and notices