

# TUGBOAT

Volume 31, Number 3 / 2010

|                                      |     |  |
|--------------------------------------|-----|--|
| <b>General Delivery</b>              | 158 | From the president / <i>Karl Berry</i>   |
|                                      | 158 | Editorial comments / <i>Barbara Beeton</i><br>Matthew Carter named MacArthur Fellow;<br>Indie Excellence Awards for self-published books;<br>City maps made entirely of type; U&lc on line;<br>Some “under-the-covers” uses of T <sub>E</sub> X; Beyond literate programming |
|                                      | 160 | Hyphenation exception log / <i>Barbara Beeton</i>  |
| <b>Fonts</b>                         | 161 | A story of <i>kpfonts</i> : Reaching the limits of NFSS / <i>Christophe Caignaert</i>  |
| <b>Publishing</b>                    | 175 | Giving it away / <i>Jim Hefferon</i>   |
|                                      | 177 | Glisterings: Meandering miniature books / <i>Peter Wilson</i>  |
| <b>Software &amp; Tools</b>          | 184 | Three things you can do with LuaT <sub>E</sub> X that would be extremely painful otherwise /<br><i>Paul Isambert</i>   |
| <b>L<sup>A</sup>T<sub>E</sub>X</b>   | 191 | Some misunderstood or unknown L <sup>A</sup> T <sub>E</sub> X <sub>2<math>\epsilon</math></sub> tricks II / <i>Luca Merciadri</i>  |
| <b>L<sup>A</sup>T<sub>E</sub>X 3</b> | 194 | L <sup>A</sup> T <sub>E</sub> X3 news, issue 3 / <i>L<sup>A</sup>T<sub>E</sub>X Project Team</i>   |
|                                      | 195 | From <code>\newcommand</code> to <code>\DocumentNewCommand</code> with <code>xparse</code> / <i>Joseph Wright</i>  |
| <b>ConT<sub>E</sub>Xt</b>            | 197 | Tagged PDF in ConT <sub>E</sub> Xt / <i>Hans Hagen</i>   |
|                                      | 203 | Introduction to colours in ConT <sub>E</sub> Xt MKiV / <i>Luigi Scarso</i>   |
| <b>Electronic Documents</b>          | 208 | Generate T <sub>E</sub> X documents using <code>pdfscript</code> / <i>Oleg Parashchenko</i>  |
|                                      | 213 | <code>illumino</code> : An XML document production system with a T <sub>E</sub> X core /<br><i>Matteo Centonza and Vito Piserchia</i>  |
|                                      | 219 | Managing printed and online versions of large educational documents /<br><i>Jean-Michel Hufflen</i>  |
| <b>Problems</b>                      | 223 | Aligning text in diagrams exported by Mathematica: A question about the<br>PostScript infrastructure / <i>Michael Barnett</i>  |
| <b>Hints &amp; Tricks</b>            | 227 | The treasure chest / <i>Karl Berry</i>   |
| <b>Abstracts</b>                     | 229 | <i>ArsT<sub>E</sub>Xnica</i> : Contents of issue 9 (October 2010)  |
|                                      | 230 | <i>MAPS</i> : Contents of issue 40 (2010)  |
|                                      | 231 | <i>The PracT<sub>E</sub>X Journal</i> : Contents of issue 2010-1   |
|                                      | 233 | <i>Zpravodaj</i> : Contents of issues 20(1–2), 20(3) (2010)  |
|                                      | 236 | <i>Die T<sub>E</sub>Xnische Komödie</i> : Contents of issue 2010/3   |
| <b>News</b>                          | 237 | Calendar   |
| <b>Advertisements</b>                | 238 | T <sub>E</sub> X consulting and production services  |
| <b>TUG Business</b>                  | 239 | TUG institutional members  |
|                                      | 240 | TUG 2011 election  |

## TeX Users Group

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

### Memberships and Subscriptions

2010 dues for individual members are as follows:

- Ordinary members: \$95.
- Students/Seniors: \$55.

The discounted rate of \$55 is also available to citizens of countries with modest economies, as detailed on our web site.

Membership in the TeX Users Group is for the calendar year, and includes all issues of *TUGboat* for the year in which membership begins or is renewed, as well as software distributions and other benefits. Individual membership is open only to named individuals, and carries with it such rights and responsibilities as voting in TUG elections. For membership information, visit the TUG web site.

Also, (non-voting) *TUGboat* subscriptions are available to organizations and others wishing to receive *TUGboat* in a name other than that of an individual. The subscription rate is \$100 per year, including air mail delivery.

### Institutional Membership

Institutional membership is a means of showing continuing interest in and support for both TeX and the TeX Users Group, as well as providing a discounted group rate and other benefits. For further information, see <http://tug.org/instmem.html> or contact the TUG office.

TeX is a trademark of the American Mathematical Society.

Copyright © 2010 TeX Users Group.

Copyright to individual articles within this publication remains with their authors, so the articles may not be reproduced, distributed or translated without the authors' permission.

For the editorial and other material not ascribed to a particular author, permission is granted to make and distribute verbatim copies without royalty, in any medium, provided the copyright notice and this permission notice are preserved.

Permission is also granted to make, copy and distribute translations of such editorial material into another language, except that the TeX Users Group must approve translations of this permission notice itself. Lacking such approval, the original English permission notice must be included.

## Board of Directors

Donald Knuth, *Grand Wizard of TeX-arcana*<sup>†</sup>  
Karl Berry, *President*<sup>\*</sup>  
Kaja Christiansen\*, *Vice President*  
David Walden\*, *Treasurer*  
Susan DeMeritt\*, *Secretary*  
Barbara Beeton  
Jon Breitenbucher  
Jonathan Fine  
Steve Grathwohl  
Jim Hefferon  
Klaus Höppner  
Ross Moore  
Steve Peter  
Cheryl Ponchin  
Philip Taylor  
Raymond Goucher, *Founding Executive Director*<sup>†</sup>  
Hermann Zapf, *Wizard of Fonts*<sup>†</sup>

<sup>\*</sup>member of executive committee

<sup>†</sup>honorary

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

### Addresses

TeX Users Group  
P. O. Box 2311  
Portland, OR 97208-2311  
U.S.A.

### Telephone

+1 503 223-9994

### Fax

+1 206 203-3960

### Web

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

### Electronic Mail

(Internet)

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

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

Technical support for  
TeX users:  
[support@tug.org](mailto:support@tug.org)

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

### Have a suggestion? Problems not resolved?

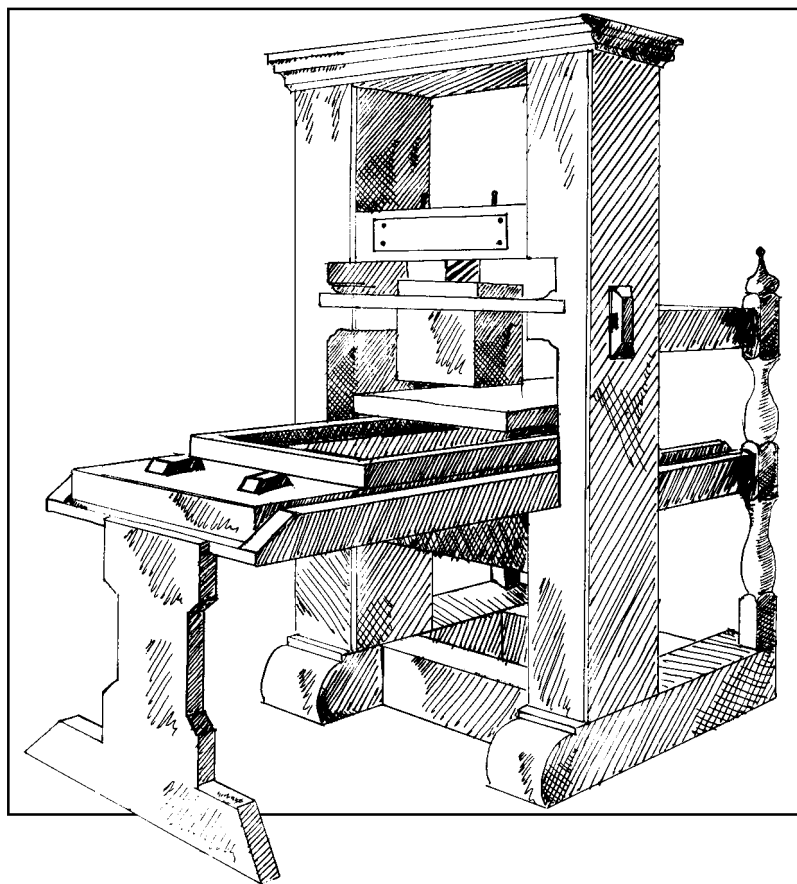
The TUG Board wants to hear from you:  
Please email [board@tug.org](mailto:board@tug.org).

[printing date: November 2010]

Printed in U.S.A.

# TUGBOAT

The Communications of the TeX Users Group



Volume 31, Number 3, 2010

## TeX Users Group

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

### Memberships and Subscriptions

2010 dues for individual members are as follows:

- Ordinary members: \$95.
- Students/Seniors: \$55.

The discounted rate of \$55 is also available to citizens of countries with modest economies, as detailed on our web site.

Membership in the TeX Users Group is for the calendar year, and includes all issues of *TUGboat* for the year in which membership begins or is renewed, as well as software distributions and other benefits. Individual membership is open only to named individuals, and carries with it such rights and responsibilities as voting in TUG elections. For membership information, visit the TUG web site.

Also, (non-voting) *TUGboat* subscriptions are available to organizations and others wishing to receive *TUGboat* in a name other than that of an individual. The subscription rate is \$100 per year, including air mail delivery.

### Institutional Membership

Institutional membership is a means of showing continuing interest in and support for both TeX and the TeX Users Group, as well as providing a discounted group rate and other benefits. For further information, see <http://tug.org/instmem.html> or contact the TUG office.

TeX is a trademark of the American Mathematical Society.

Copyright © 2010 TeX Users Group.

Copyright to individual articles within this publication remains with their authors, so the articles may not be reproduced, distributed or translated without the authors' permission.

For the editorial and other material not ascribed to a particular author, permission is granted to make and distribute verbatim copies without royalty, in any medium, provided the copyright notice and this permission notice are preserved.

Permission is also granted to make, copy and distribute translations of such editorial material into another language, except that the TeX Users Group must approve translations of this permission notice itself. Lacking such approval, the original English permission notice must be included.

## Board of Directors

Donald Knuth, *Grand Wizard of TeX-arcana*<sup>†</sup>  
Karl Berry, *President*<sup>\*</sup>  
Kaja Christiansen\*, *Vice President*  
David Walden\*, *Treasurer*  
Susan DeMeritt\*, *Secretary*  
Barbara Beeton  
Jon Breitenbucher  
Jonathan Fine  
Steve Grathwohl  
Jim Hefferon  
Klaus Höppner  
Ross Moore  
Steve Peter  
Cheryl Ponchin  
Philip Taylor  
Raymond Goucher, *Founding Executive Director*<sup>†</sup>  
Hermann Zapf, *Wizard of Fonts*<sup>†</sup>

<sup>\*</sup>member of executive committee

<sup>†</sup>honorary

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

### Addresses

TeX Users Group  
P. O. Box 2311  
Portland, OR 97208-2311  
U.S.A.

### Telephone

+1 503 223-9994

### Fax

+1 206 203-3960

### Web

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

### Electronic Mail

(Internet)

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

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

Technical support for  
TeX users:  
[support@tug.org](mailto:support@tug.org)

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

### Have a suggestion? Problems not resolved?

The TUG Board wants to hear from you:  
Please email [board@tug.org](mailto:board@tug.org).

[printing date: November 2010]

Printed in U.S.A.



---

## From the President

Karl Berry

### Conferences

TUG 2010 (<http://tug.org/tug2010>) was a great success, with Don Knuth's earthshaking (or at least side-splitting) announcement, and numerous participants from all over the T<sub>E</sub>X world. Video recordings of many talks are at <http://river-valley.tv/conferences/tug-2010>, thanks to Kaveh Bazargan and River Valley Technologies.

As part of the conference, a commemorative book was prepared, with selected articles from *TUGboat* by the Stanford T<sub>E</sub>X project members, along with a foreword by Barbara Beeton and drawings by Duane Bibby, some especially commissioned for the anniversary. The hardcover book is available from the TUG store and general online bookstores. Also, the full PDF is available online to TUG members in the members area — <http://tug.org/store/tug10>.

Looking ahead to 2011, the TUG meeting will be held in Cairo, Egypt, from November 14–17. Hosam Fahmy, long-time TUG supporter and *TUGboat* contributor, is the chief organizer. <http://tug.org/tug2011> will be updated as planning proceeds.

<http://tug.org/meetings> has information on most T<sub>E</sub>X meetings, past, present, and future.

### Interviews

Since my last column, Dave Walden has interviewed Bart Childs, Joe Weening, Frank Liang, and Herbert Voß for the TUG Interview Corner (<http://tug.org/interviews>).

Incidentally, the book of interviews we prepared last year is still available — <http://tug.org/store/texpeople>. The full PDF for this book is now also in the TUG members area.

### Software

The 2010 release of the T<sub>E</sub>X Collection software was made near the beginning of September. It's been available for download from CTAN since that time. The physical DVDs have been manufactured and should begin mailing as this *TUGboat* goes to press.

The 2010 release contains the same major items as in the past few years: T<sub>E</sub>X Live, MacT<sub>E</sub>X, proT<sub>E</sub>Xt, and a CTAN snapshot. More details at <http://tug.org/texcollection>.

Again, we welcome anyone's participation, from testing the final candidate release to core development. And thanks to all the many, many, people involved already at every level.

◇ Karl Berry  
<http://tug.org/TUGboat/Pres/>

---

## Editorial comments

Barbara Beeton

### Matthew Carter named MacArthur Fellow

Matthew Carter, type designer par excellence, is among the 23 Fellows named by the MacArthur Foundation for 2010. This award, sometimes called a “genius grant”, consists of \$500,000, no strings attached, over a period of five years. It is awarded “to talented individuals who have shown extraordinary originality and dedication in their creative pursuits and a marked capacity for self-direction.”

Carter's citation reads, in part,

... a master type designer who crafts letterforms of unequalled elegance and precision for a seemingly limitless range of applications and media. Throughout his career, which spans the migration of text from the printed page to the computer screen, he has pursued typographic solutions for the rapidly changing landscape of text-based communications. He has cut metal letterforms by hand in the manner invented over four centuries ago, created enduring works for machine- and phototype-setting, and produced many of the world's most widely used digital fonts.

Also among this year's Fellows is Nicholas Benson, stone carver, of the John Stevens Shop, Newport, Rhode Island. Like his father and grandfather, John and John, Jr. (“Fud”), Nicholas is a master of hand letter carving and a calligrapher. The family's inscriptional works embellish many important monuments and memorials in the U.S. In addition to his practice of this craft, Benson is “committed to teaching young artisans, who will create their own works and ensure that the legacy of this centuries-old artistic practice endures.” Benson is currently working in Washington, DC, on the new Martin Luther King National Memorial, on a site halfway between the Lincoln and Jefferson memorials. Nicholas is the second member of the Benson family to be recognized by the MacArthur Foundation; in 1986, his uncle Richard, a photographer and emeritus dean of the Yale University School of Art, was named a Fellow.

Carter and Benson are not the first “craftsmen of letters” to be recognized as MacArthur Fellows. Chuck Bigelow, creator with his partner, Kris Holmes, of the Lucida fonts, and now Melbert B. Cary Distinguished Professor at the Rochester Institute of Technology (the chair formerly occupied by Hermann Zapf), was named a Fellow in 1982.

More information about the award and the Fellows appears at [www.macfound.org](http://www.macfound.org).

### Indie Excellence Awards for self-published books

Increasingly, T<sub>E</sub>X users are choosing self-publishing. The Indie Excellence Awards, now in their fifth year, are sponsored by a marketing consultant with experience in bringing independently produced books to public attention.

All English-language books available for sale online or off, both e-books and in print, with publication dates from 2008–2011 inclusive are eligible. The deadline for submission is 31 March 2011. An entry fee is involved. Winners and finalists will be announced in May 2011. See [www.indieexcellence.com](http://www.indieexcellence.com).

### City maps made entirely of type

Have you ever tried to follow a street on a city map, only to be interrupted by cross-streets, or to lose your place when a very long street is named only at one end? A new approach to the art of the city map uses only type to delineate streets and other landmarks, with striking and wonderfully comprehensible results. The representation of Lake Michigan, off the Chicago shoreline, is truly ingenious and evocative.

See an illustration at [www.fastcodesign.com/1662468/infographic-of-the-day-city-maps-made-only-of-typefaces](http://www.fastcodesign.com/1662468/infographic-of-the-day-city-maps-made-only-of-typefaces).

### U&lc on line

Thanks to William Adams for spotting this item: [fonts.com](http://fonts.com) is making back issues of *U&lc* available as PDF scans.

The announcement and the first three issues are available here: [blog.fonts.com/2010/10/25/ulc-back-issues-to-be-made-available/](http://blog.fonts.com/2010/10/25/ulc-back-issues-to-be-made-available/)

William further comments, “I wish Adobe would do this with their *Font & Function* magazine...”

### Some “under-the-covers” uses of T<sub>E</sub>X

Jeffrey McArthur, on the `pdftex` mailing list ([lists.tug.org/pdftex](http://lists.tug.org/pdftex)), responded to an inquiry regarding the existence of T<sub>E</sub>X as a composition server with information about some very large projects with which he has been involved.

“Using UTF-8 encoding, and setting some characters active to handle the UTF-8 escape sequences [Jeffrey] typeset the Library of Congress Subject Matter headings. [...] The Library of Congress Subject Matter was particularly difficult because Unicode does not include all the glyphs needed [...]” The Library of Congress Subject Headings is a 4-volume work comprising around 7,000 pages.

The Leadership Directories Yellow Books ([www.leadershipdirectories.com](http://www.leadershipdirectories.com)), each directory being about a thousand pages, and the Warren Com-

munications Television and Cable Factbook were also prepared in a similar manner. All were composed using Plain T<sub>E</sub>X.

### Beyond literate programming

Another current discussion, in a thread “Callable T<sub>E</sub>X” on [texhax@tug.org](mailto:texhax@tug.org), has raised the topic of the evolution of computing and the lessening distinction between a program and a document.

James Quirk pointed out a newspaper article that appeared in the Manchester Guardian in February: “If you’re going to do good science, release the computer code too.” The premise: since so much scientific work is now being done by computers, the only way to be certain that conclusions are valid is to examine the programs that analyzed the data as well as the human logic written up in scientific reports.

A comment on the article by James also appears at the newspaper’s website. The URLs are too long to include here, but are linked from the thread in `texhax` in this message: [tug.org/pipermail/texhax/2010-October/015880.html](http://tug.org/pipermail/texhax/2010-October/015880.html).

James contends in his comment that not only the computer code but the process by which it is applied needs to be made visible, through “self-substantiating technical documents which allow the interested reader to sample the reported work first-hand, right down to its smallest detail.” He continues,

It has actually been possible to author self-substantiating documents for a good ten years now, but the effort involved has been too high to make them practical for mainstream scientific use. [...]

In fact, all the software pieces are now in place that one could, today, take classic research papers by the likes of von Neumann and Turing and turn them into multi-threaded, annotated affairs where the reader is walked through the research material and allowed to interact with computational examples that help convey the importance of the work. The basic idea being to produce “computational classics” that rival literary ones; entities that could be used to inspire generation, after generation, to want to seek careers in math, science, and engineering.

Such “computational classics” would also go a long way to defining computational standards, which at present are usually conspicuous by their absence.

- ◇ Barbara Beeton  
American Mathematical Society  
`tugboat (at) tug dot org`

## Hyphenation Exception Log

Barbara Beeton

This is the periodic update of the list of words that  $\TeX$  fails to hyphenate properly. The full list last appeared in *TUGboat* 16:1, starting on page 12, with updates in *TUGboat* 22:1/2, pp. 31–32; 23:3/4, pp. 247–248; 26:1, pp. 5–6; and 29:2, p. 239.

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

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

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

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

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

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

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

## Hyphenation for languages other than English

Patterns now exist for many languages other than English, including languages using accented alphabets. CTAN holds an extensive collection of patterns: see <http://mirror.ctan.org/language/hyphenation> and its subdirectories. A group of volunteers led by Mojca Miklavc and Arthur Reutenauer have created a comprehensive package of hyphenation patterns, called `hyph-utf8`; see <http://tug.org/tex-hyphen>.

### The List — English words

|                         |                             |
|-------------------------|-----------------------------|
| con-structible          | con-structible              |
| ep-stopdf               | eps-to-pdf                  |
| er-gonomic              | er-go-nom-ic                |
| er-gonom-i-cally        | er-go-nom-i-cally           |
| ethy-lamine             | eth-yl-am-ine               |
| ethy-late               | eth-yl-ate                  |
| ethynyl                 | ethy-nyl                    |
| ethyny-la-tion          | ethy-nyl-a-tion             |
| fron-tend               | front-end                   |
| in-pu-tenc              | input-enc                   |
| in-ter-vie-wee          | in-ter-view-ee              |
| leuko-cyte              | leu-ko-cyte                 |
| metaform                | meta-form                   |
| metham-phetamine        | meth-am-phet-a-mine         |
| methy-lam-mo-nium       | meth-yl-am-mo-nium          |
| methy-late              | meth-yl-ate                 |
| methy-la-tion           | meth-yl-a-tion              |
| methy-lene              | meth-yl-ene                 |
| minesweeper             | mine-sweeper                |
| nodelist                | node-list                   |
| nonzero                 | non-zero                    |
| ono-matopoeia           | ono-mat-o-poe-ia            |
| ono-matopo-etic         | ono-mat-o-po-et-ic          |
| parametriza-tion        | pa-ram-e-tri-za-tion        |
| prokary-ote             | pro-kary-ote                |
| prokary-otic            | pro-kary-ot-ic              |
| prostyle                | pro-style                   |
| provirus                | pro-virus                   |
| showhy-phens            | <code>\show-hy-phens</code> |
| tetra-butyl-lam-mo-nium | tetra-butyl-ammo-nium       |
| trans-parency(ies)      | trans-par-en-cy(ies)        |

### Names and non-English words used in English text

|               |                 |
|---------------|-----------------|
| Dorch-ester   | Dor-ches-ter    |
| Hoek-wa-ter   | Hoek-water      |
| Im-ageMag-ick | Image-Magick    |
| JavaScript    | Java-Script     |
| Minkowski     | Min-kow-ski     |
| Pythago-ras   | Py-thag-o-ras   |
| Pythagorean   | Py-thag-o-re-an |
| Schwei-d-nitz | Schweid-nitz    |
| Shored-itch   | Shore-ditch     |
| Tolch-ester   | Tol-ches-ter    |
| Vidi-assov    | Vid-ias-sov     |



## A story of *kpfonts*: Reaching the limits of NFSS

Christophe Caignaert

Like a bird on the wire,  
Like a drunk in a midnight choir,  
I have tried, in my way, to be free.

*LEONARD COHEN, Bird on the wire*

One day, some years ago, I was with DANIEL FLIPO, the author of the *letrine* package and the French module of *babel*.

He reminded me that L<sup>A</sup>T<sub>E</sub>X is community software, and, if I don't find what I want, I have to write it! Without him, probably, *kpfonts* wouldn't exist.

Greetings to him...

### 1 Before *kpfonts*

#### 1.1 I'm not a...

I have been a mathematics teacher in a high school in the north of France since 1980. My students are 19 or 20. I have been interested in computer science since the middle of the seventies.

I'm not a typographer and I'm not a T<sub>E</sub>X, or L<sup>A</sup>T<sub>E</sub>X, expert. I'm unable to program in the T<sub>E</sub>X language! Nothing fated me to become a font designer and package author... nothing at all!

#### 1.2 First steps with computer typesetting

The first computer I bought was an Apple IIe. Then I began writing some papers with the Apple Writer<sup>1</sup> software, obviously text and not math documents...

Then I bought an HP personal computer with an 8 Mhz 80286 processor! I began writing some mathematics using ChiWriter,<sup>2</sup> shareware at that time.

Some years later, I used a student release of Scientific Word, Scientific Workplace,<sup>3</sup> release 2.5. It was a private L<sup>A</sup>T<sub>E</sub>X editor with a limited wysiwyg formula editor. This was my first typesetting with good output. Scientific Word was good, but not very versatile, and month after month, I reached its limits. It was possible to insert any L<sup>A</sup>T<sub>E</sub>X command, but if it was unknown to SW, it would appear on the screen as a grey box. Some basic commands like `\sum\limits` in math resulted in a grey box for `\limits`. Because I'm never fully satisfied, I got more and more grey boxes in my documents with more and more (L<sup>A</sup>)T<sub>E</sub>X commands not interpreted

on screen. Therefore, I decided to forget it and I'm now using pdfT<sub>E</sub>X.

Perhaps it seems foolish to you, but during these years I was working alone to discover this software. In my high school, most math teachers are, still at this moment, writing math by hand; some use a too-well-known word processor, and I'm alone in looking for better output quality... with L<sup>A</sup>T<sub>E</sub>X.

And you know it's not easy to discover L<sup>A</sup>T<sub>E</sub>X alone!

#### 1.3 First interest in fonts

I have been interested in typography for a long time and I read that pdfT<sub>E</sub>X can use TrueType fonts. I followed the article of DAMIR RAKITYANSKY<sup>4</sup> to install my first *tff* fonts.

Thus, I discovered ligatures, kerning, metrics, virtual fonts: *pl*, *vpl*, *tfm*, *vf*, *fd*, *map* and *sty* files of the L<sup>A</sup>T<sub>E</sub>X font world, and also *tff*, *psf*, *afm* files coming from typography.

Because some users, mainly Windows users, never use a console or command line, I wrote a new paper, in French<sup>5</sup> and in English,<sup>6</sup> about installing *tff* fonts for pdfT<sub>E</sub>X and step-by-step instructions for those using T<sub>E</sub>XnicCenter. In addition, I built the necessary support files for many Windows *tff* fonts and free *tff* fonts available from a web site.<sup>7</sup>

I also made an artistic document for a local exhibition combining computer handwriting fonts with the meaning of the message, perhaps unfortunately for you, in French, called *Rendez-Vous*.<sup>8</sup>

Doing that work, I also discovered *the* font editor, *fontforge*, but also some other font editors: one of my friends works in typography and I used his professional computer during weekends.

I obviously discovered Bezier curves and the design of non-Metafont glyphs...

#### 1.4 First steps of the future *kpfonts*

My first font was called *Christophe*; it was my first attempt to alter *Palladio* (the URW *Palatino*) as a challenge ... for myself!

From the beginning, the principles were:

- very basic design, with a minimum number of Bezier curves,
- dynamic design with a marked diagonal force line from WSW to ENE.

<sup>4</sup> <http://www.radamir.com/tex/ttf-tex.htm>

<sup>5</sup> <http://c.caignaert.free.fr/Installer-Police-ttf.pdf>

<sup>6</sup> <http://c.caignaert.free.fr/Install-ttf-Font.pdf>

<sup>7</sup> <http://c.caignaert.free.fr/ttf.html>,

<http://c.caignaert.free.fr/ttf-english.html>

<sup>8</sup> <http://c.caignaert.free.fr/Rendez-Vous.pdf>

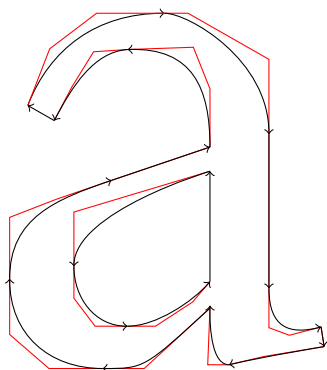
<sup>1</sup> [http://en.wikipedia.org/wiki/Apple\\_Writer](http://en.wikipedia.org/wiki/Apple_Writer)

<sup>2</sup> <http://en.wikipedia.org/wiki/ChiWriter>

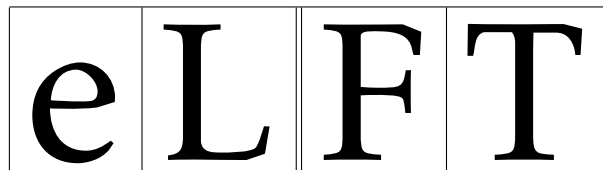
<sup>3</sup> [http://en.wikipedia.org/wiki/Scientific\\_WorkPlace](http://en.wikipedia.org/wiki/Scientific_WorkPlace)

We can see here the roman upright *a* of both *kpfonts* and some other font packages, and the approximate corresponding set of Bezier curves.

| Kpfonts | CM | Palatino | Utopia | Times |
|---------|----|----------|--------|-------|
| a       | a  | a        | a      | a     |



Next, you can see here the force line (sharp cut) and its symmetrical echo in *kpfonts*:



## 2 The development

I saw a beggar leaning on his wooden crutch,  
he said to me, "You must not ask for so much."

And a pretty woman leaning in her darkened door,  
she cried to me, "Hey, why not ask for more?"

LEONARD COHEN, *Bird on the wire*

### 2.1 Beginnings of the math set fonts

My first tests with math fonts was to use URW *Garamond* with the math symbols of *pxfonts*, the package I use at this moment in my documents. I called this *gxfnts*...

I discovered the global organisation of math fonts, with the main

- *operators*, like 0123 + - =  $\Gamma\Delta$ , and math operators like "sin",
- *letters* like *abc*  $\alpha\beta\gamma$ ,
- *symbols*, the basic symbols, like  $\rightarrow \mapsto \Rightarrow \exists$ ,
- *largesymbols*, the multi-size basic symbols, like

$$\Sigma \sum \int \int$$

and a lot of other things like AMS symbols, etc.

I also learned about the math alphabets, math delimiters. . .

I was impressed by the special tricks of DONALD KNUTH as

- long arrows made with minus sign and a regular arrow: '−' and '→' gives '→',
- long double arrows with equal sign and regular double arrow: '=' and '⇒' give '⇒',
- the use of the fake width and italic correction in math mode, width for subscript and italic correction for superscript,
- the famous *skewchar*, fake kerning to create the math accents:  $\tilde{a}$ .

It's like building the Golden Gate Bridge with three oz of spaghetti. . .

When the *gxfnts* package was in  $\beta$ -release, I sent a note to MICHEL BOVANI, the author of the *fourier* package, asking him his opinion.

Many thanks to him: he told me, with chosen words, it was *very bad*! And, even better, he told me *why*! For instance, the roman and greek letters of *gxfnts* were like cats and dogs. . .

Thus, I saw, at that moment, I had designed the Greek letters according to the design of the roman letters of *Christophe*. Even though the two projects were not linked at first, it was not so surprising: the same author and the same mood for design. . .

Therefore, it was obvious I had to combine these. . .

### 2.2 The *kpfonts* package

#### 2.2.1 The 1.0 release

Then I decided to make a *full* package of fonts, i.e. needing only one `\usepackage` to run.

It was 2005/04/20, my fiftieth birthday. Often, many people think that your life is behind you at 50! And perhaps I had to prove I was not a has-been!

From that moment I decided to write a comprehensive package including

- the roman, sans serif and teletype fonts,
- all the symbols including AMS symbols, "not" symbols, et al.,
- calligraphic and script alphabets,
- a *frenchstyle* math option needing upright uppercase and greek letters.

At that time, I had:

- the normal and bold text fonts including small caps, from *Christophe*,
- the slanted greek letters, from *gxfnts*,

and my todo list was cluttered:

- the sans-serif and teletype fonts,

- the textcomp symbols,
- the symbols, large symbols, AMS symbols,
- the upright greeks, calligraphic, script, full mathbb and fraktur alphabets,
- reading the *fontinst* doc file carefully,
- fixing the font's math dimensions: I keep the math font dimensions as DONALD KNUTH had them, except for the position of a subscript with no superscript, lower in *kpfonts* than CM.

I didn't realize the great deal of work needed at that time. The next two years were the busiest of the story. If you look at the `readme.txt` file, you can find:

Release 1.0 2007/04/20

It was my 52nd birthday. For many years, the first new set of fonts designed for L<sup>A</sup>T<sub>E</sub>X. I was very anxious about the feedback.

Since the beginning, *kpfonts* has supported the *frenchstyle* option, with upright uppercase roman and lowercase greek letters in math mode. Even if, at that time, I had no idea about the future of *kpfonts*, from the beginning, I thought I would propose some options to customize the typesetting.

### 2.2.2 Old style options

At that time, my birthday was obviously very important, because the next line of `readme.txt` is

Release 1.1 2007/05/04 New 'oldstyle' option, and `\sqrt` bug fixed.

only *fifteen days* later!

I had built the *oldstyle* option during the two previous years and it was almost ready when I uploaded the 1.0 release...

In fact, I think it is a good thing to build a package but a better thing to build a *different* package. A large set of options to customize the typesetting will make the *difference*. This appeared little by little during the work, like an obvious element.

In France, we have a well-known collection of books called *La Pleïade* using a *Garamond* font set with the *ct* and *st* old ligatures and a long tail *Q*.

I decided, because I liked them, to offer these possibilities as an option, with *oldstyle* numbers as the default. Later, asked by German users, I built an *oldstylenums* option without the extra ligatures (in Release 2.1 2008/03/21).

Here you can see the design of the ligature forms compared to the standard forms, using the *light* option:

| upright   | oldstyle  | upright   | oldstyle  |
|-----------|-----------|-----------|-----------|
| ct        | ct        | st        | st        |
| italic    | oldstyle  | italic    | oldstyle  |
| <i>ct</i> | <i>ct</i> | <i>st</i> | <i>st</i> |

The font dimensions of the superscripts are altered with *oldstyle* numbers in math mode taking their design into account.

Because the T1 encoding is full, I had to find two slots for the new *ct* and *st* old ligatures. I chose to use the slots of two Icelandic letters. I had nothing against the Icelandic people or their language, but I had to make a decision... Obviously, *kpfonts* sends a warning in this case.

### 2.3 The *kpfonts* package, release 2

It was the first major evolution of the package:  
Release 2.0 2008/01/01.

The new *f* ligatures, light fonts and very old style options appeared with this release. You can see that the second part of 2007 was a very intensive work period!

It was a new main number because, for me, the *light* option is *the major alteration* of *kpfonts*.

At that time, I thought, once again, that *kpfonts* was finished, except for the inevitable bug corrections...

#### 2.3.1 Light fonts

In my opinion, too-bold fonts are in bad taste. Using the facilities of the font editors and a good deal of work, I built lighter fonts with the same metrics, corresponding to the *light* option.

It was necessary to design again the mathematical symbols: when you have a line that is .7 pt in 10 pt, the light font would be .5 pt.

You can see below the normal weight and light, upright and italic *a*:









| upright | light | italic   | light    |
|---------|-------|----------|----------|
| a       | a     | <i>a</i> | <i>a</i> |

What's more, it's not insignificant to save up to 20% toner when printing...

### 2.3.2 New *f* ligatures

A ligature is the way to combine two characters into one. The most common ligatures with T<sub>E</sub>X are the *f* ligatures: *ff*, *fi*, *fl*, *ffi* and *ffl*.

There are different ways to design them. See examples below with the *fi* ligature:

|   |   |   |   |
|---|---|---|---|
| URW<br>Garamond   | Kpfonts 1.x   | Kpfonts   | Palladio  |
|  |  |  |  |
| URW<br>Garamond   | Kpfonts 1.x   | Kpfonts   | Palladio  |
|  |  |  |  |




At first, I made a bad choice, like a bridge, as with *Garamond* for instance. It was a bad choice relative to the design of the *f* of my fonts: the effect was not good because of the short terminal of its ascender. Thus, I decided to change it. It was necessary to change the design of the ascenders of these ligatures.

Note the old and new *fi* of *kpfonts* and the almost fake ligature in upright *URW Palladio* used by the *palatino*, *pxfonts*, and *mathpazo* packages.

### 2.3.3 Very old style options

In very old documents, instead of the round *s*, we find a long *f*, except at the end of the word. I couldn't find any package to typeset text and math with the long *f*. Then, I decided to built the necessary files and to offer these possibilities. It was done with *ReRelease 2.1* 2008/03/21.

For instance, here is *st* using italic shape and light fonts; you can see I also installed new ligatures:

|   |   |   |
|---|---|---|
| Default   | Old style   | Very old style  |
|  |  |  |





At this moment, the idea to make a package with a large set of options to customize the typesetting was definitely established.

### 2.3.4 Large small capitals

It's interesting in a font package to have real small capitals and not fakes... From the beginning, I designed some small caps. In fact, I designed *very small* small caps, approximately as high as an *x*. I like it because they are different!

It's also not usual because in many cases, the small caps are fakes, scaled uppercase indeed. Don't forget that a fake seems not too bad if the scaling is not too strong, i.e. if the small caps are not too small! This is another of the reasons why small caps are usually rather large.

I decided then to work on a large small caps set of fonts. Indeed, the font editors are able to "blend" some fonts. Blending the existing *small* small caps and usual uppercase letters gives a good design to begin the work. It was done in *ReRelease 2.2* 2008/05/21, shown here using the *light* option.

|  |  |  |  |
|--|--|--|--|
| lowercase  | small cap  | large small cap  | uppercase  |
|  |  |  |  |





Thus, *kpfonts* has two sizes of small caps. It's very rare and even the very extensive OpenType font file doesn't allow for it!

As in the present article, I usually use small small caps for people's names and large small caps for acronyms.

### 2.3.5 The lowercase *q* record

Usually, with a given font set, you get four designs for a letter: upright and italic, normal and bold. If there are true small caps, you get also them in normal and bold, six designs in this case.

For the lowercase *q* in *kpfonts*, you get forty roman designs, and then more with the sans-serif and teletype fonts! Perhaps a record, even though there is no italic small caps *q*. Let's start with the default designs:

|   |   |   |   |
|---|---|---|---|
| upright   | bold  | italic  | bold  |
|  |  |  |  |

Light:

| upright | bold | italic | bold |
|---------|------|--------|------|
| q       | q    | q      | q    |

Small caps:

| default | bold | large | bold |
|---------|------|-------|------|
| Q       | Q    | Q     | Q    |

Light small caps:

| default | bold | large | bold |
|---------|------|-------|------|
| Q       | Q    | Q     | Q    |

Long tail small caps:

| default | bold | large | bold |
|---------|------|-------|------|
| Q       | Q    | Q     | Q    |

Long tail light small caps:

| default | bold | large | bold |
|---------|------|-------|------|
| Q       | Q    | Q     | Q    |

We get all these glyphs with the lowercase *q*!

### 2.3.6 No *f* ligatures

The option `nofligatures` appeared with `Release 2.3 2008/09/09`, requested by users who didn't like the ligatures.

With some packages, or modern T<sub>E</sub>X-based engines, you can disable these ligatures but the result can be ugly:

| Times<br><code>f{i}</code> | Utopia<br><code>f{i}</code> | Kpfonts<br><code>f{i}</code> | Kpfonts<br><code>nofligatures</code> |
|----------------------------|-----------------------------|------------------------------|--------------------------------------|
| fi                         | fi                          | fi                           | fi                                   |

| Times<br><code>f{i}</code> | Utopia<br><code>f{i}</code> | Kpfonts<br><code>f{i}</code> | Kpfonts<br><code>nofligatures</code> |
|----------------------------|-----------------------------|------------------------------|--------------------------------------|
| <i>fi</i>                  | <i>fi</i>                   | <i>fi</i>                    | <i>fi</i>                            |

And, don't forget it's worse at normal size! With upright Times, *f* and *i* seem incompatible, like cats and dogs, and with Utopia, the ascender of the *f* and the dot of the *i* are too close and don't fit together.

You can see that the result is not too bad with my fonts, but, I preferred to shorten the ascender of the *f* letter in this case. In my opinion, the look is better at normal size!

### 2.3.7 Slanted small caps

In the L<sup>A</sup>T<sub>E</sub>X new font selection scheme (NFSS), small caps and slanted (or italic) are *shapes*. The result is the impossibility of getting slanted small caps.

Installing slanted small caps, a new shape `scsl`, requires only some lines in the installation file used by `fontinst` program, and also some lines in the `sty` file. Here's an example:

*Everybody, including TED SLANTED, can see it's better than JACK UPRIGHT does usually!*

Slanted small caps also appeared with `Release 2.3 2008/09/09`. Later, a new option `easyscsl` allows you to fit together `\textsc` and `\textsl`. It's an option because, if you use `\textsc{\textsl{...}}` with other fonts, you get some edge effect. This option appeared with `Release 3.3 2010/04/20`, and sent a warning to the console. This point will be discussed in a later section.

### 2.3.8 Math fonts during this time

For some time now, we have been speaking about text fonts but math typesetting is also going on!

First, the `oldstylemath`, `veryoldstylemath` and `oldstylenumsmath` appeared at the same time as the text equivalent.

As of `Release 2.2`, you can use `narrowiints` option, For `\displaystyle\iiint dx, dy, dz`, let's see the output:

| default           | <i>narrowiints</i> |
|-------------------|--------------------|
| $\iiint dx dy dz$ | $\iiint dx dy dz$  |

And with Release 2.3, the *partialup* option is added. For `\dfrac{\partial z}{\partial x}`, the output is:

| default                         | <i>partialup</i>                |
|---------------------------------|---------------------------------|
| $\frac{\partial z}{\partial x}$ | $\frac{\partial z}{\partial x}$ |

## 2.4 The 3.0 release: new text kerning and math accents

### 2.4.1 New kerning

There were some inherited defaults in *kpfonts*, and, even at that time, we could see that the main problem was the kernings. One of the first lines of the `Readme` file is

Release 1.11 2007/06/03 Correct bad kernings of 'quote' symbols

It proves that, from the beginning, the kerning was a problem. Perhaps it's the biggest challenge for a beginner! The kerning by pairs is the way to tighten or spread two characters depending on their exact design. For instance, see *Ye* with and without kerning, here using the *light* option:

| with      | without   |
|-----------|-----------|
| <i>Ye</i> | <i>Ye</i> |

The font editors offer a lot of possibilities. One of these is automatic kerning. Usually you have to choose:

- the left and right characters to kern,
- the required space between two characters,
- the technique: minimum distance, average distance, average weight,
- the exceptions: numerals, lowercase-uppercase: in 'L<sup>A</sup>T<sub>E</sub>X', for instance, there is a kerning *T-e* but no kerning *a-T*...
- the equivalents, *o* and *ô* have often the same kerning...

These programs do their best but are regrettably not very good. And a beginner like me was too confident in their results. Even if, at the time, I corrected all the generated kernings by hand, I was too

confident about the basic results of the automatic kernings...

Some users protest rightly about incoherent kerning. I asked on *fctt*, the French version of *ctt*, and everybody thought new kernings would be a good thing although it can change the typesetting. I decided to work on it...

At the same time, subscript and superscript position, i.e. width and italic correction, of all the math alphabets were revisited. It's a very long hard job, with a large set of tests and much reinstallation of *kpfonts*. During these six months, I produced, with *fontinst* and batch files, at least 200 000 files...

It was available on CTAN as of Release 3.0 2009/03/03, and I thought now the work was not too far from being good fonts. Therefore, the new main number version.

See for instance the *Av* kerning in upright shape and *To* in italic with the 1.xx or 2.xx release versus the same with 3.xx (default fonts here).

| before 3.0 | 3.0 and after | no kerning |
|------------|---------------|------------|
| <i>Av</i>  | <i>Av</i>     | <i>Av</i>  |

| before 3.0 | 3.0 and after | no kerning |
|------------|---------------|------------|
| <i>To</i>  | <i>To</i>     | <i>To</i>  |

Scaled this much, the first may not appear to spread, but it's the case at normal size. That's the reason why the first kernings are too strong. Then, I was working on a screen... Thus I bought a laser printer and work now with printed tests!

### 2.4.2 New math accents and *widermath* option

Also with the 3.0 release, I installed new math accents such as `\widearc`, as in some other packages. Here are some examples:

| <code>\widearc</code>   | <code>\widearcarrow</code> |
|-------------------------|----------------------------|
| $\widearc{M_0 M_1}$     | $\widearcarrow{M_0 M_1}$   |
| <code>\wideparen</code> | <code>\widering</code>     |
| $\wideparen{M_0 M_1}$   | $\widering{M_0 M_1}$       |

You get also the new option *widermath*. The object is to provide slightly wider math typesetting, particularly for users working with 9 or 10 pt as the basic font size. Small sizes need proportionally bigger spaces...

### 2.4.3 *amsmath* options

Release 3.1 2009/05/20 offers the possibility to use the options of *amsmath* as options of *kpfonts*. This affects the basic and AMS math fonts and also the special math fonts of *kpfonts*...

These too-little-known options affect the default position of subscript in integral or summation symbols. To get more information, see the documentation of the AMS or *kpfonts* packages.

### 2.4.4 Sans-serif math versions

The last major evolution *kpfonts* was Release 3.2 2010/03/03 allowing math typesetting using sans-serif fonts. You can do it with a new option, *sfmath*, or with the new math versions *sf* and *boldsf*. Obviously, for full support, you also get both *rm* and *boldrm* math versions.

Some default symbols are serified, like  $\sum$ ; thus they have a new design, as you can see:

| roman                         | sans-serif                    |
|-------------------------------|-------------------------------|
| $\sum_{p=0}^n$ $\sum_{p=0}^n$ | $\sum_{p=0}^n$ $\sum_{p=0}^n$ |

In addition, I designed some sans-serif greek letters, uppercase and lowercase, slanted and upright:

| roman   | sans-serif  |
|---|---|
| $\alpha$ $\beta$ $\alpha$ $\beta$ $\Gamma$ $\Psi$ $\Gamma$ $\Psi$ | $\alpha$ $\beta$ $\alpha$ $\beta$ $\Gamma$ $\Psi$ $\Gamma$ $\Psi$ |

In case you are getting slightly sleepy reading this, let me explain exactly what it means. For instance, when you type  $\backslash\alpha$ , depending on the options and math version, you can get any of 12 different designs: normal or bold ( $\times 2$ ); upright or slanted ( $\times 2$ ); default or light roman; or sans-serif ( $\times 3$ )!

## 2.5 Special tricks

In fact, I don't like to have special tricks in a package, but I still use this possibility sometimes!

- To get the *veryoldstyle* *s*, usually at the end of a word, I use a classic fake ligature  $s=$ .

- *narrowiints*

In the *kpfonts.sty* file, we find this code:

```
\re@DeclareMathSymbol{\iintop}{\mathop}{\largesymbolsA}{\narrowiints33}
```

where

- $\backslash\text{narrowiints}$  is 1 if the *narrowiints* option is selected, empty if not, and,
- the default  $\backslash\text{iint}$  symbol is decimal 33 and the narrower one is decimal 133.

- Long tail Q is called:

- *Qoldstyle* in the *afm* files and the *etx* files used by *fontinst* and,
- *Q* in the *pdf* and *enc* files.

Thus, in *pdf* and *ps* output, it's *Q* and the search functions of *Acroread* and *Ghostscript* can find it in any case...

I use the same trick for the *veryoldstyle* long *s*.

## 3 Some examples

### 3.1 Text

I use the example of *testfont.tex* and the *L<sup>A</sup>T<sub>E</sub>X Companion*, slightly altered when using the *veryoldstyle* option.

#### 3.1.1 Default

For the price of £45, almost anything can be found floating in fields. ¡THE DAZED BROWN FOX QUICKLY GAVE 12345-67890 JUMPS! — ¿But aren't Kafka's Schloß and Æsop's Œuvres often naïve vis-à-vis the dæmonic phœnix's official rôle in fluffy soufflés?

*For the price of £45, almost anything can be found floating in fields. ¡THE DAZED BROWN FOX QUICKLY GAVE 12345-67890 JUMPS! — ¿But aren't Kafka's Schloß and Æsop's Œuvres often naïve vis-à-vis the dæmonic phœnix's official rôle in fluffy soufflés?*

#### 3.1.2 Options *oldstylenums* and *light* or *textlight*

For the price of £45, almost anything can be found floating in fields. ¡THE DAZED BROWN FOX QUICKLY GAVE 12345-67890 JUMPS! — ¿But aren't Kafka's Schloß and Æsop's Œuvres often naïve vis-à-vis the dæmonic phœnix's official rôle in fluffy soufflés?

*For the price of £45, almost anything can be found floating in fields. ¡THE DAZED BROWN FOX QUICKLY GAVE 12345-67890 JUMPS! — ¿But aren't Kafka's Schloß and Æsop's Œuvres often naïve vis-à-vis the dæmonic phœnix's official rôle in fluffy soufflés?*

### 3.1.3 Option *nofligatures*

For the price of £45, almost anything can be found floating in fields. ¡THE DAZED BROWN FOX QUICKLY GAVE 12345-67890 JUMPS! — ¿But aren't Kafka's Schloß and Æsop's Œuvres often naïve vis-à-vis the dæmonic phœnix's official rôle in fluffy soufflés?

*For the price of £45, almost anything can be found floating in fields. ¡THE DAZED BROWN FOX QUICKLY GAVE 12345-67890 JUMPS! — ¿But aren't Kafka's Schloß and Æsop's Œuvres often naïve vis-à-vis the dæmonic phœnix's official rôle in fluffy soufflés?*

### 3.1.4 Option *oldstyle*

For the price of £45, almost anything can be found floating in fields. ¡THE DAZED BROWN FOX QUICKLY GAVE 12345-67890 JUMPS! — ¿But aren't Kafka's Schloß and Æsop's Œuvres often naïve vis-à-vis the dæmonic phœnix's official rôle in fluffy soufflés?

*For the price of £45, almost anything can be found floating in fields. ¡THE DAZED BROWN FOX QUICKLY GAVE 12345-67890 JUMPS! — ¿But aren't Kafka's Schloß and Æsop's Œuvres often naïve vis-à-vis the dæmonic phœnix's official rôle in fluffy soufflés?*

### 3.1.5 Option *veryoldstyle* and *light* or *textlight*

For the price of £45, almost anything can be found floating in fields. ¡THE DAZED BROWN FOX QUICKLY GAVE 12345-67890 JUMPS! — ¿But aren't Kafka's Schloß and Æfop's Œuvres often naïve vis-à-vis the dæmonic phœnix's official rôle in fluffy soufflés?

*For the price of £45, almost anything can be found floating in fields. ¡THE DAZED BROWN FOX QUICKLY GAVE 12345-67890 JUMPS! — ¿But aren't Kafka's Schloß and Æfop's Œuvres often naïve vis-à-vis the dæmonic phœnix's official rôle in fluffy soufflés?*

### 3.1.6 Quiz

Exercise: find the minimal set of package options that are used in each of these cases. Except when using the *veryoldstyle* option, the source file is always the same, sometimes upright, sometimes italic.

1. A.QUEER says: making 29 **active** characters if *definitely nasty!*
2. A.QUEER says: making 29 **active** characters is **definitely nasty!**
3. A.QUEER says: making 29 **active** characters is *definitely nasty!*

4. A.QUEER says: making 29 **active** characters is *definitely nasty!*
5. A.QUEER says: making 29 **active** characters is **definitely nasty!**
6. A.QUEER says: making 29 **active** characters is **definitely nasty!**
7. A.QUEER says: making 29 **active** characters is **definitely nasty!**
8. A.QUEER says: making 29 **active** characters is **definitely nasty!**
9. A.QUEER says: making 29 **active** characters is *definitely nasty!*
10. A.QUEER says: making 29 **active** characters is *definitely nasty!*

*Read the solution at the end of the article!*

*If you were very attentive, you can get 10 points!*

## 3.2 Math

The figures on the following pages show math samples. These also use an example from the *L<sup>A</sup>T<sub>E</sub>X Companion*...

## 3.3 This document

This article uses only the *textlight* option. Obviously, in some parts, the options described are simulated using `\fontfamily`...

In the math examples, I use two *special tricks* to get the narrow `\iiint` and the upright `\partial` symbol.

In both the text and math examples, the output is scaled to the available line length.

Personal names are in default small caps, and acronyms are in large small caps.

## 4 The limits of NFSS

### 4.1 Non-existing features

Some features of *kpfonts* don't exist in the new font selection scheme:

- Two sizes of small caps:
  - The commands `\textothersc{...}` and `\otherscshape` allow you to use both sizes. They are often used in this document.
  - The option *largesmallcaps* changes the default small caps size. Then, you can use standard commands for large small caps!
- slanted small caps:
  - The following commands allow you to use the slanted small capitals:
    - `\textscsl{...}`
    - `\scslshape`
    - `\textotherscsl{...}`
    - `\otherscslshape`



## 1 Sample page of mathematical typesetting

First some large operators both in text:  $\iiint_{\mathbb{Q}} f(x, y, z) dx dy dz$  and  $\prod_{\gamma \in \Gamma_{\mathbb{C}}} \partial(\tilde{X}_{\gamma})$ ; and also on display:

$$\begin{aligned} \iiint_{\mathbb{Q}} f(w, x, y, z) dw dx dy dz &\leq \oint_{\partial \mathbb{Q}} f' \left( \max \left\{ \frac{\|w\|}{|w^2 + x^2|}, \frac{\|z\|}{|y^2 + z^2|}, \frac{\|w \oplus z\|}{\|x \oplus y\|} \right\} \right) \\ &\approx \bigsqcup_{\mathbb{Q} \in \bar{\mathbb{Q}}} \left[ f^* \left( \frac{f(\mathbb{Q}(t))}{\sqrt{1-t^2}} \right) \right]_{t=\alpha}^{t=\vartheta} \end{aligned} \quad (1)$$

For  $x$  in the open interval  $] -1, 1[$  the infinite sum in Equation (2) is convergent; however, this does not hold throughout the closed interval  $[-1, 1]$ .

$$(1-x)^{-k} = 1 + \sum_{j=1}^{\infty} (-1)^j \binom{k}{j} x^j \quad \text{for } k \in \mathbb{N}; k \neq 0. \quad (2)$$

Figure 1: Default

## 1 Sample page of mathematical typesetting

First some large operators both in text:  $\iiint_{\mathbb{Q}} f(x, y, z) dx dy dz$  and  $\prod_{\gamma \in \Gamma_{\mathbb{C}}} \partial(\tilde{X}_{\gamma})$ ; and also on display:

$$\begin{aligned} \iiint_{\mathbb{Q}} f(w, x, y, z) dw dx dy dz &\leq \oint_{\partial \mathbb{Q}} f' \left( \max \left\{ \frac{\|w\|}{|w^2 + x^2|}, \frac{\|z\|}{|y^2 + z^2|}, \frac{\|w \oplus z\|}{\|x \oplus y\|} \right\} \right) \\ &\approx \bigsqcup_{\mathbb{Q} \in \bar{\mathbb{Q}}} \left[ f^* \left( \frac{f(\mathbb{Q}(t))}{\sqrt{1-t^2}} \right) \right]_{t=\alpha}^{t=\vartheta} \end{aligned} \quad (1)$$

For  $x$  in the open interval  $] -1, 1[$  the infinite sum in Equation (2) is convergent; however, this does not hold throughout the closed interval  $[-1, 1]$ .

$$(1-x)^{-k} = 1 + \sum_{j=1}^{\infty} (-1)^j \binom{k}{j} x^j \quad \text{for } k \in \mathbb{N}; k \neq 0. \quad (2)$$

Figure 2: Options *lightmath* and *narrowiints*

## 1 Sample page of mathematical typesetting

First some large operators both in text:  $\iiint_{\mathcal{Q}} f(x, y, z) dx dy dz$  and  $\prod_{\gamma \in \Gamma_{\bar{c}}} \partial(\tilde{X}_{\gamma})$ ; and also on display:

$$\begin{aligned} \iiint_{\mathcal{Q}} f(w, x, y, z) dw dx dy dz &\leq \oint_{\partial \mathcal{Q}} f' \left( \max \left\{ \frac{\|w\|}{|w^2 + x^2|}, \frac{\|z\|}{|y^2 + z^2|}, \frac{\|w \oplus z\|}{\|x \oplus y\|} \right\} \right) \\ &\approx \bigcup_{\mathcal{Q} \in \bar{\mathcal{Q}}} \left[ f^* \left( \frac{f(\mathcal{Q}(t))}{\sqrt{1-t^2}} \right) \right]_{t=\alpha}^{t=\vartheta} \end{aligned} \quad (1)$$

For  $x$  in the open interval  $] -1, 1[$  the infinite sum in Equation (2) is convergent; however, this does not hold throughout the closed interval  $[-1, 1]$ .

$$(1-x)^{-k} = 1 + \sum_{j=1}^{\infty} (-1)^j \binom{k}{j} x^j \quad \text{for } k \in \mathbb{N}; k \neq 0. \quad (2)$$

Figure 3: Options *nofligatures* and *uprightgreek*s

## 1 Sample page of mathematical typesetting

First some large operators both in text:  $\iiint_{\mathcal{Q}} f(x, y, z) dx dy dz$  and  $\prod_{\gamma \in \Gamma_{\bar{c}}} \partial(\tilde{X}_{\gamma})$ ; and also on display:

$$\begin{aligned} \iiint_{\mathcal{Q}} f(w, x, y, z) dw dx dy dz &\leq \oint_{\partial \mathcal{Q}} f' \left( \max \left\{ \frac{\|w\|}{|w^2 + x^2|}, \frac{\|z\|}{|y^2 + z^2|}, \frac{\|w \oplus z\|}{\|x \oplus y\|} \right\} \right) \\ &\approx \bigcup_{\mathcal{Q} \in \bar{\mathcal{Q}}} \left[ f^* \left( \frac{f(\mathcal{Q}(t))}{\sqrt{1-t^2}} \right) \right]_{t=\alpha}^{t=\vartheta} \end{aligned} \quad (1)$$

For  $x$  in the open interval  $] -1, 1[$  the infinite sum in Equation (2) is convergent; however, this does not hold throughout the closed interval  $[-1, 1]$ .

$$(1-x)^{-k} = 1 + \sum_{j=1}^{\infty} (-1)^j \binom{k}{j} x^j \quad \text{for } k \in \mathbb{N}; k \neq 0. \quad (2)$$

Figure 4: Options *lightmath* and *partialup*

## 1 Sample page of mathematical typesetting

First some large operators both in text:  $\iiint_{\mathbb{Q}} f(x, y, z) dx dy dz$  and  $\prod_{\gamma \in \Gamma_{\mathbb{C}}} \partial(\tilde{X}_{\gamma})$ ; and also on display:

$$\begin{aligned} \iiint_{\mathbb{Q}} f(w, x, y, z) dw dx dy dz &\leq \oint_{\partial \mathbb{Q}} f' \left( \max \left\{ \frac{\|w\|}{|w^2 + x^2|}, \frac{\|z\|}{|y^2 + z^2|}, \frac{\|w \oplus z\|}{\|x \oplus y\|} \right\} \right) \\ &\approx \bigcup_{\mathbb{Q} \in \mathbb{Q}} \left[ f^* \left( \frac{f(\mathbb{Q}(t))}{\sqrt{1-t^2}} \right) \right]_{t=\alpha}^{t=\vartheta} \end{aligned} \quad (1)$$

For  $x$  in the open interval  $] -1, 1[$  the infinite sum in Equation (2) is convergent; however, this does not hold throughout the closed interval  $[-1, 1]$ .

$$(1-x)^{-k} = 1 + \sum_{j=1}^{\infty} (-1)^j \binom{k}{j} x^j \quad \text{for } k \in \mathbb{N}; k \neq 0. \quad (2)$$

Figure 5: Option *sfmathbb*

## 1 Sample page of mathematical typesetting

First some large operators both in text:  $\iiint_{\mathbb{Q}} f(x, y, z) dx dy dz$  and  $\prod_{\gamma \in \Gamma_{\mathbb{C}}} \partial(\tilde{X}_{\gamma})$ ; and also on display:

$$\begin{aligned} \iiint_{\mathbb{Q}} f(w, x, y, z) dw dx dy dz &\leq \oint_{\partial \mathbb{Q}} f' \left( \max \left\{ \frac{\|w\|}{|w^2 + x^2|}, \frac{\|z\|}{|y^2 + z^2|}, \frac{\|w \oplus z\|}{\|x \oplus y\|} \right\} \right) \\ &\approx \bigcup_{\mathbb{Q} \in \mathbb{Q}} \left[ f^* \left( \frac{f(\mathbb{Q}(t))}{\sqrt{1-t^2}} \right) \right]_{t=\alpha}^{t=\vartheta} \end{aligned} \quad (1)$$

For  $x$  in the open interval  $] -1, 1[$  the infinite sum in Equation (2) is convergent; however, this does not hold throughout the closed interval  $[-1, 1]$ .

$$(1-x)^{-k} = 1 + \sum_{j=1}^{\infty} (-1)^j \binom{k}{j} x^j \quad \text{for } k \in \mathbb{N}; k \neq 0. \quad (2)$$

Figure 6: Options *lightmath*, *fulloldstylenums* and *widernmath*

## 1 Sample page of mathematical typesetting

First some large operators both in text:  $\iiint_{\mathcal{Q}} f(x, y, z) dx dy dz$  and  $\prod_{\gamma \in \Gamma_{\bar{c}}} \partial(\tilde{X}_{\gamma})$ ; and also on display:

$$\begin{aligned} \iiint_{\mathcal{Q}} f(w, x, y, z) dw dx dy dz &\leq \oint_{\partial \mathcal{Q}} f' \left( \max \left\{ \frac{\|w\|}{|w^2 + x^2|}, \frac{\|z\|}{|y^2 + z^2|}, \frac{\|w \oplus z\|}{\|x \oplus y\|} \right\} \right) \\ &\approx \bigcup_{\mathcal{Q} \in \bar{\mathcal{Q}}} \left[ f^* \left( \frac{|\mathcal{Q}(t)|}{\sqrt{1-t^2}} \right) \right]_{t=\alpha}^{t=\beta} \end{aligned} \quad (1)$$

For  $x$  in the open interval  $] -1, 1[$  the infinite sum in Equation (2) is convergent; however, this does not hold throughout the closed interval  $[-1, 1]$ .

$$(1-x)^{-k} = 1 + \sum_{j=1}^{\infty} (-1)^j \binom{k}{j} x^j \quad \text{for } k \in \mathbb{N}; k \neq 0. \quad (2)$$

Figure 7: Option *sfmath*

## 1 Sample page of mathematical typesetting

First some large operators both in text:  $\iiint_{\mathcal{Q}} f(x, y, z) dx dy dz$  and  $\prod_{\gamma \in \Gamma_{\bar{c}}} \partial(\tilde{X}_{\gamma})$ ; and also on display:

$$\begin{aligned} \iiint_{\mathcal{Q}} f(w, x, y, z) dw dx dy dz &\leq \oint_{\partial \mathcal{Q}} f' \left( \max \left\{ \frac{\|w\|}{|w^2 + x^2|}, \frac{\|z\|}{|y^2 + z^2|}, \frac{\|w \oplus z\|}{\|x \oplus y\|} \right\} \right) \\ &\approx \bigcup_{\mathcal{Q} \in \bar{\mathcal{Q}}} \left[ f^* \left( \frac{|\mathcal{Q}(t)|}{\sqrt{1-t^2}} \right) \right]_{t=\alpha}^{t=\beta} \end{aligned} \quad (1)$$

For  $x$  in the open interval  $] -1, 1[$  the infinite sum in Equation (2) is convergent; however, this does not hold throughout the closed interval  $[-1, 1]$ .

$$(1-x)^{-k} = 1 + \sum_{j=1}^{\infty} (-1)^j \binom{k}{j} x^j \quad \text{for } k \in \mathbb{N}; k \neq 0. \quad (2)$$

Figure 8: Options *sfmath*, *narrowiints*, *uprightgreek*s and *partialup*

- You can also use the *easyscs* option or the *slantsc* package to get slanted small caps as expected: `\textsc{\textsl{...}}`. But, you have to redefine these commands and the result can be disappointing!

```
\documentclass{minimal}
\usepackage{palatino}
\usepackage{slantsc}
\begin{document}
\textsl{\textsc{Hello}}
\end{document}
```

gives you 2 warnings... and the output is an upright “Hello” in *palatino*! This is the reason that the *easyscs* option of *kpfonts* gives an explicit warning.

- The option *largesmallcaps* also affects the default slanted small caps size.
- light variant fonts:
  - Because of the edge effects described below, there are no commands to switch between default and light fonts.

For instance, you have to redefine commands like `\textit`: therefore, you want the *italic* of the **actual** font, not the default!

- But the option *rmx* allows you to use these fonts without the usual

```
\fontfamily{...}\selectfont!
```

The corresponding table:

| option       | weight | <i>rmx</i> |
|--------------|--------|------------|
| <i>light</i> | m      | l          |
|              | m      | m          |
| <i>light</i> | b/bx   | sb/sbx     |
|              | b/bx   | b/bx       |

## 4.2 Exponential number of files

### 4.2.1 Usual case

In most cases, when you have a font family, like *URW Garamond*, you get basically 4 fonts:

- upright or italic, and,
- normal or bold.

Thus

- 4 pairs *pf*/*af*, the design of characters and the metrics, and/or,
- 4 *ttf* or *otf* files including design and metrics.

( $\LaTeX$ ) doesn’t need the design of the characters, they need only the metrics, the *tfm* files to build the *dvi*. It’s one of the reasons why *dvi* files are small.

A *dvi* viewer or *dvips* (or equivalent) or  $\text{pdf}\TeX$  does need the actual characters to produce the final document, of course.

We describe briefly the chain of events to find the good metrics. Now, imagine you are  $\TeX$  (it’s easy if you try):

- You have an active family and encoding, for instance *jkp* and T1, i.e. default *kpfonts* family and Cork encoding,
- you read the `t1jpk.fd` file, for font definitions,
- you have an active shape and weight, for instance *it* and *n*, i.e. italic and normal weight,
- you read the line:
 

```
\DeclareFontShape{T1}{jpk}{m}{it}
      {<-> jkpmi8t}{}
      in the fd file. jkpmi8t is the needed tfm file, including metrics, ligatures and kerning.
```

Usual cases, like *Garamond*, require less than 50 files for an OT1 and T1 installation...

### 4.2.2 *kpfonts* case

For “hackers”, special use or curiosity, look at the rules to build the corresponding family names:

|                   |   |
|-------------------|---|
| <b>roman</b>      | <code>jkp[l,x][k][f][osn,os,vos]</code> |
| <b>sans serif</b> | <code>jkps[k][f][osn,os,vos]</code>     |
| <b>teletype</b>   | <code>jkpt[osn,os,vos]</code>           |

with the corresponding options:

|                     |   |
|---------------------|---|
| <b>l, x</b>         | light, <i>rmx</i>                           |
| <b>k</b>            | <i>largesmallcaps</i>                       |
| <b>f</b>            | <i>nofligatures</i>                         |
| <b>osn, os, vos</b> | <i>oldstylenums, oldstyle, veryoldstyle</i> |

For the roman fonts, because we can choose between OT1 and T1 encoding, we have 72 families, excluding the TS1 ones for *textcomp*... The total number of families is 187 in the 3.12 release!

Most of the roman families have 15 *tfm* metrics:

- upright, italic, small caps, slanted, slanted small caps,
- each in normal, bold and bold extended...

But each *tfm* corresponds here to a *vf*, virtual font, file because there is no direct link between these *tfm* files and a *pf* file!

In the 3.12 release, we get

- 668 virtual font, *vf*, files,
- 858 tex font metric, *tfm*, files.

The *kpfonts* TDS tree has a total of 1,875 files...

I’m on my own and it’s impossible for me to be ensure with any probability that there is no bug! Indeed, writing this paper, I found one bug: *light* and *veryoldstyle* medium weight font were not *light* but the default!

### 4.2.3 About a new option

A new option, long tail Q without special ligatures *ct* and *st*, but with old style numbers or not, was requested by a user. This would mean 240 more *tfm* files, 240 more *vf* files and 24 more *fd* files, this just for T1 encoding and roman fonts.

For sans serif fonts, it's 96 *tfm*, 96 *vf* and 8 *fd* new files. No action and no more files for teletype fonts!

The number required by the OT1 encoding is the same, for a complete sum of 1,408 new files... Increasing the total number of *kpfonts*' files about 75%!

You see here the explicit exponential effect!

In fact, this option would not be hard to install (2 new *etx* files, the encoding files for *fontinst*, and some new lines in the installation file), but I don't agree with the request because there are already commands `\othertailQ` and `\othertailscq` to do the work...

### 4.2.4 Last way to be free

And if you want some options to choose freely:

- classic L<sup>A</sup>T<sub>E</sub>X *f* ligatures or not,
- *cl* ligature or not,
- *sl* ligature or not,
- oldstyle or lining numbers,
- round *s* or long *f*,
- long tail *Q* or classic *Q*,

if I'm not wrong it's about 20,000 files more...

The object of *kpfonts* is not to increase indefinitely the number of files on your hard disk!

The object of *kpfonts* is not to be in the Guinness book!

I don't think it's sensible to exceed 2,000 files in a package, even if it's possible!

To go further, to be free, I think somebody has to build some *otf* fonts using their advanced possibilities and has to use it running X<sub>E</sub>L<sub>A</sub>T<sub>E</sub>X or Lua<sub>T</sub>E<sub>X</sub>, but that's another challenge...

Obviously, *otf* fonts will solve the above features problems without an exponential number of files, but won't easily solve these:

- small or large small caps;
- light or default fonts.

## 5 The end

Now I think the work is (almost) done and I'm proud of three things:

- the package runs mainly correctly,
- some people like the fonts and some people don't like them,
- some people like to customize their text and/or math typesetting using the set of options.

If everybody finds these three axioms are reasonable, you know what, I'm happy...

If I, if I have been unkind,  
I hope that you can just let it go by.

LEONARD COHEN, *Bird on the wire*

◇ Christophe Caignaert  
<http://ctan.org/pkg/kpfonts>

Solution to the quiz (p.168):

1. *ligh*ttxt and *veryoldstyle*  
2. *oldstyle*lennums and *largesmallcaps*  
3. *easyscs*l and *oldstyle*  
4. *ligh*ttxt and *no*lfigatures  
5. *ligh*ttxt and *oldstyle*  
6. no options  
7. *no*lfigatures  
8. *ligh*ttxt  
9. *largesmallcaps*  
10. *ligh*ttxt, *largesmallcaps*, *easyscs*l and *no*lfigatures

---

## Giving it away

Jim Hefferon

In the early 90s I wrote an undergraduate textbook. Inspired by the tools used to write it, including L<sup>A</sup>T<sub>E</sub>X, I made the book available under a free license, the GNU Free Documentation License.<sup>1</sup>

Authors today do this more often but back then giving away a book was unusual. Since this material has been around for longer than most, perhaps a discussion of my experience would be helpful to someone considering such a project.

I will discuss advantages and disadvantages of using T<sub>E</sub>X for this, and a few other points. I won't list the T<sub>E</sub>X code but you can get it from the book's web page: <http://joshua.smcvt.edu/linearalgebra>.

### 1 Background

The book *Linear Algebra* is for a US undergraduate course often taken during a student's second year. The pedagogical goal is to help these young students make a transition from the formula-driven early classes to proof-driven later courses. It is popular, with 100,000 downloads last year, and it is often listed first in a "linear algebra" web search.

In addition to the PDF of the text, downloaders can get the full L<sup>A</sup>T<sub>E</sub>X source. They can also get a PDF of the fully-worked answers to all exercises, even the proofs.

All this was helped by using L<sup>A</sup>T<sub>E</sub>X. For one thing, I wasn't paying a typesetter so I didn't have to recoup that cost, and revisions cost me no money. I also benefited from the advanced and free tools, such as GNU/Linux and Emacs with AUCT<sub>E</sub>X, that fit a L<sup>A</sup>T<sub>E</sub>X workflow.

### 2 L<sup>A</sup>T<sub>E</sub>X helps

I put *Linear Algebra* up for download more than a decade ago. What I offer now is essentially unchanged from what I offered then. That is, because I use L<sup>A</sup>T<sub>E</sub>X, I have had no bit rot: I've never had emails that say, "I have version 5 of the program and you've used version 6 and I'm having trouble." This is great because an author providing material at no profit has nothing to gain from version maintenance.

With time, I have enjoyed a number of other advantages of T<sub>E</sub>X-based production. The main one is that because T<sub>E</sub>X produces first-class output, an instructor can without apology use the material in class. Another advantage is that the source is compact, limiting the amount by which downloads impact my college's bandwidth.

<sup>1</sup> <http://www.gnu.org/licenses/fdl.html>

### 3 Doing the exercises

When I started, I knew very little L<sup>A</sup>T<sub>E</sub>X. Back then there were fewer packages and I had to program many of my needs myself. I'll try to give a potential author a sense of the process by discussing what happened in just one area, producing the exercises.

First, I wanted to number the formal parts in a single sequence, including the exercises. Thus, if a section ends with a lemma and a theorem numbered 1.15 and 1.16 then the problems should start with 1.17. For this, I had to read some L<sup>A</sup>T<sub>E</sub>X source and even small adjustments of existing macros can take some head-scratching. Looking back, I'd guess this beginner's step took a day to work out.

Next I wanted to mark some exercises for people reading the text on their own. I needed that

```
\begin{exercises}
\recommended\item Calculate the ..
```

would put a check in the margin next to the exercise. This used L<sup>A</sup>T<sub>E</sub>X lists and I had to ask online about a point — perhaps it cost me two days.

My third problem was harder. I wanted the source file to include answers to the exercises.

```
\begin{exercises}
\item Prove that ..
\answer{Observe first that ..}
```

For this, T<sub>E</sub>X writes the answer text to a separate file, including in that file the exercise number. I could not have done the programming but fortunately Mike Piff provided the `answers` package to do exactly this.

(Many texts have either limited answer sets or else the answers are not written by the author. If you are a potential author, I urge you to consider doing full answers. While providing these answers, and L<sup>A</sup>T<sub>E</sub>X-ing them myself, was a great deal of trouble, it made the book much better for learners. For example, answering an exercise might bring out a subtlety and so I'd go back to adjust one of the examples.)

However, for my exercises I had to do more than Mike's package provided. I wanted a hyperlink from each question to its answer and one from the answer to the question. For this I had to code inside the `hyperref` package, which is hard. Perhaps this cost me three days.

Another problem with the exercises arose after I put the book up for download. Instructors wanted to assign hand-in problems but didn't want to invent their own. The fact that students could download all the answers prevented these instructors from using the book. In response I tried posting only some answers, which required that I develop an option to produce only the answers to recommended exercises.

$\TeX$ 's `if` constructs gave me trouble, so this cost me a couple of days. (At that time my policy was that to get all the answers a person had to email me with a good story. After a while the absurdity of this became compelling and besides, finding the entire set of answers by searching online became easy, so I am now back to offering all the answers.)

I never solved my final problem. My workflow was to compile the document, generating the answers as a separate file, and then to compile those answers. If  $\LaTeX$  found errors in the answers then it reported line numbers from the generated file. But I needed the line number from the original source file. I hacked at this a bit, but eventually felt that I should be writing the book instead of writing the tool used for the book, and so I never got it to go.

#### 4 Positives, negatives

Providing the book free for download has had some positive effects. I am delighted to get emails from people, particularly people with few resources, who say that they have been helped by the text and by its availability. Another positive is the bug reports that some readers send. That is, providing it freely has garnered both exposure and good will.

There have also been some aspects of this distributing method that were more mixed.

I know of five projects to use the source as the basis for a translation. But while one project is still in progress and looks promising, the other attempts have petered out.

I also know of three projects to use the source to make a wiki. The one I know the best was very well done and includes all of the text and illustrations. However, these projects never achieved true wiki-osity in that they never became dynamic documents with many contributors.

The experiences of the wiki folks matches my own. I imagined that providing the  $\LaTeX$  source would allow instructors to adjust the text by adding or deleting sections or exercises. In particular, each chapter has sections of topics, which are optional, light, extensions of the material. On the download page I solicited contributions of more topics and exercises and seeded my collection by imposing on a few colleagues. To get contributors started, I provided a booklet on compiling the text's source. However my imagination was wrong; no contributions have appeared.

Finally, I will mention a potential negative aspect of free distribution: people have downloaded the

book and put it up for sale at online print-on-demand publishers. Some of these are instructors or schools who want their students to buy the paper book for a course, which is perfectly natural and fine (in fact, after many calls I've put on the download page a note to college bookstores assuring them that it is allowed). In another case the people involved sell the text at cost, to make a paper version easily available. But I also know of people who simply grabbed some freely available books to sell for a profit, which is annoying. Perhaps I will someday put up my own on-demand version but so far I have stuck to online distribution.

#### 5 Possibilities

When I started, there were no stand-alone book display devices so I did not provide the material in a format that suits these. Were I starting this project today, I would study the possibilities of these alternative platforms.

Even more interesting are the possibilities for interactive goodness. Today PDF is an open standard and allows JavaScript, so there is a stable way to get cross-platform interactivity.

The most exciting possibility would be to have a group of people contributing applications. Anyone who watches an active Internet community has to be impressed with the tremendous creativity and energy that can happen when great people get going. Again, the fact that  $\LaTeX$  is a standard with first-rate output makes this at least conceivable.

#### 6 Closing

Free distribution, particularly based on  $\TeX$ , has some real advantages but some trade-offs as well.

Chief among the advantages of  $\LaTeX$  for this project were its high-quality output, its stability, and the widespread availability of associated tools. Because of these advantages, I could offer a text free for download without a long-term commitment to maintenance.

The main disadvantage to producing the work in  $\LaTeX$  was the coding. This may be mitigated by the fact that there are more  $\LaTeX$  packages today, so the need for individual coding may be reduced.

If you take on such a project, enjoy!

◇ Jim Hefferon  
 Saint Michael's College  
 Colchester, VT USA  
 ftpmaint (at) tug dot ctan dot org



## Glisterings

Peter Wilson

If our understanding have a film of  
ignorance over it, or be blear with gazing  
on other false glisterings, what is that to  
truth?

*Of reformation in England*, JOHN MILTON

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

Corrections, suggestions, and contributions will always be welcome.

When true simplicity is gained,  
To bow and bend, we will not be ashamed.  
To turn, turn, will be our delight  
'Til by turning, turning, we come 'round  
right.

*Simple Gifts*, a Shaker hymn

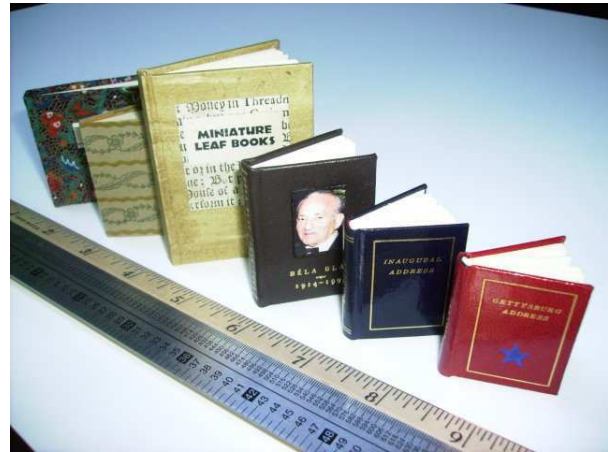
## Meandering miniature books

Some years ago William Adams produced an eight-page booklet called *One Typeface, Many Fonts* [1], which I encourage you to get if you do not already have it. Apart from the content and the various typefaces an interesting aspect was that it was printed on one side of a single sheet of letterpaper, which could then be cut and folded to make the final booklet. I found this the other month when I was clearing out old papers getting ready to move house.

A little earlier I had come across a class of books called *miniatures* [2], which are defined as books not more than 3 in, or 76 mm, in height.<sup>1</sup> Some are shown in Figure 1. The largest is 3 by 2 1/8 inches and is a miniature book about miniature books. The two smallest in the group are 1 5/8 by 1 1/4 inches. One is John Kennedy's Inaugural Address in January 1961 and the other is Abraham Lincoln's speech at Gettysburg in November 1863. The type in these appears to be just a little smaller than that in the footnotes here.

These two events got me to wondering whether there were other methods like William's of creating a (miniature) booklet. I tried cutting and folding scrap paper in many ways with not much success until I remembered that I had a book by Cherryl Moote [6] which had been advertised with:

<sup>1</sup> In 2000 the record for the smallest miniature was held by *The Twelve Horary Signs — Chinese Zodiac* published in an edition of 100 by the Topan Printing Company, Tokyo, Japan. It measured just 0.95 mm square!



Herries Collection

**Figure 1:** Some miniature books; the scales are marked in inches, points, and picas

|          |   |   |   |     |
|----------|---|---|---|-----|
| <u>1</u> | 2 | 3 | 4 | etc |
|----------|---|---|---|-----|

**Figure 2:** Accordion layout

This book features book forms perfect for creating small editions of art books with your photocopier or computer printer. . . Standard size papers are used for most projects . . .

There are several ways in which you can print on one side of a single sheet and by folding and cutting make acceptable small works. The simplest is an accordion book where the pages on the sheet are laid out as in Figure 2. In this and later diagrams the numbers correspond to the page ordering and text orientation, with potentially ambiguously oriented numbers underlined, such as 1, 6, and 8. Thin lines are where the paper is to be folded, and thick lines where it is to be cut. In the accordion form the folds are alternately up and down (or down and up). Many Japanese and Chinese books use this basic form. One example is illustrated in Figure 3 which shows one quarter of a book whose English title is *Biographies of Twelve Chinese Great Scholars*; the text is in both Chinese and a form of English.

Another simple form is one called *French Folds* where the paper is folded in half one way then folded in half again the other way. This is often used for greetings cards, and is illustrated in Figure 4.

William Adams refers to his booklet form as a *Stroke Book* and it is called a *Two-Minute Book* by Cherryl Moote. The general layout for such a



Herries Collection

**Figure 3:** A Chinese book in the accordion style, showing one quarter of the overall work which measures 8 1/4 inches by 14 feet

|   |          |
|---|----------|
| ε | ζ        |
| 4 | <u>1</u> |

**Figure 4:** French Folds layout

work is shown in Figure 5. William’s instructions for ‘binding’ the booklet are:

After printing fold in half (top to bottom), unfold, fold in half lengthwise, then fold in half and open, and fold each resulting panel in half, unfold and restore to the original fold, then cut along the inner half of the lengthwise fold, open and fold lengthwise. Then all the pages should be folded within the front and back covers and *voilà!* a single signature booklet.

Another way of describing the procedure is:

Fold in half, short side to short side (1/2 to 6/5) with text exposed; this is called a *mountain fold*. Fold each half in half again (6/5 to 7/4 and 1/2 to 8/3) with text hidden; these are both *valley folds*. Unfold to original flat sheet. Fold in half lengthwise (1/8/7/6 to 2/3/4/5) with text exposed (i.e., a mountain fold). Cut along the inner half of the lengthwise fold (the thick line in the diagram). Refold lengthwise, push the two pairs of end pages (1/2 and 6/5) towards each other and the center pages should fold outwards. Finally, fold the result so that the pages are in the prescribed order.

I didn’t know how William imposed his eight pages onto the one sheet but I suspected that he typeset each page on separate sheets then used some imposition software like `psnup` to arrange these on a single sheet. I have since learnt from him<sup>2</sup> that:

<sup>2</sup> Personal email, 2008/06/25.

|    |    |   |    |
|----|----|---|----|
| 1̄ | 8̄ | 2 | 9̄ |
| 2  | 3  | 4 | 5  |

**Figure 5:** Layout for a Two-Minute book

I composed *One Typeface Many Fonts* in Altsys Virtuoso (a drawing program) on my NeXT Cube—I saved out .eps files (a nifty facility of a pdf viewing program I was using in NeXTstep) and then arranged and rotated the bits by hand working up from a folded “dummy” (you’re a dummy if you don’t make a dummy).

I know of some pure  $\text{\LaTeX}$  methods for imposition. There is the `booklet` package [9] for creating a booklet of half-size pages from full size originals, and Nicola Talbot’s `flowfram` package [8] which lets you define ‘frames’ on a page and the text will automatically flow from one frame to the next; Andreas Matthias’ `pdfpages` package [5] provides imposition facilities if you are using  $\text{pdf\LaTeX}$ . As separate processes Angus Duggans’ `PSUtools` suite [3] has several programs, such as `psnup`, for imposition of `ps` files, while Tom Phelps’ `Multivalent` suite of tools [7] provides similar programs for dealing with `pdf` files. I was, though, particularly interested using  $\text{\LaTeX}$  to create miniature books from a single sheet of paper, printed on one side only, of short poems or epigrams without using any packages. For this kind of work it seemed reasonable to do ‘page breaking’ by hand rather than automatically, as the `flowfram` package would provide.<sup>3</sup>

Cheryl presented several layouts that, applied to the most common size<sup>4</sup> of paper, result in a miniature book. These are illustrated in Figures 6–9. To create a book from one of these, cut along the thick lines and then start folding starting with pages 1 and 2, where 1 will be the first one in the book, and proceeding along until page 16 is reached. The folds should be alternately mountain and valley. The diagrams show the initial orientation of the text ‘pages’ to give a reasonable orientation after the folding. The pages may be hinged at the left or right or top or bottom. The orientation sequence does depend

<sup>3</sup> Also, I couldn’t work out how to automatically put a ‘page number’ in the frames.

<sup>4</sup> Either letter or A4 size.

|           |          |          |          |
|-----------|----------|----------|----------|
| <u>1</u>  | 2        | 3        | 4        |
| 12        | 13       | 14       | 5        |
| <u>11</u> | 9I       | 15       | <u>6</u> |
| 0I        | <u>6</u> | <u>8</u> | 7        |

**Figure 6:** Spiral layout (mountain)

|          |    |           |    |
|----------|----|-----------|----|
| <u>1</u> | 2  | 3         | 4  |
| <u>8</u> | 2  | <u>9</u>  | 9  |
| <u>9</u> | 10 | <u>11</u> | 12 |
| 9I       | 9I | 7I        | 13 |

**Figure 8:** Snake layout (mountain)

|           |          |          |          |
|-----------|----------|----------|----------|
| <u>1</u>  | 2        | 3        | 4        |
| 7I        | 13       | 14       | 5        |
| <u>11</u> | 9I       | 9I       | <u>6</u> |
| 0I        | <u>6</u> | <u>8</u> | 7        |

**Figure 7:** Spiral layout (valley)

|    |    |           |          |
|----|----|-----------|----------|
| 4  | 5  | <u>6</u>  | 7        |
| 8  | 7  | <u>1</u>  | <u>8</u> |
| 14 | 15 | 16        | <u>9</u> |
| 8I | 7I | <u>11</u> | 10       |

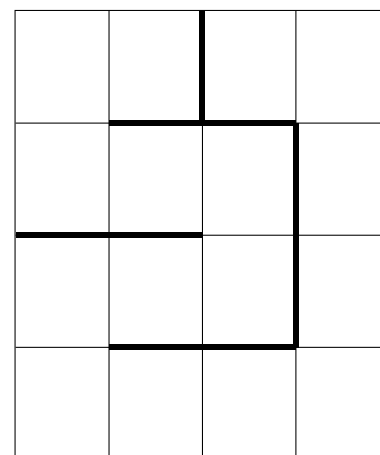
**Figure 9:** T layout (mountain)

on whether the first fold is a mountain or a valley. When unfolding most of these layouts to read the contents it may be necessary to twist and turn the book in unexpected ways.

I don't know if there are any commonly accepted names for these layouts so I have used my own.

You can design your own layout if you prefer. For instance Figure 10 is one that I made up; I make no claim regarding either its usefulness or its aesthetics, nor how the folds or page numbering should be configured. The shape of the cuts vaguely reminds me of stoking a wood burning stove, hence the name.

Perhaps you have been wondering how I produced these diagrams? But even if you haven't, I did it by using the `graphicx` package and the `picture` environment. For example, here is the essence of the code for Figure 6; it does get tedious after a while.



**Figure 10:** Stove layout

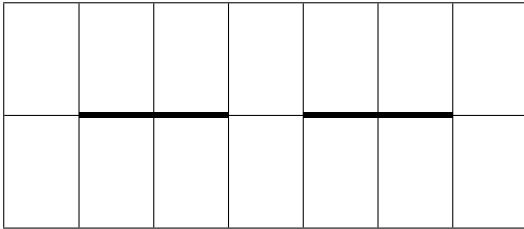


Figure 11: Interleaved or Dos-a-Dos layout

```
%% turn its argument upside down
\newcommand*\rupd[1]{%
  \rotatebox[origin=c]{180}{#1}}
%% save some typing
\let\ul\underline
%% the diagram
\begin{figure}
\centering
\setlength{\unitlength}{0.003\textwidth}
\begin{picture}(100,120)
  %% draw the boxes
  \thinlines
  \put(0,0){\framebox(100,120){}}
  \put(25,0){\line(0,1){120}}
  \put(50,0){\line(0,1){120}}
  ...
  \put(0,90){\line(1,0){100}}
  %% draw the cutting lines
  \linethickness{2pt}
  \put(0,90){\line(1,0){75}}
  \put(75,90){\line(0,-1){60}}
  \put(75,30){\line(-1,0){50}}
  \put(25,30){\line(0,1){30}}
  \put(25,60){\line(1,0){25}}
  %% insert the page numbers
  \linethickness{0pt}
  \put(0,90){\framebox(25,30){\ul{1}}}
  \put(25,90){\framebox(25,30){2}}
  ...
  \put(0,0){\framebox(25,30){\rupd{10}}}
  \put(25,0){\framebox(25,30){\rupd{\ul{9}}}}
  \put(50,0){\framebox(25,30){\rupd{\ul{8}}}}
  \put(75,0){\framebox(25,30){7}}
\end{picture}
\caption{Spiral layout (mountain)}
\label{fig:lay1M}
\end{figure}
```

Cherryl Moote described other layouts that led to more complex results after cutting and folding, one of which is illustrated in Figure 11. Essentially this consists of two Two-Minute layouts (see Figure 5) joined together. By folding this in one way you can produce a Dos-a-Dos book which is two Two-Minute books conjoined back to back, and in folding another way you can interleave pages from the left and right halves.

Peter Wilson

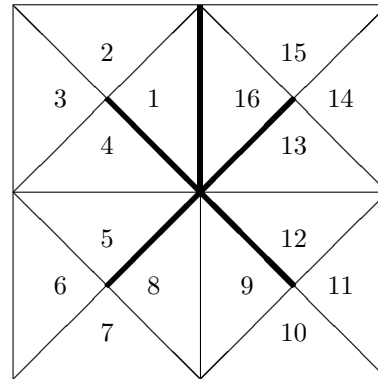


Figure 12: A triangular meander

A rather different one is illustrated in Figure 12, being a kind of twisty triangular accordion book. The numbers show the sequence of the triangular pages but not the orientation of any text that might be on them.

With all these layouts you have to experiment to see what is best for the particular project you have in mind.

As an aid to seeing how miniature books can be based on one or other of the presented layouts I offer Figure 13 and Figure 14. You can photocopy these and cut and fold the copies to see what the result(s) look like. Both of the offerings are based on the Serpent layout, with the first following Figure 8 which starts off with a mountain fold. In this case the title is on the very first page and the colophon is on the last.

The second is meant to start off with a valley fold, and the first and last pages after folding are both blank. This is so you can use these pages as endpapers and attach cover boards to them to give a more finished look to the booklet.

If you would like to try something similar, here is the code for the layout in Figure 14.

```
%% The miniature page sizes
\newlength{\across}
\newlength{\down}
\setlength{\across}{0.2\textwidth}
\setlength{\down}{0.2\textheight}
%% no space between an \fbx and contents
\setlength{\fbxsep}{0pt}
\let\fbx\fbx
%% vplace environment is in memoir class
%% vertical placement of contents,
%% centered by default
\providecommand{\vplace}[1][1]{%
  \par\vspace{\stretch{#1}}}
\def\endvplace{\vspace*{\stretch{1}}\par}
%% Put one minipage centered inside another
\newcommand{\portion}[1]{\fbx{%
```

|   |  |  |  |
|---|--|--|--|
| <p><i>Vitae<br/>Summa<br/>Brevis</i></p> <p>Ernest<br/>Dowson</p> |  | <p>They are not<br/>long,</p> <p>3</p>           | <p>The weeping and<br/>the laughter,</p> <p>4</p>      |
| <p>We pass the gate.</p> <p>8</p>                                 | <p>2</p> <p>in us after</p>                      | <p>9</p> <p>I think they have<br/>no portion</p> | <p>Love and desire<br/>and hate:</p> <p>5</p>          |
| <p>They are not<br/>long,</p> <p>6</p>                            | <p>the days of wine<br/>and roses:</p> <p>10</p> | <p>Out of a misty<br/>dream</p> <p>11</p>        | <p>Our path<br/>emerges for a<br/>while,</p> <p>12</p> |
| <p>2008</p> <p>The Herries Press</p>                              |  | <p>14</p> <p>Within a dream.</p>                 | <p>then closes</p> <p>13</p>                           |

Figure 13: Layout of a miniature book based on that shown in Figure 8, starting with a mountain fold

|  |   |  |  |
|--|---|--|--|
|  | <p><i>Vitae<br/>Summa<br/>Brevis</i></p> <p>Ernest<br/>Dowson</p> | <p>They are not<br/>long,</p> <p>3</p>           | <p>The weeping and<br/>the laughter,</p> <p>4</p>      |
| <p>8</p> <p>We pass the gate.</p>      | <p>7</p> <p>in us after</p>                                       | <p>6</p> <p>I think they have<br/>no portion</p> | <p>5</p> <p>Love and desire<br/>and hate:</p>          |
| <p>They are not<br/>long,</p> <p>9</p> | <p>the days of wine<br/>and roses:</p> <p>10</p>                  | <p>Out of a misty<br/>dream</p> <p>11</p>        | <p>Our path<br/>emerges for a<br/>while,</p> <p>12</p> |
|  | <p>2008</p> <p>The Herries Press</p>                              | <p>14</p> <p>Within a dream.</p>                 | <p>13</p> <p>then closes</p>                           |

Figure 14: Layout of a miniature book based on that shown in Figure 8, starting with a valley fold

```

\begin{minipage}[c][\down][t]{\across}
#1
\end{minipage}}
\renewcommand{\portion}[1]{\fbx{%
  \begin{minipage}[c][\down][t]{0.8\across}
  #1
  \end{minipage}
\end{minipage}}}
%%%% Vertically center contents of a portion
\newcommand{\vcp}[3][1]{%
\portion{\centering%
  \begin{vplace}[#1]#2\end{vplace}#3%
  \vspace*{\onelineskip}}
%%%% turn contents of a \vcp upside down
\newcommand{\rvcp}[3][1]{%
  \rotatebox[origin=c]{180}{%
    \vcp[#1]{#2}{#3}}
%%%% for typesetting page numbers
\newcommand*{\pgn}[1]{\tiny #1}
%%%% Typeset
\noindent
%%%% top row
\vcp{\mbox{}}{}
\vcp{%
  \Large \textit{Vitae Summa Brevis} \\\[5mm]
  \large Ernest Dowson}{}
\vcp{They are not long,}{\pgn{3}}
\vcp{The weeping and the laughter,}{\pgn{4}}
%%%% second row
\noindent
\rvcp{We pass the gate.}{\pgn{8}}
\rvcp{in us after}{\pgn{7}}
\rvcp{I think they have no portion}{\pgn{6}}
\rvcp{Love and desire and hate:}{\pgn{5}}
%%%% third row
\noindent
\vcp{They are not long,}{\pgn{9}}
\vcp{the days of wine and roses:}{\pgn{10}}
\vcp{Out of a misty dream}{\pgn{11}}
\vcp{Our path emerges for a while,}{\pgn{12}}
%%%% bottom row
\noindent
\vcp{\mbox{}}{}
\rvcp{\footnotesize The Herries Press\\[1cm]
  2008}}{}
\rvcp{Within a dream.}{\pgn{14}}
\rvcp{then closes}{\pgn{13}}

```

I have not used the whole of the printed sheet in producing the miniatures, but rather the extent of the typeblock (see the definitions of `\across` and `\down` which I used in the specification of the size of the final pages). If you are using the memoir class you can easily change the size of the typeblock,

otherwise you can use the geometry package. I also boxed, using `\fbox` (via `\fbx`), each final page. If you do not want to do that then change `\fbx`, for example:

```
\renewcommand*{\fbx}[1]{#1}
```

May you have much pleasure in creating your own unique miniature books.

### Acknowledgements

William Adams was kind enough to review the column and I have incorporated many of his suggestions. One that I didn't, but will now, is to say that he felt that another possible source for folding techniques would be the literature on origami. Though perhaps not directly related, he said that *The Folding Universe* [4] is a fascinating book regardless and well worth looking at.

### References

- [1] William Adams. *One Typeface, Many Fonts*. William Adams, [http://mysite.verizon.net/william\\_franklin\\_adams/portfolio/typography/onetype-sheet.pdf](http://mysite.verizon.net/william_franklin_adams/portfolio/typography/onetype-sheet.pdf), 1997.
- [2] Anne C. Bromer and Julian I. Edison. *Miniature Books: 4,000 Years of Tiny Treasures*. Abrams in association with The Grolier Club, 2007. ISBN 978-0-8109-9299-3.
- [3] Angus Duggan. PSUtils, 2008. (<http://www.tardis.ed.ac.uk/~ajcd/psutils>).
- [4] Peter Engel. *The Folding Universe*. Vintage, 1989. ISBN 0394757513.
- [5] Andreas Matthias. The pdfpages package, 2010. <http://mirror.ctan.org/macros/latex/contrib/pdfpages>.
- [6] Cheryl Moote. *Copied, Bound & Numbered*. At Your Ease Publications, 2003. ISBN 0-9688811-7-3.
- [7] Tom Phelps. Multivalent, 2008. <http://multivalent.sourceforge.net>.
- [8] Nicola L. C. Talbot. Creating flow frames for posters, brochures or magazines using flowfram.sty, 2010. <http://mirror.ctan.org/macros/latex/contrib/flowfram>.
- [9] Peter Wilson. Printing booklets with L<sup>A</sup>T<sub>E</sub>X, 2009. <http://mirror.ctan.org/macros/latex/contrib/booklet>.

◇ Peter Wilson  
herries dot press (at)  
earthlink dot net

## Three things you can do with LuaTeX that would be extremely painful otherwise

Paul Isambert

### Introduction

LuaTeX has made some typographic operations so easy one might wonder why it wasn't invented thirty years ago (probably because Lua didn't exist then).

Here I'm going to describe three simple features that would require advanced wizardry to do the same with any other engine. LuaTeX allows you to explore some of TeX's most intimate parts with a rather easy programming language, and the result is you can quite readily access things that were unreachable before. The three issues I'm going to address are:

- Turning lines into rules whose color depends on the line's original stretch or shrink.
- Underlining.
- Margin notes that align properly with the text.

I'll try to explain some of LuaTeX's basic functionality as we encounter these issues, but two of them are worth mentioning right now: callbacks and nodes.

First, we can control TeX's operations at various stages thanks to *callbacks*. These are points at which we can insert Lua code to modify or enhance TeX's processing. Callbacks range from processing TeX's input buffer (e.g. to accommodate a special encoding) to rewriting the paragraph builder and loading OpenType fonts.

Second, we can manipulate lists of *nodes*. To put it simply, nodes are the atoms that TeX uses to create pages: boxes, glyphs, glues, but also penalties, whatsits, etc. A list of nodes is a sequence of such atoms linked together. A simple paragraph, for instance, is a list made of horizontal boxes (the lines), penalties and glues. The boxes themselves are lists containing mostly glyph and glue nodes. Nodes are linked together like beads on a string, and the `prev` field of a node points to the preceding node in the list, whereas the `next` field returns the one that follows (there is an understandable exception for the first and last nodes of a list, whose `prev` and `last` fields respectively return `nil`). An important point to keep in mind is that when you query the content of, say, an `\hbox`, which in TeX's internal is

a horizontal list, what you get is the first node of that list; you access the rest by sliding from `next` to `next`.

Nodes also have several other *fields*, depending on their types. These types are recorded as a number in their `id` field, a numeric value. For instance, a glue node has `id` 10, whereas a glyph node has `id` 37. As long as LuaTeX hasn't reached version 1, though, such values might change. So, in order for our code to last, we must use the following workaround: the `node.id()` function, when fed a string denoting a node type, returns the associated `id` number. For instance, `node.id("glue")` returns 10. Thus, when using symbolic names, we can get the right `id` value, regardless of changes in versions of LuaTeX. Another important field for nodes is `subtype`, which distinguishes between nodes with the same `id`. It's a numeric value, and for whatsits (which are numerous), one should use `node.subtype()` like `node.id()`.

Symbolic names won't change; they are listed in the LuaTeX reference manual, in the chapter called *Nodes*, available from the LuaTeX web site; they're also listed in the tables returned by `node.types()` and `node.whatsits()`. It's simpler to define variables beforehand rather than call `node.id` and `node.subtype` each time we need them. That's what we'll do here: the following declarations should start any file containing our code; it can also be made global by removing the `local` prefix and thus used anywhere once declared, but local variables are faster and safer. I use uppercase to mark their status.

```
local HLIST = node.id("hlist")
local RULE  = node.id("rule")
local GLUE  = node.id("glue")
local KERN  = node.id("kern")
local WHAT  = node.id("whatsit")
local COL   = node.subtype("pdf_colorstack")
```

### The color of a page

Typographers speak of a page's color. While the color itself depends on several factors, its *evenness* depends on how lines are justified: loose lines make the page uneven in color, because large interword space creates holes in the overall greyness.

The code that follows takes the metaphor literally: it turns a page's color into a real color pattern. The idea is to replace each line with a rule of the same height and width, and whose color depends on the line's badness. If we take 0 as black and 1 as white, then a good line gets .5, tight lines approach 0 (which represents an overfull line) and

---

*Author's note:* I'm not a member of the LuaTeX team and this paper has no kind of official authority—it's just the result of experimentation by a LuaTeX user. Any error or misconception is mine.



loose lines tend to 1 (an underfull line). Now we have paragraphs and pages made of grey bars; the less contrast between them, the better the page.

To do this, we retrieve the horizontal boxes created by the paragraph builder, check the badness of each, then replace the box with the desired rule. This is easy to do in LuaTeX: we register a function in the `post_linebreak_filter` callback. This callback accesses the list of nodes output by the paragraph builder, i.e. the lines of text interspersed with interline penalties and glues, plus perhaps other things (whatsits, inserts, ...) that we'll ignore. Among these nodes we retrieve the ones we want, namely the lines of text, and replace them as described.

The code that follows, as all Lua code, should be fed to `\directlua` or stored in a `.lua` file.

```
local color_push = node.new(WHAT, COL)
local color_pop  = node.new(WHAT, COL)
color_push.stack = 0
color_pop.stack  = 0
color_push.cmd   = 1
color_pop.cmd    = 2
```

Here we have created two new whatsit nodes identified by their `subtype` as the Lua equivalents of `\pdfcolorstack`. They both modify stack 0 and `color_push` adds code to the stack while `color_pop` removes it. We'll use them to set the color of each line, with the exact content of the code added by `color_push` to be specified each time.

```
textcolor = function (head)
  for line in node.traverse_id(HLIST, head) do
    local glue_ratio = 0
    if line.glue_order == 0 then
      if line.glue_sign == 1 then
        glue_ratio = .5 * math.min(line.glue_set,
                                   1)
      else
        glue_ratio = -.5 * line.glue_set
      end
    end
    color_push.data = .5 + glue_ratio .. " g"
```

Here's the beginning of our main function. It takes a node as its argument: it will be the first node of the list returned by the paragraph builder. That node, remember, denotes the entire list. We retrieve each line of text in this list, i.e. each node with `id` `HLIST`, and check its `glue_order` field; if it is 0, then the line has been justified with finite glue and we want to know how bad it is (if the line uses infinite glue then it is good by definition, as far as glue setting is concerned). We access `glue_sign` to know whether stretching or shrinking was used and `glue_set` to know the ratio (1 means the stretch/shrink was fully used; glues can also be overstretched, but we

don't allow more than 1 in order to remain in the color range).

The last line sets the color of the line as the code to `color_push`, i.e. '`n g`', where `n` is a number between 0 and 1 and `g` a PDF operator setting the color in the grey model. In the rest of the loop we replace the line's content with a sequence of three nodes: `color_push`, a rule, and `color_pop`:

```
local rule = node.new(RULE)
rule.width = line.width
local p = line.list
line.list = node.copy(color_push)
node.flush_list(p)
node.insert_after(line.list,
                  line.list, rule)
node.insert_after(line.list,
                  node.tail(line.list),
                  node.copy(color_pop))
end
```

What is done here is: first, we create a rule whose width is the same as the original line's (we could have created this rule beforehand with a width equal to `\hspace`, but this way we accommodate changing line widths). Then we set the line's list as a copy of `color_push` (we use a copy since we need that node for each line), and then we insert the rule node and a copy of `color_pop`. The first argument to `node.insert_after` is the list (denoted by its first node!) where we perform the insertion, the second one is the node in that list after which the insertion is performed, and the last one is the inserted node; `node.tail` returns the last node of its argument, so the third `node.insert_after` inserts at the end of the list.

The story with `p` is this: we retrieve the line's content before replacing it, so we can erase it from TeX's memory; it has no effect on the output.

Finally, and most importantly, we return the mutated list for TeX to continue its operations, and close the function.

```
return head
end
```

Now, to use the function, we register it in the `post_linebreak_filter` callback:

```
\directlua{%
  callback.register("post_linebreak_filter",
                    textcolor)}
```

Note that we could improve this code for the first and last lines of a paragraph, taking the indent and `\parfillskip` into account to create more faithful images of those lines. I leave it as an exercise to the reader, as is customary.

## Underlining

The previous code was (hopefully) fun but not terribly useful (well, who knows?); let's do something (hopefully) more useful and no less fun.

Everybody knows that underlining is in bad typographic taste. That said, it may have its uses, and anyway allows us to investigate Lua $\TeX$  further. Underlining has been done in  $\TeX$  (see Donald Arseneau's `ulem`, for instance); it requires great wizardry and has some limitations. With Lua $\TeX$ , it's (almost) child's play.

The problem with underlining in  $\TeX$  is that you have to add the underline before the paragraph is built, and this hinders hyphenation. In Lua $\TeX$  we can do it after hyphenation is done: we retrieve the nodes to underline in the typeset lines. But how do we spot them? The answer lies with another basic Lua $\TeX$  functionality, namely attributes. These are very simple yet very powerful. An attribute is like a count register in that it holds a number. The difference with a count register is that nodes retain the values of all attributes in force *when they were created*. Thus, we can set an attribute to some value, input some text, and then reset the attribute; the text will have the value attached to it for the rest of  $\TeX$ 's processing.

This leads to the first definition:

```
\def\underline#1{%
  \quitvmode \attribute100 = 1 #1%
  \attribute100 = -"7FFFFFFF
  \directlua{callback.register(
    "post_linebreak_filter", get_lines)}%
}
```

It's important to use `\quitvmode` so that the indentation box is inserted before the attribute is set and not be underlined (in case the underlined text is the beginning of a paragraph).

An attribute is 'set' if it has any value but `-"7FFFFFFF`. So setting it to 1 here would be the same thing as setting it to `-45` (see the end of this section for an example of use for different values). Now all nodes produced by the argument to `underline` have the value 1 for attribute 100—which was arbitrarily chosen. Attribute 458 would have been equally good. Actually, one should use attributes with greater care, i.e. they should be allocated with macros like `\newcount`, so that one never uses the same attribute for different tasks.

The last action performed by `\underline` is to register a function in the `post_linebreak_filter` callback. It does so because the Lua function used to underline clears the callback (as we'll see), so that it is called only on those paragraphs where it

is required. It could be called on all paragraphs, but it'd waste  $\TeX$ 's time.

Let's now turn to the Lua functions:

```
get_lines = function (head)
  for line in node.traverse_id(HLIST, head) do
    underline(line.list, line.glue_order,
      line.glue_set, line.glue_sign)
  end
  callback.register
    ("post_linebreak_filter", nil)
  return head
end
```

This first function retrieves all lines in the paragraph and feeds their content to the `underline` function along with information about glue setting. It then clears the callback and returns the head. This part is nothing we haven't seen in the previous code.

Some nodes might have inherited the attribute's value, although we don't want to underline them: `\leftskip`, `\rightskip`, and `\parfillskip`. These are glue nodes and their `subtypes` are 8, 9 and 15, respectively. The following function is meant to filter them out. (Note: versions prior to v0.62 had a bug where `\leftskip` and `\rightskip` were not properly identified, so `item.subtype == 7` should be added to the `or` conditional below. Both  $\TeX$  Live 2010 and Mik $\TeX$  2.9 uses v0.60, so they are affected.)

```
local good_item = function (item)
  if item.id == GLUE and
    (item.subtype == 8 or item.subtype == 9
    or item.subtype == 15) then
    return false
  else
    return true
  end
end
```

Now, here's how the `underline` Lua function starts:

```
underline =
  function (head, order, ratio, sign)
    local item = head
    while item do
      if node.has_attribute(item,100)
        and good_item(item) then
        local item_line = node.new(RULE)
        item_line.depth = tex.sp("1.4pt")
        item_line.height = tex.sp("-1pt")
```

The `while` loop is basically the same thing as traversing the list, but we'll sometimes want to skip nodes, so we'll set the `next` one by hand. We scan nodes, and once we've found one with the right value for the attribute (and which is not one of the glues above), we create our rule (with arbitrary dimensions). `tex.sp` turns a dimension

(expressed as a string) into scaled points, the native measure for Lua code. How wide should the rule be? The length of the material starting at the current node up to the last node with the right attribute. To find this last node, we use the following loop, and then retrieve the length of that material via `node.dimensions`, which returns the material's length when it is typeset with the text line's glue setting. We use `end_node.next` because the function actually measures up to its last argument's `prev` node.

```

local end_node = item
while end_node.next and
good_item(end_node.next) and
node.has_attribute(end_node.next, 100) do
end_node = end_node.next
end
item_line.width = node.dimensions
(ratio, sign, order, item, end_node.next)

```

Finally we insert the line into the list. That's pretty simple: we insert a negative kern (with subtype 1, i.e. a handmade kern, not a font kern) as long as the line after the last underlined node, followed by the line itself. This is equivalent to using `\llap` in plain  $\TeX$ . The end of the code sets the `next` node to be analyzed (including the false part of the overall conditional).

```

local item_kern = node.new(KERN, 1)
item_kern.kern = -item_line.width
node.insert_after(head, end_node,
item_kern)
node.insert_after(head, item_kern,
item_line)
item = end_node.next
else
item = item.next
end
end
end
end

```

We could use different values of the attribute to distinguish different underlining styles. To do so, we would still use `node.has_attribute`, since it returns the value of the attribute, or `nil` if the attribute isn't set. That's another exercise left to the reader.

### Marginal notes

When a document has comfortable margins and notes are infrequent and short, marginal notes are an elegant and convenient alternative to footnotes. They are best typeset with their first line level with the line in the text to which they refer. However, such a rule cannot be absolute. Suppose for instance that a note is called on the last line of a page, and

itself is made of more than one line. If we follow the rule then the note will invade the bottom margin and ruin the design of the page. So it should be shifted up so that its last line is level with the last line of the page. Doing this is also an improvement when the text doesn't fill the page, e.g. at the end of a chapter, even though there might remain space on the page to accommodate the note. The page looks better that way: a note is a note and would be too conspicuous if it were allowed to run without the main text by its side. Ideally, a note should also be shifted up if it runs along a section break, but I'll ignore that case, to keep things simpler. (For an alternative approach in  $\LaTeX$ , see Stephen Hicks' article in *TUGboat* 30:2.)

Generally marginal notes are typeset in a smaller font size and on a smaller leading than the main text. Since the leading is smaller, some lines of the notes won't be level with the textblock's lines; however, there should be some 'cyclical synchronicity' between the two blocks, so that for instance three lines of the main text have the same height as four lines of the note (in  $\TeX$  terms it would mean, for instance, `\baselineskip` at 12pt and 9pt respectively), and the following lines are level again.

Here, however, I will typeset notes with the same leading as the main text to avoid complications. Extra calculations are required to achieve what's been previously described—nothing very complicated, though. I'll simply use italics to distinguish the notes from the main text.

Margin notes so numerous that they sometimes overlap each other and must be shifted upward should probably be converted to footnotes, all the more as they'll require a number or symbol so the reader can spot where in the main text they refer to—whereas sparse notes don't need such a mark, since they're supposed to start on the same line as the text they comment, with the known exception we're investigating here. However, we can use the code below to shift notes whatever the reason, so we'll leave aesthetics aside and shift all notes (the shift might go wrong if there are stretchable vertical glues on the page, e.g. `\parskip`; that can be amended, and it's left as yet another exercise). We won't allow more than one note per line, though, because that definitely doesn't make sense.

Here's the  $\TeX$  part of the code:

```

\newcount\notecount
\suppressoutererror=1
\def\note#1{%
\advance\notecount 1
\expandafter\newbox
\csname marginnote_\the\notecount\endcsname

```

```

\expandafter\setbox
  \csname marginnote_\the\notecount\endcsname=
  \vtop{\hsize=4cm
    \rightskip=0pt plus 1fil
    \noindent\it #1}%
\bgroup
\attribute100=\expandafter\the
  \csname marginnote_\the\notecount\endcsname
\adjust pre {\pdfliteral{}}%
\egroup
}

```

This might be somewhat unfamiliar, even to advanced T<sub>E</sub>Xies, because what we're doing is preparing the ground for Lua code. First, we choose not to insert the note directly in the paragraph (to be shifted later if necessary). Instead, we store the note in a box. For each note, we create a new box; that might seem somewhat resource-consuming, but there are 65,536 available boxes in LuaT<sub>E</sub>X, so a shortage seems only a distant possibility. Alternatively, we could store only the source code for the note (in a macro), and typeset it in a box only when we place notes on the page in the output routine, but the asynchronicity between the processing of the main text and the note might lead to trouble.

So we create boxes instead, with proper settings (mostly, a reduced `\hsize`). To allow `\newbox` to appear inside a macro definition in plain T<sub>E</sub>X, we suppress the `outer` error beforehand; then we set the note in its box with a uniquely defined name (thanks to `\newcount`), and most importantly we set an attribute to the value of the box register and `\adjust` a literal with that attribute. This literal's only role is to mark the line it comes from, so we'll be able to spot lines with margin notes when needed, along with the box's number (the value of the attribute).

The following Lua function, to be inserted in the `post_linebreak_filter` callback, does exactly that: our special `\pdfliterals` give their attributes to the lines they come from, and are removed. Now, the reader might have wondered why we used the `pre` version of `\adjust` instead of the default: it's because of a bug in the actual version of LuaT<sub>E</sub>X (to be fixed in v0.64, I am told): some `prev` fields are sometimes wrong, as would be the case here, and we couldn't link each literal to its line if the latter was before the former. So we use `next` instead. Note that we can't just take for granted that the first `next` node is the line, first because 'pre-`\adjusted`' material is inserted before the `baselineskip` glue, and because there might be more adjusted material between the literal and the line. So we recurse over `next` fields until we find a line (i.e. a node `id` `HLIST`).

```

mark_lines = function (head)
  for mark in node.traverse_id(WHAT, head) do
    local attr = node.has_attribute(mark, 100)
    if attr then
      local item = mark.next
      while item do
        if item.id == HLIST then
          node.set_attribute(item, 100, attr)
          item = nil
        else
          item = item.next
        end
      end
      head = node.remove(head, mark)
    end
  end
  return head
end

```

The following function scans the content of a vertical list, probably box 255, finds the lines that have attribute 100 set to some value, and adds the margin notes to those lines. Remember that our goal is to avoid margin notes running into the space below the textblock (either the bottom margin or the vacant space at the end of a chapter). So we must compute how much space remains to accommodate the note. To do so, we scan the box (the page), starting at the bottom, and accumulate the height and depth of lines and the width of kerns and glues—except kerns and glues that might appear before the last line, i.e. space filling the page. To do so, we have a `first` boolean that is `true` as long as a line hasn't been found and prevents adding the width of glues and kerns. With `node.slide` we grasp the last node of the list, since we're reading it backward.

```

process_marginalia = function (head)
  local remainingheight, first, item =
    0, true, node.slide(head)
  while item do
    if node.has_field(item, "kern") then
      if not first then
        remainingheight = remainingheight
          + item.kern
      end
    elseif node.has_field(item, "spec") then
      if not first then
        remainingheight = remainingheight
          + item.spec.width
      end
    end
  end

```

Now, if we find a line, we add its depth if and only if it's not the first one we encounter (i.e. the last one on the page), because in that case its depth belongs to the bottom margin. Its height is added later, if and only if the line doesn't take a note.

```

elseif node.has_field(item, "height") then
  if first then
    first = false
  else
    remainingheight = remainingheight
      + item.depth
  end
end

```

If attribute 100 is set to some value, then the line takes a note. In that case, we retrieve the box, measure its depth, and compare it to the remaining height. Note that the depth of the box is all its material barring the height of its first line (since we used a `\vtop`), which is exactly what we want: its first line can't go wrong, since it's level with the main text's line from whence it came. We also remove the depth of the last line, since its going into the bottom margin is perfectly ok.

```

local attr = node.has_attribute(item, 100)
if attr then
  local note = node.copy(tex.box[attr])
  local upward = note.depth
    - node.tail(note.list).depth
  if upward > remainingheight then
    upward = remainingheight - upward
  else
    upward = 0
  end
end

```

Now we insert the note box after the line: first, we add a negative vertical kern to account for the upward shift (possibly 0), plus the line's depth and the note's height (i.e. the height of its first line), so it is level with the line. We then set the note's height and depth to 0, so it doesn't take up space on the page. (Since the kern becomes the head of the list, we have to explicitly set `note.list` to it, otherwise `TeX` still thinks the previous head is the good one.)

```

local kern = node.new(KERN, 1)
kern.kern = upward - note.height
  - item.depth
node.insert_before(note.list,
  note.list, kern)
note.list = kern
note.height, note.depth = 0, 0

```

Finally, we insert the note and set its horizontal `shift` (here it goes into the right margin, but this should depend on whether the page is even or odd), and reset `first` and `remainingheight`, the latter to `upward` so the vertical shift of the current note (if any) is taken into account for the following one. The rest of the code is the end of the `attr` conditional (false, so we add the line's height to the `remainingheight`) and the end of the main loop.

```

node.insert_after(head, item, note)
note.shift = tex.hsize + tex.sp("1em")
first = true
remainingheight = upward
else
  remainingheight = remainingheight
    + item.height
end
end
item = item.prev
end
end

```

When a page is found good, before we ship it out (and before we add inserts too), we feed it to the function, so notes are added. For instance, a very simple output routine would be:

```

\output{%
  \directlua{%
    process_marginalia(tex.box[255].list)
  }%
  \shipout\box255}

```

The important part is, of course, the Lua code.

## Conclusion

Lua`TeX` has much to offer: UTF-8 encoding, non-TFM fonts, a comfortable programming language, ... Access to `TeX`'s internals is, to me, one of its most valuable features: it enables the user to do things that were previously unthinkable, and gives such control over typography that the software's limitations almost vanish, as if we were working on a hand press—except we don't manipulate metal, but nodes.

A final note: in this paper, functions have been added to callbacks with Lua`TeX`'s bare mechanism. If two functions are added to the same callback this way, the second erases the first. To do this properly, the `luatexbase` package can be used for plain `TeX` and `LATeX`, and it is taken care of in `ConTeXt`.

The next page shows examples of our three programs. First comes the page color, displaying a typeset text and its translation to shades of grey; the second text uses font expansion to show the resulting improvement in justification. Then are examples of underlining and marginal notes. The text used is the first page of Robert Coover's novel *The Adventures of Lucky Pierre*.

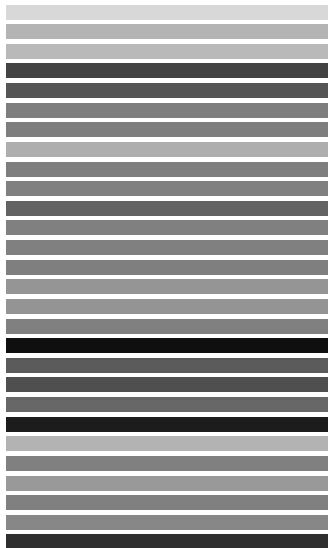
◇ Paul Isambert  
 Université de la Sorbonne Nouvelle  
 France  
 zappathustra (at) free dot fr



In the darkness, softly. A whisper becoming a tone, the echo of a tone. Doleful, incipient lament blowing in the night like a wind, like the echo of a wind, a plainsong wafting silently through the windy chambers of the night, wafting unisonously through the spaced chambers of the bitter night, alas, the solitary city, she that was full of people, thus a distant and hollow epiodion laced with sibilants bewailing the solitary city.

And now, the flickering of a light, a pallor emerging from the darkness as though lit by a candle, a candle guttering in the cold wind, a forgotten candle, hid and found again, casting its doubtful luster on this faint white plane, now visible, now lost again in the tenebrous absences behind the eye.

And still the hushing plaint, undeterred by light, plying its fricatives like a persistent woeful wind, the echo of woe, affanato, piangevole, a piangevole wind rising in the fluttering night through its perfect primes, lamenting the beautiful princess become an unclean widow, an emergence from C, a titular C, tentative and parenthetical, the widow then, weeping sore in the night, the candle searching the pale expanse for form, for the suggestion of form, a balm for the anxious eye, weeping she weepeth.



In the darkness, softly. A whisper becoming a tone, the echo of a tone. Doleful, incipient lament blowing in the night like a wind, like the echo of a wind, a plainsong wafting silently through the windy chambers of the night, wafting unisonously through the spaced chambers of the bitter night, alas, the solitary city, she that was full of people, thus a distant and hollow epiodion laced with sibilants bewailing the solitary city.

And now, the flickering of a light, a pallor emerging from the darkness as though lit by a candle, a candle guttering in the cold wind, a forgotten candle, hid and found again, casting its doubtful luster on this faint white plane, now visible, now lost again in the tenebrous absences behind the eye.

And still the hushing plaint, undeterred by light, plying its fricatives like a persistent woeful wind, the echo of woe, affanato, piangevole, a piangevole wind rising in the fluttering night through its perfect primes, lamenting the beautiful princess become an unclean widow, an emergence from C, a titular C, tentative and parenthetical, the widow then, weeping sore in the night, the candle searching the pale expanse for form, for the suggestion of form, a balm for the anxious eye, weeping she weepeth.

And now, the flickering of a light, a pallor emerging from the darkness as though lit by a candle, a candle guttering in the cold wind, a forgotten candle, hid and found again, casting its doubtful luster on this faint white plane, now visible, now lost again in the tenebrous absences behind the eye.

And still the hushing plaint, undeterred by light, plying its fricatives like a persistent woeful wind, the echo of woe, **affanato**, **piangevole**, a piangevole wind rising in the fluttering night through its perfect primes, lamenting the beautiful princess become an unclean widow, an emergence from C, a titular C, tentative and parenthetical, the widow then, weeping sore in the night, the candle searching the pale expanse for form, for the suggestion of form, a balm for the anxious eye, weeping she **weepeth**.

*'Affanato' means*

*'anguished'*

*'Piangevole' means*

*'plaintive'*

*'Weepeth' is an archaic form of 'weeps'*

## Some misunderstood or unknown L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> tricks (II)

Luca Merciadri

### 1 Introduction

L<sup>A</sup>T<sub>E</sub>X is written in such a way that even skilled T<sub>E</sub>Xnicians sometimes learn new tricks, or come to problems or errors that they cannot easily solve, or explain. This time, our article will be divided in two (imaginary) parts: the first (Section 2) will treat

1. Avoiding erroneous references for floats, and the second (the rest) will give, as in my preceding paper (Merciadri, 2010), some ways to achieve special things in L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>. These tricks are often explained on the Internet, but can be difficult to find. Specifically, the second part will speak about

2. Exporting spreadsheets into L<sup>A</sup>T<sub>E</sub>X,
3. Writing QED symbols as nicely as possible,
4. Counting the number of pages and tables,
5. Writing messages on would-be blank pages,
6. Writing dots in matrices,
7. Drawing logic gates,
8. Writing enumerations with textcircled numbers.

### 2 Avoiding erroneous references for floats

When writing a paper with L<sup>A</sup>T<sub>E</sub>X, the authors often let L<sup>A</sup>T<sub>E</sub>X do the cross-reference work. This results in a notable gain of time, because the work for every reference is automated. Consider a reference  $r$  declared using `\label{r}`. If  $r$  is cited, L<sup>A</sup>T<sub>E</sub>X will

1. Know its page number, which can be displayed and linked (if `hyperref` is used) using `\pageref{r}`,
2. Know its reference, meaning that it knows  $r$ 's place in the document structure.

But consider now a `tabular` environment placed in a `table` environment. Placing the `tabular` environment centered at the page is a good idea, thereby using

```
\begin{table}
  \begin{center}
    \begin{tabular}{cc}
      Text & Text
    \end{tabular}
  \end{center}
\end{table}
```

or its `\centering` variant. To link this table to a reference, one needs to place a `\label{reference}` in the `table` environment. One thing to remember is that `\label{}` always comes *after* `\caption{}`. That is, you must use neither

```
\begin{table}
\label{reference}
\caption{Name of the table.}
\begin{center}
  \begin{tabular}{cc}
    Text & Text
  \end{tabular}
\end{center}
\end{table}
```

nor

```
\begin{table}
\begin{center}
  \begin{tabular}{cc}
    Text & Text
  \end{tabular}
\end{center}
\label{reference}
\caption{Name of the table.}
\end{table}
```

You also need to *end* the `center` environment *before* using `\caption{}`. That is, you should not use

```
\begin{table}[!h]
\begin{center}
  \begin{tabular}{cc}
    Text & Text
  \end{tabular}
\caption{Name of the table.}
\end{center}
\label{tab:test}
\end{table}
```

but rather use

```
\begin{table}[!h]
\begin{center}
  \begin{tabular}{cc}
    Text & Text
  \end{tabular}
\caption{Name of the table.}
\label{tab:test}
\end{table}
```

Notice also the better reference: `tab:test` is clearer than `reference`. As `\centering` is local to the (most nested) environment which contains it, you can evidently replace the `center` environment by a simple `\centering`:

```
\begin{table}[!h]
\centering
\begin{tabular}{cc}
  Text & Text
\end{tabular}
\caption{Name of the table.}
\label{tab:test}
\end{table}
```

This concept is very important: some classes will not render a reference if the `\caption{}`–`\label{}` order is not respected. Even worse, others will put

unrelated reference numbers, such as `\thesection`, which can be disastrous: writing “thanks to Theorem  $x$ , we have [...]” is a good way not to lose the reader, but if  $x$  is a theorem number which does not exist, or which has no link with the citation, the whole paper might seem hastily written, or at least not edited, or simply confusing to the reader.

### 3 Exporting spreadsheets into L<sup>A</sup>T<sub>E</sub>X

It is sometimes desirable to export spreadsheets into L<sup>A</sup>T<sub>E</sub>X. It can be useful for many purposes, such as scientific experiments (collected data, for example), or financial reports. This is easily achieved with Calc2LaTeX ([calc2latex.sourceforge.net](http://calc2latex.sourceforge.net)).

### 4 Writing QED symbols as nicely as possible

When ending an environment, it is often desirable to let the reader know that the environment (property, theorem, etc.) has ended. It is often done using an elegant symbol: a QED symbol. This symbol might be anything you want, but such symbols are often small, and geometric shapes (squares, diamonds, ...). You can define many QED symbols. For example, you might define a QED symbol for each environment of your choice, or use the same one for every environment.

For example, to use  $\diamond$  as the QED symbol, you could simply use `\diamond`. The problem with such a simple approach is that, since you will end an environment with it (using `\diamond` or a homemade command such as `\myqedsymbol`), nothing guarantees that it will be placed correctly, *i.e.* that it will not begin a new line, or be placed at a new page.

So, you can use a tricky combination of `\hfill` and other commands to have your QED symbols placed as nicely as you want. Such a combination can be used to define a personal command such as `\qedsymbol`, like this:

```
\def\qedsymbol{%
  \mbox{}%
  \nolinebreak
  \hfill
  $\diamond$\% the qed symbol
  \medbreak
  \par
}
```

where the `\medbreak` is optional. You can then use `\qedsymbol`, or, better, a package which does it for you, such as `ntheorem`.

### 5 Counting the number of pages or tables

It might be interesting to know the number of pages of the current document. This can be done easily (MrUnix.de, 2010), e.g. by calling

```
\ref{TotPages}
```

which would give as output the number of pages. In an analogous way, one would for example want to know the number of tables of the document. This can be achieved using

```
\AbsTables
```

But before using the latter command, we must declare (in the preamble)

```
\newcommand*\OrigChapter{}
\let\OrigChapter\chapter
\newcounter{abstables}
\renewcommand*\chapter{%
  \addtocounter{abstables}{\value{table}}%
  \OrigChapter%
}
\newcommand*\AbsTables{0}
\makeatletter
\AtBeginDocument{%
  \AtEndDocument{%
    \addtocounter{abstables}{\value{table}}%
    \immediate\write\@mainaux{%
      \string\gdef\string\AbsTables{%
        \number\value{abstables}}%
    }%
  }%
}
```

```
\makeatother
```

For the former command, we need only

```
\usepackage{totpages}
```

in the preamble.

### 6 Writing messages on would-be blank pages

When reading a book, one sometimes encounters “blank” pages whose only text is some sentence like ‘This page intentionally left blank.’

This allows the reader to know that there has not been any printing issue with the book he is reading, and that the blank pages he sees are normal, and there for editorial reasons.

If you want such a message to appear in a L<sup>A</sup>T<sub>E</sub>X document whose class is `book`, you might redefine `\cleardoublepage` as follows:

```
\makeatletter
\def\cleardoublepage{\clearpage\if@twoside%
  \ifodd\c@page\else
  \vspace*{\fill}
  \hfill
  \begin{center}
    This page intentionally left blank.
  \end{center}
  \vspace{\fill}
  \thispagestyle{empty}
  \newpage}
```



```
\if@twocolumn\hbox{}\newpage\fi\fi\fi
}
```

```
\makeatother
```

in the preamble. The pages which would otherwise be left blank will now contain this message.

## 7 Writing dots in matrices

If sometimes happen to write special matrices, such as matrices where elements of one column could be moved to the next column, because other elements could replace them. An example is given by

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & \vdots \\ & & i \end{pmatrix}$$

This can be achieved using

```
\left(
\begin{array}{@{}cc@}{c@}{c@}{c@}{}
```

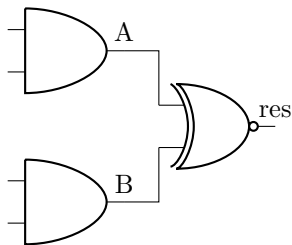
```
a & b & & c \\
d & e & & f \\
g & h & & \makebox[2\arraycolsep]{\smash{\vdots}}
& i
```

```
\end{array}
\right)
```

in a math environment. Thanks to Philipp Stephani for this trick.

## 8 Drawing logic gates

I found myself disappointed when looking for a simple way to *draw logic gates*. After much research, I found `circuitikz`, which allows you to write “traditional” circuits (that is, circuits with simple resistances, generators, inductors, ...), but also



This part of a circuit is created with

```
\begin{circuitikz} \draw
(0,2) node[and port] (myand1) {}
(0,0) node[and port] (myand2) {}
(2,1) node[xnor port] (myxnor) {}
(myand1.out) node[above] {A} -| (myxnor.in 1)
(myand2.out) node[above] {B} -| (myxnor.in 2)
(myxnor.out) node[above] {res};
\end{circuitikz}
```

and

```
\usepackage{tikz}
\usepackage{circuitikz}
```

in the preamble. There are other features to this package. This example was given to me by Massimo Redaelli. You might check the package’s manual (Redaelli, 2009) for other details.

## 9 Writing enumerations with textcircled numbers

Using the `enumerate` package, you can write

```
\begin{enumerate}[\textcircled{\arabic{enumi}}]
\item Item 1
\item Item 2
\item \ldots
\item Item $n$
\end{enumerate}
```

for such a result:

- ① Item 1
- ② Item 2
- ③ ...
- ④ Item  $n$

This is simple to achieve, and might improve some enumerated lists. Do not forget to put

```
\usepackage{enumerate}
```

in the preamble.

- ◊ Luca Merciadri  
University of Liège  
Luca.Merciadri (at) student dot ulg dot  
ac dot be  
<http://www.student.montefiore.ulg.ac.be/~merciadri/>

## References

- Merciadri, Luca. “A Practical Guide to L<sup>A</sup>T<sub>E</sub>X Tips and Tricks”. 2009.
- Merciadri, Luca. “Some misunderstood or unknown L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> tricks”. *TUGboat* **31**(1), 76–78, 2010. <http://tug.org/TUGboat/31-1/tb97merciadri.pdf>.
- MrUnix.de. “Gesamtanzahl Seiten, Abbildungen usw.—mrunix.de”. 2010. <http://www.mrunix.de/forums/showthread.php?t=56716>.
- Redaelli, Massimo. “CircuiTikZ”. 2009. <http://mirror.ctan.org/graphics/pgf/contrib/circuitikz/doc/latex/circuitikz/circuitikzmanual.pdf>.

# L<sup>A</sup>T<sub>E</sub>X3 News

Issue 4, July 2010

Now that we're back from the T<sub>E</sub>X Users Group conference in San Francisco, it's time to discuss what's been going on over the last six months. Due to some extra travel plans after the conference, this issue is slightly late in coming out.

## *expl3 in practice*

Joseph Wright and Will Robertson have both released significant new versions of their packages, resp., `siunitx` and `fontspec`. These have been re-written in the L<sup>A</sup>T<sub>E</sub>X3 programming language `expl3`, which we have discussed here previously. Using `expl3` for production code has been very successful, both in demonstrating that the concepts are sound and highlighting areas that still need some attention.

In the case of `fontspec`, `expl3` programming is being used to target L<sup>A</sup>T<sub>E</sub>X running on either X<sub>Y</sub>L<sup>A</sup>T<sub>E</sub>X and Lua<sup>A</sup>T<sub>E</sub>X. In the latter case, the package is a mixture of Lua code and `expl3` code; Will presented the `unicode-math` package at TUG 2010, which is developed in the same style.

## *New xpackages*

Frank Mittelbach has started to work on a new experimental L<sup>A</sup>T<sub>E</sub>X3 package `xhead` that provides templates for one of the most complex areas of document design: section headings and document divisions. This is the beginning of an ambitious idea to map out the requirements for typesetting most documents currently processed with L<sup>A</sup>T<sub>E</sub>X.

One of the challenges here is providing a “natural” design language for describing the two-dimensional spatial relationships of objects participating in the design, e.g., the placement of a heading number in relation to the heading title, a possible sub-title, etc. In answer to this challenge Frank developed the `xcoffin` package, which he presented at TUG 2010. It is designed as a high-level interface for placing and aligning boxes on a page, allowing a ‘designer’s approach’ for indicating the positional relationship between boxes. (A ‘coffin’ is a box with handles.) As an example, it is possible to represent ideas such as ‘align the lower-left corner of box A with the upper-right corner of box B after rotating it ninety degrees’, without having to calculate the intermediate positions.

We expect a future version of `xcoffin` (after some further work on its interface layer and its internal

implementation) to play a major role in all packages providing layout templates for higher-level document objects, such as table of contents designs, floats, etc.

Finally, Joseph Wright has begun work with the current ‘galley’ packages, producing the new, minimal, `xgalley` based on `xfm-galley` as a testbed for what we need and what will work.

## *Developments with expl3*

Meanwhile, Joseph’s *also* been writing a new floating-point calculation module, called `l3fp`, for `expl3`. This module allows manipulation and calculation of numbers with a much larger range than T<sub>E</sub>X allows naturally. The `l3fp` module has already been utilised in the `xcoffin` code for calculations such as coordinate rotations and intersection points of vectors.

The modules `l3io` and `l3file` have been revised, re-thinking the way that read and write streams are dealt with. T<sub>E</sub>X has a hard limit of sixteen input and output streams open at any one time, and the new implementation for `expl3` provides more flexibility in how they are allocated; there’s now much less chance of running into a ‘No room for a new `\read`’ (or `\write`) error.

Sometimes we discuss ideas for `expl3` that *don’t* end up making it into the final code. One example of this is the concept of having ‘local registers’ for integers, boxes, and so on, that do not survive outside of the group they are defined in (in contrast to Plain T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X, where allocators such as `\newcount` and `\newbox` are always global). Despite the scope for some small benefit, we decided that the extra complexity that the additional functions required, in both syntax and documentation, was not justified.

## *TUG 2010 reflections*

Our interpretation of the broad themes discussed at the conference are that T<sub>E</sub>X-based systems are still thriving and there are some big problems to solve with robust solutions to transform L<sup>A</sup>T<sub>E</sub>X source, including mathematics, into a form such as HTML. While there are big pushes for standardising various aspects of the L<sup>A</sup>T<sub>E</sub>X syntax, we also believe that it is L<sup>A</sup>T<sub>E</sub>X’s very flexibility—its inherently non-standardised markup—that has allowed it to survive for so many years. There is a delicate trade-off here between moving forward into more standards-based territory while also retaining the extensibility of the third-party package system.

---

## From `\newcommand` to `\NewDocumentCommand` with `xparse`

Joseph Wright

### Abstract

The `xparse` package provides a new method for creating document macros, moving beyond `\newcommand`. With `xparse` it is possible for ordinary  $\LaTeX$  users to create functions with multiple optional arguments, stars and mixtures of these. This brief article highlights using the `xparse` approach for the  $\LaTeX$  user (as distinct from the  $\LaTeX$  programmer).

### 1 Introduction

In recent articles, I've been discussing how some of the ideas that the  $\LaTeX$ 3 Project have developed can be used by  $\LaTeX$  programmers today. However, most users of  $\LaTeX$  don't want to deal with the programming side: they just want to use  $\LaTeX$ . The existing  $\LaTeX$ 3 packages can already offer benefits directly to  $\LaTeX$  users. Here, I want to show how the `xparse` package ( $\LaTeX$ 3 Project, 2010) can be used to replace `\newcommand` with a much more powerful way of creating commands for day-to-day  $\LaTeX$  use.

Before getting started, let me pose the question 'Why would you want to replace `\newcommand`?' With `\newcommand`, you can make a macro that takes a number of mandatory arguments, or a macro where the first argument is optional and in square brackets, but that is it as far as variation goes. Anything else then needs the use of  $\TeX$  programming or internal  $\LaTeX$ 2 $\epsilon$  macros: not really helpful for end users. The macros that `\newcommand` creates are also 'fragile'. This shows up where you need to `\protect` things, which can be very confusing. Macros created using `xparse` are robust (*i.e.* not 'fragile'), and are therefore reliable in places like section headings.

### 2 Getting started with `xparse`

The `xparse` package is part of a larger bundle of material (`expl3` and `xpackages`) which the  $\LaTeX$ 3 Project has released to CTAN for general use and distribution. As such, it is included in  $\text{MiK}\TeX$  2.7,  $\TeX$  Live 2009, and later releases. If you are using an older  $\TeX$  distribution you can download both `expl3` and `xpackages` from CTAN, ready to install.

`xparse` can be loaded as usual for  $\LaTeX$ 2 $\epsilon$ :

```
\usepackage{xparse}
```

It adds a number of new macros to  $\LaTeX$ , but here I'll discuss just a few. The main one I'll be using is `\NewDocumentCommand`, which is the  $\LaTeX$ 3 version of  $\LaTeX$ 2 $\epsilon$ 's `\newcommand`.

### 3 Macros with no arguments

The simplest type of macro is one with no arguments at all. This isn't going to show off `xparse` very much but it's a starting point. The standard  $\LaTeX$ 2 $\epsilon$  method to make a macro with no arguments at all is

```
\newcommand\NoArgs{Text to insert}
```

which with `xparse` would instead read

```
\NewDocumentCommand\NoArgs{}{Text to insert}
```

That does not look too bad, I hope. Notice that I've got an empty set of braces in the `xparse` case: this is where the arguments for the new macro would be listed. With `\NewDocumentCommand` there always has to be a list of arguments, even if it is empty. That's in contrast with the `\newcommand` approach, where we only need to mention arguments when there are any.

### 4 Macros with simple mandatory arguments

The most common type of argument for a macro is a mandatory one. With `\newcommand`, we'd give a number of arguments to use:

```
\newcommand\OneArg[1]{Text #1}
\newcommand\TwoArgs[2]{Text #1 and #2}
```

`\NewDocumentCommand` is a bit different. Since it can work with different types of arguments, each is individually specified with a letter. A mandatory argument is 'm', so we'd need

```
\NewDocumentCommand\OneArg{m}{Text #1}
\NewDocumentCommand\TwoArgs{mm}{Text #1 and #2}
```

This is still pretty similar to `\newcommand`: the useful stuff starts when life gets a little more complicated.

### 5 Macros with one or more optional arguments in square brackets

To get something clever out of `xparse`, the arguments need to be a little more varied than we've seen so far. Let's look at optional arguments, which  $\LaTeX$  puts in square brackets. If I want the first argument to be optional, then `\newcommand` can help:

```
\newcommand\OneOptOfTwo[2] []
  {Text with #2 and perhaps #1}
\newcommand\OneOptOfThree[3] []
  {Text with #2, #3 and perhaps #1}
```

If I want anything else, I'm on my own. First, let's do the above examples using `xparse`. There, an optional argument in square brackets, as in `\newcommand`, is specified by 'O' followed by {}:

```
\NewDocumentCommand\OneOptOfTwo{O{}m}
  {Text with #2 and perhaps #1}
\NewDocumentCommand\OneOptOfThree{O{}mm}
  {Text with #2, #3 and perhaps #1}
```

How about two optional arguments? You can't do this with `\newcommand`. Although it is provided by add-ons like the `twoopt` package (Oberdiek, 2010), `xparse` is overall much more flexible. All we need to do is use two `O{}` statements.

```
\NewDocumentCommand\TwoOptOfThree{O{}O{}m}
  {Text with #3 and perhaps #1 and #2}
```

Then we can do:

```
\TwoOptOfThree{Mandatory}
\TwoOptOfThree[Optional1]{Mandatory}
\TwoOptOfThree[Optional1][Optional2]{Mandatory}
\TwoOptOfThree[] [Optional2]{Mandatory}
```

(You can't give only the second optional argument: you still need an empty first one.)

What if we want a default value for the optional argument? With `\newcommand`, that would be

```
\newcommand\OneOptWithDefault[2][myval]
  {Text using #1 (could be 'myval') and #2}
```

This is where the braces come in: whatever we put inside the braces becomes the default value.

```
\NewDocumentCommand\OneOptWithDefault
  {O{myval}m}
  {Text using #1 (could be 'myval') and #2}
```

The same idea applies to each optional argument: whatever is in braces after the `O` is the default value.

## 6 More complicated optional values

You might be wondering why we need the `{}` after `O` when there is no default value: why not just `o`? Well, there is `o` as well, but it's a bit different. Unlike `\newcommand`, `\NewDocumentCommand` can tell the difference between an optional argument that is not given and one that is empty. To do that, it provides a test to see if the argument is empty:

```
\NewDocumentCommand\OneOptOfTwoWithTest{om}
  {\IfNoValueTF{#1}
   {Do stuff with #2 only}
   {Do stuff with #1 and #2}}
```

Don't worry if you forget to do the test: the special marker that is used here will print `'-NoValue-'` as a reminder!

Sometimes you might want two different optional arguments, and be able to tell which is which. This can be done by using something other than square brackets, often angle brackets (`<` and `>`). We can do that using the letter `d` (or `D` if we give a default).

```
\NewDocumentCommand\TwoTypesOfOpt{D<>{}O{}m}
  {Text using #1, #2 and #3}
```

What input syntax does this recognize? Let's look at some examples:

```
% One mandatory
\TwoTypesOfOpt{text}
% A normal optional
\TwoTypesOfOpt[text]{text}
% A special optional
\TwoTypesOfOpt<text>{text}
% Both optionals
\TwoTypesOfOpt<text>[text]{text}
```

How did that work? The first two characters after the `D` are used to find the optional argument, so in this case `<` and `>`. The same could be done with `(` and `)`, or almost anything else you fancy.

Another common idea in  $\LaTeX$  is to use a star to indicate a special variant of a macro. Creating those with `\newcommand` is difficult, but it is easy with `\NewDocumentCommand`:

```
\NewDocumentCommand\StarThenArg{sm}
  {\IfBooleanTF#1
   {Use #2 with a star}
   {Use #2 without a star}}
```

Here, `'s'` represents a star argument. We see that it ends up as `#1`, while the mandatory argument is `#2`. We also need a test to determine if there is a star (`\IfBooleanTF`). This doesn't mention stars as the test can be used for other things.

## 7 Summary

There is more to `xparse` than I've mentioned here, but I hope that this gives a flavour of what it can be useful for. To get more flexibility there is a bit more to think about compared to `\newcommand`, but the overall consistency is hopefully worth it. By using `xparse` a whole range of argument arrangements can be supported without needing to know any  $\LaTeX$  internal functions. This makes the process of creating commands much clearer.

## References

- $\LaTeX$ 3 Project. "The `xparse` package: Generic document command processor". Part of the `xpackages` bundle, [mirror.ctan.org/macros/latex/contrib/xpackages](http://mirror.ctan.org/macros/latex/contrib/xpackages), 2010.
- Oberdiek, Heiko. "The `twoopt` package". Part of the `oberdiek` bundle, [mirror.ctan.org/macros/latex/contrib/oberdiek](http://mirror.ctan.org/macros/latex/contrib/oberdiek), 2010.

◇ Joseph Wright  
Morning Star  
2, Dowthorpe End  
Earls Barton  
Northampton NN6 0NH  
United Kingdom  
[joseph.wright \(at\) morningstar2 dot co dot uk](mailto:joseph.wright(at)morningstar2 dot co dot uk)

## Tagged PDF in ConTeXt

Hans Hagen

### 1 Introduction

Occasionally users asked me if ConTeXt can produce tagged PDF and the answer to that has been: I'll implement it when I need it. However, users tell me that publishers more and more demand tagged PDF files, although one might wonder what for, except maybe for accessibility. Another reason for not having spent too much time on it before is that the specification was not that inviting.

At any rate, when I saw Ross Moore<sup>1</sup> presenting tagged math at TUG 2010, I decided to look up the spec once more and see if I could get into the mood to implement tagging. Before I started it was already clear that there were a couple of boundary conditions:

- Tagging should not put a burden on the user but users should be able to tag themselves.
- Tagging should not slow down a run too much; this is no big deal as one can postpone tagging till the last run.
- Tagging should in no way interfere with typesetting, so no funny nodes should be injected.
- Tagging should not make the code look worse, neither the document source, nor the low level ConTeXt code.

And of course implementing it should not take more than a few days' work, certainly not in an exceptionally hot summer.

You can 'google' for one of Ross's documents (like `DML_002-2009-1_12.pdf`) to see how a document source looks at his end using a special version of pdfTeX. However, the version on my machine didn't support the shown primitives, so I could not see what was happening under the hood. Unfortunately it is quite hard to find a properly tagged document so we have only the reference manual as starting point. As the pdfTeX approach didn't look that pleasing anyway, I just started from scratch.

Tags can help Acrobat Reader when reading out the text aloud. But you cannot browse the structure in the no-cost version of Acrobat and as not all users have the professional version of Acrobat, the fact that a document has structure can go unnoticed. Add to that the fact that the overhead in terms of bytes is quite large as many more objects are generated, and you will understand why this feature is not enabled by default.

<sup>1</sup> He is often exploring the boundaries of PDF, Unicode and evolving techniques related to math publishing so you'd best not miss his presentations when you are around.



Figure 1: A tag list in Acrobat.

### 2 Implementation

So, what does tagging boil down to? We can best look at how tagged information is shown in Acrobat. Figure 1 shows the content tree that has been added (automatically) to a document while figure 2 shows a different view.

In order to get that far, we have to do the following:

- Carry information with (typeset) text.
- Analyse this information when shipping out pages.
- Add a structure tree to the page.
- Add relevant information to the document.

That first activity is rather independent of the other three and we can use that information for other purposes as well, like identifying where we are in the document. We carry the information around using attributes. The last three activities took a bit of

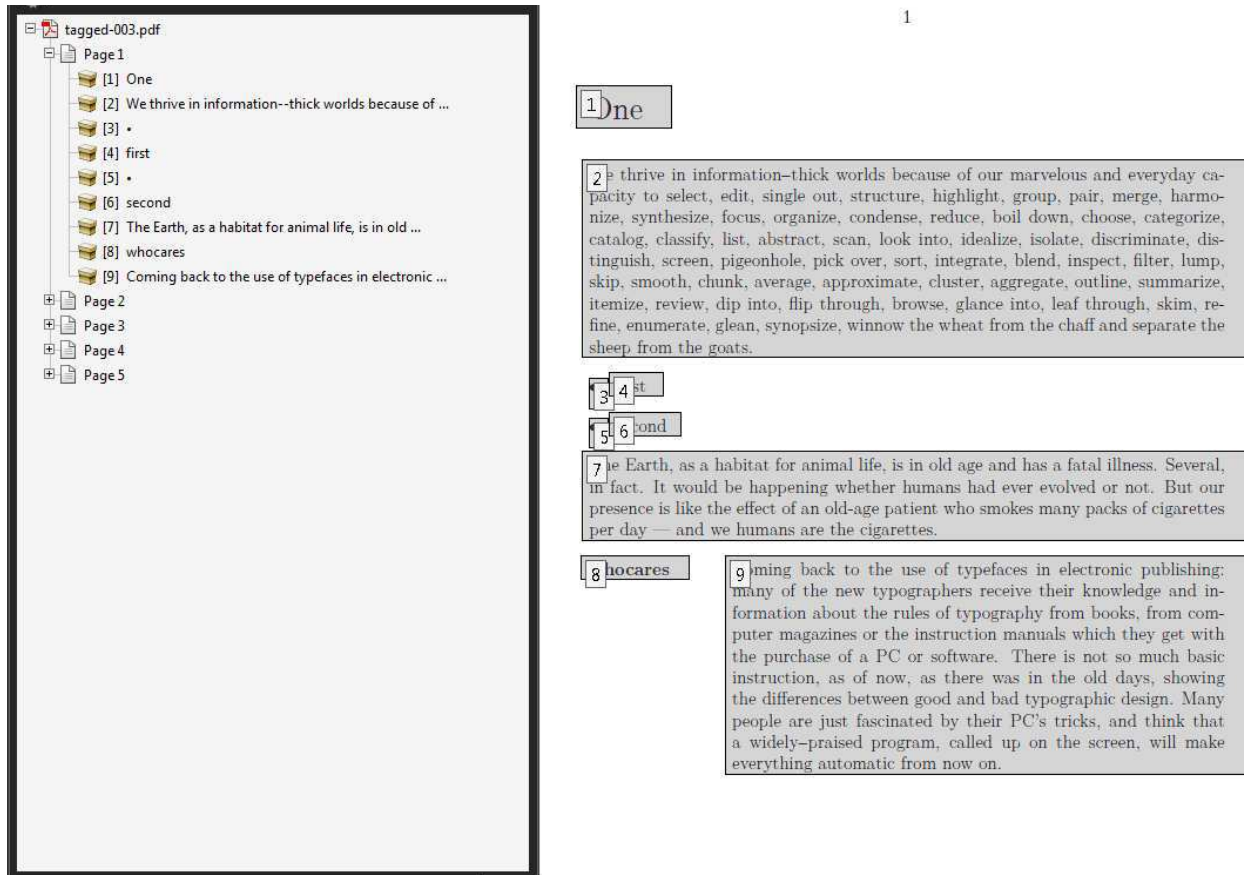


Figure 2: Acrobat showing the tag order.

experimenting mostly using the “Example of Logical Structure” from the PDF standard 32000-1:2008.

This resulted in a tagging framework that uses explicit tags, meaning the user is responsible for the tagging:

```
\setupstructure[state=start,method=none]
\starttext
\startelement[document]
\startelement[chapter]
\startelement[p] \input davis \stopelement\par
\stopelement

\startelement[chapter]
\startelement[p] \input zapf \stopelement\par
\startelement[whatever]
\startelement[p] \input tufte \stopelement\par
\startelement[p] \input knuth \stopelement\par
\stopelement
\stopelement
...
\stopelement
\stoptext
```

Since this is not much fun, we also provide an automated variant. In the previous example we

explicitly turned off automated tagging by setting method to none. By default it has the value auto.

```
\setupstructure[state=start]
% default is method=auto

\definedescription[whatever]
\starttext
\startfrontmatter
\startchapter[title=One]
\startparagraph \input tufte \stopparagraph
\startitemize
\startitem first \stopitem
\startitem second \stopitem
\stopitemize
\startparagraph \input ward \stopparagraph
\startwhatever {Hermann Zapf} \input zapf
\stopwhatever
\stopchapter
\stopfrontmatter

\startbodymatter
...
```

If you use commands like `\chapter` you will not get the desired results. Of course these can be

supported but there is no real reason for it, as in MkIV we advise using the `start-stop` variant.

It will be clear that this kind of automated tagging brings with it a couple of extra commands deep down in ConTeXt and there (of course) we use symbolic names for tags, so that one can overload the built-in mapping.

```
\setuptaglabeltext[en][document=text]
```

As with other features inspired by viewer functionality, the implementation of tagging is independent of the backend. For instance, we can tag a document and access the tagging information at the TeX end. The backend driver code maps tags to relevant PDF constructs. First of all, we just map the tags used at the ConTeXt end onto themselves. But, as validators expect certain names, we use the PDF rolemap feature to map them to (less interesting) names. The next list shows just a few of the currently used internal names, with the PDF ones between parentheses.

```
construct (Span), delimited (Quote),
delimitedblock (BlockQuote), description (Div),
...
tabulaterow (TR), verbatim (Code),
verbatimblock (Code), verbatimline (Code).
```

So, the internal ones show up in the tag trees as shown in the examples but applications might use the rolemap which normally has less detail.

Because we keep track of where we are, we can also use that information for making decisions.

```
\doifinelementelse{structure:section}
{yes} {no}
\doifinelementelse{structure:chapter}
{yes} {no}
\doifinelementelse{division:*-structure:chapter}
{yes} {no}
\doifinelementelse{division:*-structure:*}
{yes} {no}
```

As shown, you can use `*` as a wildcard. The elements are separated by `-`. If you don't know what tags are used, you can always enable the tag related tracker:

```
\enabletrackers[structure.tags]
```

This tracker reports the identified element chains to the console and log.

### 3 Special care

Of course there are a few complications. First of all the tagging model sort of contradicts the concept of a nicely typeset document where structure and outcome are not always related. Most TeX users are aware of the fact that TeX does not have space characters and does a great job on kerning and hyphenation. The tagging machinery on the other hand

uses a rather dumb model of strings separated by spaces.<sup>2</sup> But we can trick TeX into providing the right information to the backend so that words get nicely separated. The non-optimized function that does this looks as follows:

```
function injectspaces(head)
local p
for n in node.traverse(head) do
local id = n.id
if id == node.id("glue") then
if p and p.id == node.id("glyph") then
local g = node.copy(p)
local s = node.copy(n.spec)
g.char, n.spec = 32, s
p.next, g.prev = g, p
g.next, n.prev = n, g
s.width = s.width - g.width
end
elseif id == node.id("hlist")
or id == node.id("vlist") then
injectspaces(n.list,attribute)
end
p = n
end
end
```

Here we squeeze in a space (given that it is in the font which it normally is when you use ConTeXt) and make a compensation in the glue. Given that your page sits in box 255, you can do this just before shipping the page out:

```
injectspaces(tex.box[255].list)
```

Then there are the so-called suspects: things on the page that are not related to structure at all. One is supposed to tag these specially so that the built-in reading equipment is not confused. So far we could get around them simply because they don't get tagged at all and therefore are not seen anyway. This might well be enough of a precaution.

Of course we need to deal with mathematics. Fortunately the presentation MathML model is rather close to TeX and so we can map onto that. After all we don't need to care too much about back-mapping here. The currently present code is rather experimental and might get extended or thrown out in favour of inline MathML. Figure 3 demonstrates that a first approach does not even look that bad. In future versions we might deal with table-like math constructs, like matrices.

This is a typical case where more energy has to be spent on driving the voice of Acrobat but I will do that when we find a good reason.

<sup>2</sup> The search engine on the other hand is rather clever on recognizing words.





1 chapter

test oeps test whow test

test

`\whatever[goes]{here}`

Figure 4: Verbatim, including dedicated instances.

Contents

1 One 2

1.1 alpha 2

1.2 beta 2

1.3 gamma 2

1.4 delta 2

Figure 6: Tables of contents with specific entries tagged.

|         |         |
|---------|---------|
| test 11 | test 12 |
| test 21 | test 22 |
| test 33 |         |

test Coming back new typograp typography fr which they ge sic instruction between good by their PC's the screen, w

test Coming back new typograp typography fr which they ge sic instruction between good by their PC's the screen, w

ublishing: many of the ation about the rules of the instruction manuals There is not so much ba showing the differences ople are just fascinated program, called up on on.

ublishing: many of the ation about the rules of the instruction manuals There is not so much ba showing the differences ople are just fascinated program, called up on on.

Figure 5: Natural tables and the tabulate mechanism are both supported.

Index

o one 1, 2

t two 1, 2

Figure 7: A detailed view of registers is provided.

Tags

- <document>
  - <structure> chapter
    - <structurenumber>
    - <structuretitle>
    - <structurecontent>
      - <paragraph>
        - Let's see what a user defined command does:
        - <construct> notabene
          - whow
          - !
        - <float> figure
          - <floatcontent>
            - a simple graphic
          - <floatcaption>
            - Figure 1.1
          - <floattag>
            - test
        - <float> figure
        - <float> figure
          - <floatcontent>
            - <image>
              - PathPathPath
          - <floatcaption>
          - <mpgraphic>
            - Path
          - <paragraph>
            - Yet another paragraph.

# 1 chapter

Let's see what a user defined command does: **whow!**

a simple graphic

Figure 1.1 test

a simple graphic

Figure 1.2 test

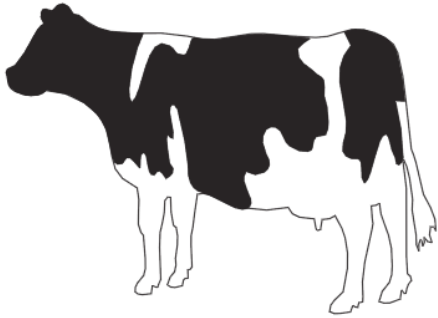


Figure 1.3 test

Yet another paragraph.

Figure 8: Float tags end up in the text stream. Watch the user defined construct.

sheep from the

- first<sup>1</sup>
- second

The Earth, as a  
in fact. It would  
presence is like  
per day — and

**whocares**

t test

Content Order Tags

Tags

- <document>
  - <division> frontpart
    - <structure> chapter
      - <structuretitle>
      - <structurecontent>
        - <paragraph>
          - We thrive in information--thick worlds because of ...
        - <itemgroup> itemize
          - <item>
            - <itemtag>
            - <itemcontent>
              - first1
            - <description> footnote
              - <descriptiontag>
                - 1
              - <descriptioncontent>
                - test
            - <item>
        - <paragraph>
        - <description> whatever

Figure 9: Footnotes are shown at the place in the input (flow).

## Introduction to colours in ConT<sub>E</sub>Xt MKiV

Luigi Scarso

### Abstract

This paper is a short introduction to colours from both theoretical and practical points of view. The last section is devoted to colour in ConT<sub>E</sub>Xt MkIV.

### 1 Theoretical colours

While light is a well-known physical phenomenon, its interaction with the human body is still a complex subject. An important part of this complexity is due to the eyes being sensitive to a narrow part of the spectrum in a way that is neither uniform nor linear with wavelength; they transmit their signals to the brain by the optic nerves where they are “elaborated” to obtain a stereoscopic colour image.

Leaving out the three-dimensional aspect of vision, the human eye has two groups of specialised cells: the cones, which show three peaks of sensitivity around 420–440 nm (“blue”), 530–540 nm (“green”) and 560–580 nm (“red/orange”); and the rods, which show a peak around 490–495 nm and are sensitive to low brightness. The fundamental law is empirical, due to Hermann Graßmann (1809–1877) around 1853 as a result of his experiments with “pure” colour sources. Graßmann was able to measure the same sensation of a colour  $C$  as a composition of sensations of 3 primary sources **R**ed, **G**reen and **B**lue weighted between 0 and 100. So, if  $C_1$  and  $C_2$  are two colours such that

$$\begin{aligned} C_1 &= r_1 \mathbf{R} + g_1 \mathbf{G} + b_1 \mathbf{B} \\ C_2 &= r_2 \mathbf{R} + g_2 \mathbf{G} + b_2 \mathbf{B} \end{aligned} \quad (1)$$

then the sensation given by a colour  $C_3$  that is a composition of  $C_1$  and  $C_2$  is

$$\begin{aligned} C_3 &= C_1 + C_2 \\ &= (r_1 + r_2) \mathbf{R} + (g_1 + g_2) \mathbf{G} + (b_1 + b_2) \mathbf{B} \end{aligned} \quad (2)$$

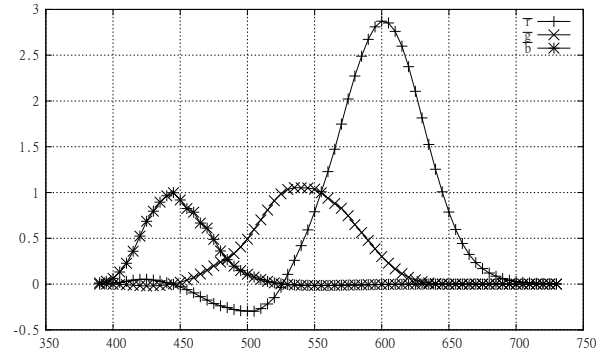
He also found that some colours  $C_k$  matched only if combined with a primary source:

$$C_k + r_k \mathbf{R} = g_k \mathbf{G} + b_k \mathbf{B} \quad (3)$$

i.e.

$$C_k = -r_k \mathbf{R} + g_k \mathbf{G} + b_k \mathbf{B} \quad (4)$$

Figure 1 shows a plot of the RGB colour-matching functions similar to those calculated from these experiments; we can see the red component is negative.



**Figure 1:** RGB colour-matching functions. X-axis is wavelength  $\lambda$  (nm). These data were obtained with a red primary of  $\lambda = 645.16\text{nm}$ , a green primary of  $\lambda = 526.32\text{nm}$  and a blue primary of  $\lambda = 444.44\text{nm}$ .

Given a colour  $C$  with a spectrum  $P(\lambda)$  it’s possible to calculate the components  $R, G, B$  with

$$\begin{aligned} R &= k \int_0^{+\infty} P(\lambda) \bar{r}(\lambda) d\lambda \\ G &= k \int_0^{+\infty} P(\lambda) \bar{g}(\lambda) d\lambda \\ B &= k \int_0^{+\infty} P(\lambda) \bar{b}(\lambda) d\lambda \end{aligned} \quad (5)$$

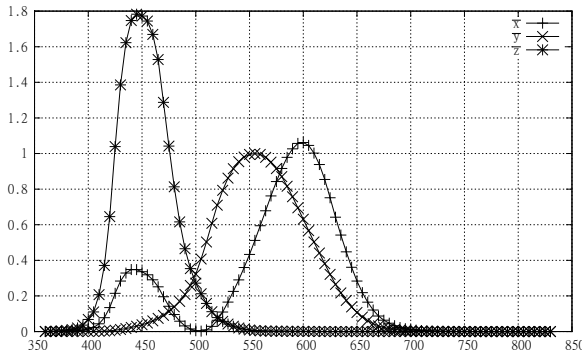
To avoid calculations with negative numbers and to make use of these data easier, in 1931 the CIE consortium introduced a linear and non-orthogonal transformation  $\mathbf{C}_{\mathbf{xr}}$  between RGB colour-match space and a new XYZ colour-match space called CIE XYZ 1931 where all values are positive. Hence we have  $X = \mathbf{C}_{\mathbf{xr}} \cdot \mathbf{R}$  i.e.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.49000 & 0.31000 & 0.20000 \\ 0.17697 & 0.81240 & 0.01063 \\ 0.00000 & 0.01000 & 0.99000 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (6)$$

As a consequence there is also a new set of colour-matching functions  $\bar{x}(\lambda), \bar{y}(\lambda), \bar{z}(\lambda)$  (see figure 2) so that a colour  $c$  with spectrum  $P(\lambda)$  has the components  $X, Y, Z$  where

$$\begin{aligned} X &= k \int_0^{+\infty} P(\lambda) \bar{x}(\lambda) d\lambda \\ Y &= k \int_0^{+\infty} P(\lambda) \bar{y}(\lambda) d\lambda \\ Z &= k \int_0^{+\infty} P(\lambda) \bar{z}(\lambda) d\lambda \end{aligned} \quad (7)$$

It’s easy to show that for a generic spectrum  $P(\lambda)$  not null (i.e. a visible colour) we always have  $X + Y + Z > 0$  so it makes sense to define  $x, y, z$  as



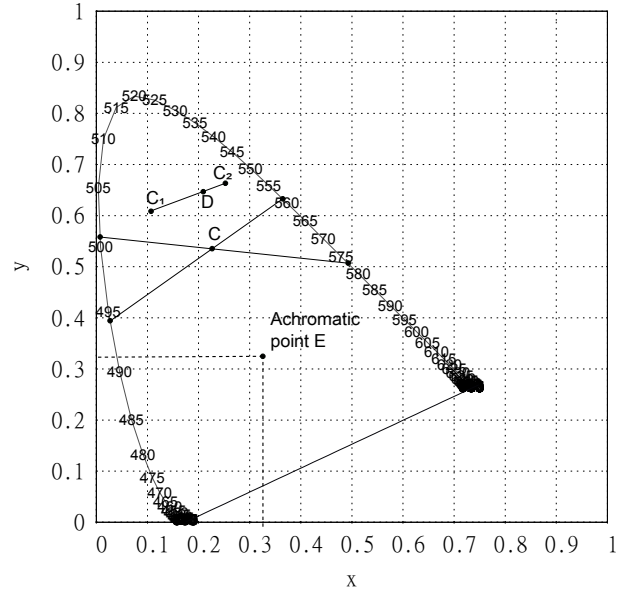
**Figure 2:** XYZ colour-matching functions per the CIE Consortium, 1931. X-axis is wavelength  $\lambda$  (nm). The CIE used slightly different virtual (i.e. calculated) primary sources from figure 1, namely 435.8nm for blue, 546.1nm for green, and 700.0nm for red. Each source is mathematically a delta (impulsive) function with amplitude +0.0601, +4.5907 and +1.0000.

$$\begin{aligned} x &= \frac{X}{X + Y + Z} \\ y &= \frac{Y}{X + Y + Z} \\ z &= \frac{Z}{X + Y + Z} \end{aligned} \quad (8)$$

and hence for all colours we have  $x + y + z = 1$ ,  $0 \leq x \leq 1$ ,  $0 \leq y \leq 1$ ,  $0 \leq z \leq 1$ . This surface is then contained inside the plane  $\mathbf{x} + \mathbf{y} + \mathbf{z} - \mathbf{1} = \mathbf{0}$  and its vertical projection onto the x-y plane is the *chromaticity diagram* whose peculiar shape is sometimes referred as a horseshoe (see figure 3). It's important to understand that it describes the complete *gamut* of an idealised human eye (it's also referred to as the gamut diagram) and it's *independent* of any particular device: *all the colours that are visible by the human eye are inside the chromaticity diagram*.

This diagram has several properties:

- a flat and uniform power spectrum  $E(\lambda)$  has coordinates  $x = 1/3, y = 1/3, z = 1/3$ . It's called the achromatic point  $E$  and corresponds to a white light that can be used as a reference;
- there is no triangle with vertices in the gamut diagram that encloses all of the diagram itself (i.e. there are not three light sources that can produce all visible colours);
- given two points in the gamut (i.e. two real colours) all the colours in the straight line that joins them can be obtained by mixing the start and the end colours;
- given  $x, y, Y$  it's possible to calculate  $X, Y, Z$  and vice versa, so that the CIE XYZ 1931 colour space is equivalent to the CIE xyY



**Figure 3:** CIE xyY 1931 chromaticity diagram. The labels are the wavelengths  $\lambda$  in nm. The point  $E$  (the achromatic point) has coordinates  $x = \frac{1}{3}, y = \frac{1}{3}$ ; the colour  $C$  can be seen as a result of mixing two different sets of monochromatic lights. Also a colour  $D$  can be a mix of the colours  $C_1$  and  $C_2$ .

1931 colour space.

But probably the most important thing is that for a given colour, in most cases there are at least two independent sets of monochromatic lights that give the same colour perception when mixed (in figure 3 the two sets are  $(\lambda = 495\text{nm}, \lambda = 569\text{nm})$  and  $(\lambda = 500\text{nm}, \lambda = 575\text{nm})$ ) and this is the best synthesis of the complexity of the human colour perception.

The CIE XYZ 1931 colour-space has several important properties, but an important disadvantage is that it's still difficult to compute the “difference” of two colours. In fact, given  $C_1 = X_1, Y_1, Z_1$  and  $C_2 = X_2, Y_2, Z_2$  then  $\Delta C = \delta_2(C_1, C_2) = \sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2 + (Z_1 - Z_2)^2}$  is not adequate for practical purposes. For an effective Euclidean distance, researchers have found more useful non-linear transformations of the CIE XYZ 1931 colour space; one of the most used led to the CIE  $L^*a^*b^*$  1976 colour-space, with the coordinates  $L^*, a^*, b^*$  given by

$$\begin{aligned} L^* &= 116 \cdot \left[ f\left(\frac{Y}{Y_n}\right) - 16 \right] \\ a^* &= 500 \cdot \left[ f\left(\frac{X}{X_n}\right) - f\left(\frac{Y}{Y_n}\right) \right] \\ b^* &= 200 \cdot \left[ f\left(\frac{Y}{Y_n}\right) - f\left(\frac{Z}{Z_n}\right) \right] \end{aligned} \quad (9)$$

and

$$f(t) = \begin{cases} t^{\frac{1}{3}} & t > \left(\frac{6}{29}\right)^3 \\ \frac{1}{3} \left(\frac{29}{6}\right)^2 t + \frac{4}{29} & \text{otherwise} \end{cases} \quad (10)$$

and  $X_n, Y_n, Z_n$  is a reference white point (for example, the achromatic point  $E$  mentioned above).

This leads to the important problem of the reference white point. It's clear now that under a monochromatic light the colour perception of a surface is completely different from the perception of the same surface under a "white light" and hence the specification of a "white light" is of extreme importance. It's also appropriate to consider a reference white light that mimics the Sun's daylight, so that we can consider a sort of natural light. Fortunately, physics can help with the well-established concept of *black-body*.

An ideal black-body is an object that absorbs *all* incident electro-magnetic radiations and re-emits them in a characteristic and continuous spectrum. The key point is that this spectrum depends only on the temperature: at room temperature the black-body radiation is mostly infrared wavelengths (invisible to human eyes, which is why the black body looks black), around 2000 K it is red, around 6000 K white and around 10000 K blue.

Hence it is possible to specify a temperature that identifies *precisely the spectrum* of a source light that gives a characteristic *perceptual* sensation of a colour, usually referred as *colour temperature*  $T_c$ . Nevertheless, the measure of this perception is still subjective, so more precisely the standards talk of *correlated colour temperature (CCT)*  $T_{cp}$ . The CIE has specified several of these sources to be used as "white light" only, and not as a general mechanism to define a colour; here is an example of these references sources:

| Source | CCT    | Note                    |
|--------|--------|-------------------------|
| D50    | 5003 K | Daylight, horizon light |
| D55    | 5503 K | Daylight, midmorning    |
| D65    | 6504 K | Daylight, noon light    |
| D75    | 7504 K | Daylight, north sky     |
| E      | 5454 K | Equal energy            |
| F4     | 2940 K | Fluorescent, warm white |

Of course every source is a point on the chromaticity diagram (figure 3); for example, D50 = (0.34567, 0.35850) and D65 = (0.31271, 0.32902).

## 2 Practical colours

The colour spaces seen so far are exhaustive and device independent — and *theoretical*. Practical devices always have a gamut that is strictly contained in the CIE xyY 1931 diagram, and in the printing world what matters is the colour of a surface (usually a paper) that is determined by the reflection of the light of a source. Hence there are three effective way to obtain a colour:

**Addition.** This is how a monitor (CRT or LCD), a *transmitting medium*, works: red, green and blue pixels are side by side (hence not overlapping) and each can emit between 0 (off) and  $2^n$  (express in a convenient unit). For example, if we have  $n = 8$  and hence  $2^8 = 256$  levels for each pixel, a colour  $C$  can have RGB components  $(R, G, B) = (0x1C, 0x45, 0x3B)$  or hexadecimal value  $0x1C453B$ . The colour depth is  $8 * 3 = 24$  bits, hence we have max  $2^{24}$  colours (more than 16 million). Some RGB colours can be specified also with 16-bit pixel values ( $2^{48}$ , more than  $281 \cdot 10^{12}$ ) or as a real value between 0 and 1 e.g.  $(r, g, b) = (0.1, 0.2, 0.45)$ . It's important also to note that digital cameras and scanners store their data as RGB values (so more bits mean more precision for image processing) because most modern consumer transmitting devices share the same basic technical implementations (i.e. they have almost the same gamut).

**Subtraction.** With an ideally white source (1.0, 1.0, 1.0) the reflected colour of a surface (a type of *reflecting media*) can be described as  $(r, g, b) = (1-c, 1-m, 1-y)$ , where  $(c, m, y)$  synthesise the filtered portion of the spectrum of the ideal white light. This is how a digital colour printer works: each dot printed is obtained by an overlapping of 4 dots coloured cyan, magenta, yellow and black (called the *key colour*) where each is between 0.0 (not drawn) and 1.0 (fully drawn); the order of overlapping is also important in industrial printers, and is usually  $c, m, y, k$ . It's clear that the white colour  $r=1.0, b=1.0, g=1.0$  is  $c=0.0, m=0.0, y=0.0$  which means 'don't print anything' or, better, 'show the colour of the surface' which can be different from white (that's why these devices cannot print a white colour on a black paper). The black component is essential to ensure correct colours (for example  $(c, m, y) = (1.0, 1.0, 1.0)$  is usually a dark brown) hence a theoretical CMY colour is always translated to a practical CMYK; also common is to express values as real values between 0.0 and 1.0 or in percentages. Another important point is that a CMYK

colour is always defined with reference to white light (e.g. D50) otherwise it makes no practical sense.

**Mixing inks.** As seen in the colour  $D$  of figure 3, a colour can be a mixing, on a precise type of paper, of a set of industrial inks (often called spot colours) taken from a de facto standard colour catalogue in a precise quantity. This is how an offset press works; for example, colour PANTONE 567 EC from *Pantone colour bridge coated Euro 1st Edition*. The support is coated paper (can also be uncoated) and the source light is D50. There can also be RGB colour that at least gives an idea of the real colour (in this case 0x1C453B) but this is not always possible: there are inks without any RGB or CMYK representation.

Thus, every media has its own colour space (i.e. a specific gamut, a subset of the CIE XYZ 1931 diagram) and a way to walk between them is given by a *ICC colour profile*, which is a map between the specific gamut of the device and a standard colour space called the *Profile Connection Space*, that is in turn based on CIE XYZ 1931 with default source light D50. The key point is that this PCS colour space is device-independent, making it possible to compare two different device-dependent colour spaces, and also to specify a ‘colour rendering style’ of the device-dependent colour space to match the desired result. These ‘styles’, called *rendering intent*, are *absolute colorimetric*, *relative colorimetric*, *perceptual* and *saturation*. From ICC specifications, “the colorimetric rendering intents operate directly on measured colorimetric values, though possibly with correction for chromatic adaptation when the measured values were not calculated for the D50 PCS illuminant. The other rendering intents (perceptual and saturation) operate on colorimetric values which are corrected in an as-needed fashion to account for any differences between devices, media, and viewing conditions”.

Here is a short list of ICC profiles:

- `sRGB_v4_ICC_preference.icc`: RGB colour space. The profile of most of LCDs, scanners and digital cameras;
- `ISOcoated_v2_eci_300.icc`: CMYK colour space. To be used for machine-finished glossy or matte coated papers. It’s considered a more or less standard profile (Europe);
- `UncoatedFOGRA29.icc`: CMYK colour space. To be used for uncoated papers (Europe);
- `GRACoL2006_Coated1v2.icc`: CMYK colour space. To be used for machine-finished glossy or matte coated papers (USA).

A very useful program to explore the colour profiles is `transicc` from `little cms`, a C library also useful in implementing a colour management system. Here is a simple example on how to convert a red RGB colour  $C_0$  to the equivalent CMYK  $C_1$ ; note the ICC profiles and the `-t0` option that means the rendering intent is ‘perceptual’:

```
# transicc.exe -isRGB_v4_ICC_preference.icc
-oISOcoated_v2_300_eci.icc -v3 -t0
LittleCMS ColorSpace conversion calculator ...
Profile:
sRGB v4 ICC preference perceptual intent beta
Output profile:
ISO Coated v2 300% (ECI)
...
Enter values, 'q' to quit
R? 255
G? 0
B? 0
C=0.6210 M=99.6170 Y=89.6544 K=2.6841
[PCS] Lab=(48.3055,86.7471,68.7393)
XYZ=(37.1789,17.0361,0.7712)
```

It’s worth observing that the theoretical CMYK colour  $C_2=(0,1.0,1.0,0)$  has the coordinates [PCS] Lab=(49.4726,65.9508,52.1581) and thus  $\Delta E^* = \sqrt{(L_1 - L_2)^2 + (a_1 - a_2)^2 + (b_1 - b_2)^2} = 26.623$  while two colours are regarded as identical if  $\Delta E^* < 2.8$ .

### 3 Colours in ConT<sub>E</sub>Xt

The PDF Reference describes three families of colour spaces:

1. the *device colour spaces*: DeviceGray, DeviceRGB, DeviceCMYK;
2. the *CIE base colour spaces*: Lab, ICCBased, CalGray, CalRB;
3. the *specials*: Pattern, Indexed, Separation, DeviceN.

In ConT<sub>E</sub>Xt MkIV it’s possible to define a device colour space with the `\setupcolors` and `\definecolor` macros, as in

```
\setupcolors[state=start,rgb=yes,cmyk=yes]
\definecolor[BlueGRAY][s=0.1]
\definecolor[BlueRGB][r=.1,g=.1,b=1]
\definecolor[BlueCMYK][c=0.9,m=.0.909,y=0,k=0]
\definecolor[Blue][r=.1,g=.1,b=1,
c=0.9,m=.0.909,y=0,k=0]
```

Then we can use it by its name as `\color[Blue]{I’m blue}`. The GRAY colour space is the last resort, and the RGB colour space has precedence over CMYK, so Blue is an RGB colour; if we use `rgb=no,cmyk=yes`, then Blue is the inverted CMYK colour (1-r,1-g,1-b), not the one specified (the same for `rgb=yes,cmyk=no`), and with

`rgb=no, cmyk=no` all colours are converted to GRAY. So, we should pay attention to the colour definitions.

ConTeXt MkIV also manages colour transparency: in

```
\definecolor[BlueTrs][r=.1,g=.1,b=1,t=0.7,
                    a=normal]
```

the key `t` specifies a solid colour with `t=1.0` or full transparency with `t=0.0`, while the key `a` is the transparency alternative method (there are 13 alternatives). Still today transparency must be used with care because printing is not reliable (some printers simply reject PDFs with these colours even if solid).

For spot colours (the Separation special colour space) things are a bit more complex: we must ensure that the tint is exactly specified by its name (case and spaces matter!) and associated with a CMYK reference colour for a low-quality print (just to see it), hence the two-steps definition:

```
\setupcolors[state=start,rgb=no,cmyk=yes,
             spot=yes,overprint=yes]
\definecolor[Pantone294][c=1,m=.56,y=0,k=.18]
\definespotcolor[BlueSPOT][Pantone294]
                    [p=1,e=PANTONE 294 CV]
```

The `overprint=yes` setting ensures that black overprinting the spot colour will not knock out the colour.

As of August 2010, ConTeXt MkIV has started to support the CIE based colour space ICCBased for RGB, CMYK and GRAY (only) and limited to the entire document (not for single object colour). An ICC profile must first be registered into the `color-profiles.xml` file by filling these fields (the function `colors.iccprofiles` (filename, verbose) in `colo-icc.lua` can help here):

**filename:** the file name of the ICC profile;  
**colorspace:** the colour space of the profile;  
**class:** the device class of the profile (`prtr` for printer, `mnr` monitor, `scnr` scanner, `spac` space);  
**id:** identifier of the measured data (not for the profile) on which the profile relies; e.g. FO-GRA39;  
**info:** (optional) descriptive text about the profile;  
**checksum:** md5 checksum of the profile;  
**version:** version number of the profile in hex;  
**url:** url where the profile can be downloaded;  
**outputcondition:** (optional) information about print technology, paper type and weight.

The profiles for RGB, CMYK and GRAY colour spaces can be set for the entire document with

```
\setupcolors[state=start,rgb=yes,cmyk=yes]
\setupbackend[profile={sRGB_v4_ICC_preference.icc,
                    default_cmyk.icc, default_gray.icc}]
```

Supporting ICCBased colour spaces is part of wide support for PDF/X specifications, a subset of full PDF focused on achieving more reliability in exchanging PDF files by enforcing restrictions; for example, fonts must all be included. All PDF/X-\* are formalized in ISO standard 15930. The specifications currently under development for ConTeXt MkIV are PDF/X-1a:2001, PDF/X-1a:2003, PDF/X-3:2002, PDF/X-3:2003, PDF/X-4, and PDF/X-4p, where X-1\* is more restrictive than X-4\*. The latest, PDF/X-5, is left out.

To enable a format we again use `\setupbackend`:

```
\setupbackend[format=PDF/X-4,
              profile={sRGB_v4_ICC_preference.icc,
                    default_cmyk.icc, default_gray.icc},intent=
                    {ISO Coated v2 300\letterpercent\space (ECI)}]
```

Here we specify in `profile` the ICC profiles of the colour's document, while in `intent` we specify the ICC profile of the intended output, a coated paper in this case. This is a feature of PDF/X-4; PDF/X-1a:2003 permits only CMYK and spot colours.

As final note, it's important to understand that PDF/X is a complex subject and ConTeXt MkIV is not a preflight tool. A PDF made with inclusion of other PDFs as images are particularly fragile: it is easy to do with ConTeXt, but can easily lead to an invalid PDF; for example if the included PDF does not have all its fonts embedded, or contains an RGB colour, or a transparency that is prohibited by the format chosen. So it's better to have a preflight tool for checking, and at present only commercial tools are reliable (yes, room for improvements here).

#### 4 Reference information

- General information: [http://en.wikipedia.org/wiki/International\\_Commission\\_on\\_Illumination](http://en.wikipedia.org/wiki/International_Commission_on_Illumination)
- A useful link for theoretical information: <http://www.fho-empden.de/~hoffmann/howww41a.html>
- The diagrams were traced with data from: <http://cvrl.ioo.ucl.ac.uk>
- ICC information: <http://www.color.org>
- Useful PDF/X information at the site of the Ghent PDF Workgroup: <http://www.gwg.org>
- Implementation of PDF/X in ConTeXt MkIV: <http://wiki.contextgarden.net/PDFX>
- More information about colours in ConTeXt: <http://wiki.contextgarden.net/Colors>, <http://wiki.contextgarden.net/Reference/en/definecolor>

◇ Luigi Scarso  
 luigi.scarso (at) gmail dot com

## Generate T<sub>E</sub>X documents using pdfscript

Oleg Parashchenko

### Abstract

Generation of correct T<sub>E</sub>X files is actually a hard task with a number of peculiarities. Therefore, it is better to delegate this task to some library or tool. A tool already exists (T<sub>E</sub>XML); now it's time for a library.

The library pdfscript helps to create T<sub>E</sub>X files from Python. The API follows the L<sup>A</sup>T<sub>E</sub>X model: it represents environments, commands and their parameters as calls of the corresponding functions in the library.

The pdfscript interface can be used as a basis for object-oriented abstractions of document elements, so that the users may create PDF documents having no idea that T<sub>E</sub>X is inside.

### 1 Introduction

Automatic generation of T<sub>E</sub>X files is much harder than one might expect. Here are some cases where bugs are possible and attention is required:

- Special symbols should be escaped
- Non-latin letters should be handled
- A space after command names may be required: `\it text`, not `\ittext`
- An empty group after a command may be required: `\PDF{ } file`, not `\PDF file`
- Opening and closing curly braces should be balanced
- It is necessary to comment-out empty lines to avoid false paragraph breaks (and do you know for sure what an empty line is?)

It is easy to code all these requirements, but at the next level, when we have several different T<sub>E</sub>X-generating programs, we would like to put the code into a common library. I tried it and found that this is a challenging task, which required a lot of thinking and several attempts before the result was satisfactory.

To compare the result with existing approaches, I asked about related work in the newsgroup `comp.text.tex` [3]. Surprisingly, the only alternatives are the use of the “print” statements and templates. When a programmer generates T<sub>E</sub>X files, he surely develops some helper functions, but so far nobody has shared his experiences, or at least I failed to find such work.

In this article, I describe my steps in designing a T<sub>E</sub>X generation library named pdfscript. Then I use the library to re-typeset an excerpt from “Essential L<sup>A</sup>T<sub>E</sub>X” by Jon Warbrick [5] and show the

artifacts of refactoring the code. Finally, I make a summary of the pdfscript API and speculate on further development.

The “proof of the concept” implementation of the pdfscript library and the examples from this article are available from <http://uucode.com/download/pdfscript-article-examples-20100909.tar.gz>. Despite its experimental status, the code is ready for use by early adopters.

## 2 Designing the interface

This section describes the steps of the design process.

### 2.1 Sample T<sub>E</sub>X document

Let's start with a very simple document, which contains only a setup, a title and a few paragraphs. (For editorial reasons, the boilerplate text is corrupted by introducing line breaks.)

```
\documentclass[a4paper]{article}
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
\begin{document}
\section{De finibus bonorum et malorum}
Lorem ipsum dolor sit amet, consetetur sad
ips cing elit, sed diam nonumy eirmod tem
por invidunt ut labore et dolore...
```

```
Duis autem vel eum iriure dolor in hendrer
it in vulputate velit esse molestie conseq
uat, vel illum dolore eu feugiat...
```

```
Ut wisi enim ad minim veniam, quis nostrud
exerci tation ullamcorper suscipit loborti
s nisl ut aliquip ex ea commodo...
\end{document}
```

### 2.2 T<sub>E</sub>XML version

The first step in the search for an API was to create a T<sub>E</sub>XML representation. To learn T<sub>E</sub>XML, visit the homepage of the project — <http://getfo.org/texml> — or read my TUGboat article [4]. For the purposes of this paper, it is enough to know the basics:

- T<sub>E</sub>XML is an XML format
- The root element is named TeXML
- A T<sub>E</sub>X command is represented by an element `cmd` with the attribute `name`:

```
\command[options]{parameter}
≡
<cmd name="command">
  <opt>options</opt>
  <parm>parameter</parm>
</cmd>
```



- If a command or an environment has options or parameters, they are represented by elements `opt` and `parm`, as in the example above.
- An environment is represented by an element `env` with the attribute name:

```
\begin{itemize}
...
\end{itemize}
≡
<env name="itemize">
...
</env>
```

This knowledge is enough to rewrite the sample document in the TeXML format:

```
<TeXML>
<cmd name="documentclass">
  <opt>a4paper</opt><parm>article</parm>
</cmd>
<cmd name="usepackage">
  <opt>utf8</opt><parm>inputenc</parm>
</cmd>
<cmd name="usepackage">
  <opt>T1</opt><parm>fontenc</parm>
</cmd>
<env name="document">
  <cmd name="section">
    <parm>De finibus bonorum et..</parm>
  </cmd>
  <TeXML>Lorem ipsum dolor sit amet, con
    setetur sadipscing elitr, sed diam n
    onumy eirmod tempor invidunt ut l...
  </TeXML>
  <cmd name="par"/>
  <TeXML>Duis autem vel eum iriure dolor
    in hendrerit in vulputate velit esse
    molestie consequat, vel illum dol...
  </TeXML>
  <cmd name="par"/>
  <TeXML>Ut wisi enim ad minim veniam, q
    uis nostrud exerci tation ullamcorpe
    r suscipit lobortis nisl ut aliip...
  </TeXML>
</env>
</TeXML>
```

The rewriting process is mostly straightforward, but two points require additional comments.

First, the use of the element `TeXML` not only as the root, but also as a container for the text. It is needed here only to satisfy my XML-related experience, which recommends avoiding mixing text and elements without a reason.

Second, in the TeX version, the empty lines give implicit `\par` commands, while TeXML version uses

`par` directly. It is possible to generate empty lines, but this is bad style when using TeXML. And by the way, I dislike the version with `par` too. In my documents I prefer to wrap paragraphs to environments and hide `\par` in the environment definitions.

### 2.3 Direct Python counterpart of the TeXML version

The TeXML version has structured the document, and now it is easy to re-write it in Python:

```
import pdfscript
from pdfscript import opt, parm
doc = pdfscript.newdoc()
doc.cmd('documentclass',
  opt('a4paper'), parm('article'))
doc.cmd('usepackage',
  opt('utf8'), parm('inputenc'))
doc.cmd('usepackage',
  opt('T1'), parm('fontenc'))
indoc = doc.env('document')
indoc.cmd('section',
  parm('De finibus bonorum et malorum'))
indoc.text('Lorem ipsum dolor sit amet, co
  nsetetur sadipscing elitr, sed diam...')
indoc.cmd('par')
indoc.text('Duis autem vel eum iriure dolo
  r in hendrerit in vulputate velit e...')
indoc.cmd('par')
indoc.text('Ut wisi enim ad minim veniam,
  quis nostrud exerci tation ullamcor...')
h = open('30_direct.texml', 'w')
doc.get_root().writexml(h)
h.close()
```

The first line instructs Python to load the library `pdfscript`, the second line allows using the short names `opt` and `parm` instead of the fully qualified `pdfscript.opt` and `pdfscript.parm`.

Then we create a document and put the commands and the environment into it. The content of the article is put inside the environment `document` by attaching the commands to the variable `indoc`, which is associated with the environment.

Finally, we get the root node of the constructed XML document and save it into the file.

### 2.4 Improved Python code

Immediate experience with the code above suggests the following improvements:

- In the most cases, the arguments of `cmd` are the parameters for the command, therefore it is logical to make `parm` calls implicit.
- Instead of `cmd('name', ...)` or `env('name', ...)`, the alias `name('...')` looks better.

- The functions could accept more than one argument.

Implementing these ideas, we get the following Python code:

```
import pdfscript
from pdfscript import opt, par
doc = pdfscript.newdoc()
doc.documentclass(opt('a4paper'), 'article')
doc.usepackage(opt('utf8'), 'inputenc')
doc.usepackage(opt('T1'), 'fontenc')
indoc = doc.document()
indoc.section('De finibus bonorum et ...')
indoc.add('Lorem ipsum dolor sit amet, consetetur sadipscing elitr, se...', par())
indoc.add('Duis autem vel eum iriure dolor in hendrerit in vulputate ve...', par())
indoc.text('Ut wisi enim ad minim veniam, quis nostrud exerci tation u...', par())
h = open('50_final.texml', 'w')
doc.get_root().writexml(h)
h.close()
```

### 3 Observations on a real world example

To test if the `pdfscript` library is powerful enough, I tried to reproduce some real life  $\text{\LaTeX}$  with it. After wandering in the `doc` directory on CTAN, I decided to re-typeset the document “Essential  $\text{\LaTeX}$ ” [5]. Surprisingly, the task was challenging. Even though the  $\text{\LaTeX}$  code contained little markup, it was enough to clutter the Python counterpart. To introduce clarity to the code, a redesign was required.

After some thought, the definition of the notion “clarity” became ambitious: a programmer who has never heard of  $\text{\LaTeX}$  should understand each line of the code. This approach produced a few artifacts:

- Python document classes
- Python macros
- Python active strings

#### 3.1 Python Document Class

Let’s consider the high-level structure of the “Lorem ipsum” example:

```
10 doc = pdfscript.newdoc()
20 doc.documentclass(...)
30 doc.usepackage(...)
40 doc.usepackage(...)
50 indoc = doc.document()
60 indoc.section(...)
70 indoc.add(..., par())
80 indoc.add(..., par())
90 indoc.text(...)
```

Let me turn into a non- $\text{\LaTeX}$  programmer and read this code. Here would be my comments:

- (10) Ok, create a new default document. (Wrong!)
- (20) This line probably defines the layout and formatting of the document I’m going to create. Why not join (10) and (20)?
- (30), (40) Some formatting plugins are loaded. WTF ([2])? What is T1? Do I really need these lines? I do a “Lorem ipsum” example, not something special. If this trivial test requires some functionality, it should be automatically loaded by default.
- (50) WTF? I’ve already created the document, why create it once more?
- (60) Good, a section is created. The argument is the title.
- (70) Looks like a paragraph is created. But what is this `par()` inside `add()`? Is it an additional vertical space to separate paragraphs, like pressing `<ENTER>` twice in OpenOffice or Word?
- (70), (80) Stylistic note. The paragraphs should belong to sections, not to the document itself.

Having these remarks in mind, I recoded the high-level structure in this way:

```
doc = esla.doc()
sect = doc.section(...)
sect.para(...)
sect.para(...)
sect.para(...)
```

The only question about this re-worked code fragment is what does `esla` in the first line mean. A programmer can guess that it is some Python package which assists in creation of the documents and hides the formatting in the commands `section()` and `para()`. Correct. For a  $\text{\LaTeX}$  user, this package is a Python version of a document class or a package. The name `esla` is an abbreviation for “Essential  $\text{\LaTeX}$ ”—I’m leaving “Lorem ipsum” test and starting work on the challenging example.

#### 3.2 Python macros

After the high-level structure is improved, time to switch to the inline markup. Here is a fragment of the source code:

```
You then get \LaTeX{} to process the file,
and it creates a new file of typesetting c
ommands; this has the same name as your fi
le but the “\fn{.TEX}” ending is replace
d by “\fn{.DVI}”. This stands for ‘{\it
D\}/}e{\it v\}/}ice {\it I\}/}ndependent’ ...
```

The direct transcription is a nightmare:

```
sect.para(
'...You then get ',
cmd('LaTeX'),
```

```
' to process the file, and it creates a
new file of typesetting commands; this
has the same name as your file but the',
verbatim(' '),
cmd('fn', '.TEX'),
verbatim("''"),
' ending is replaced by ',
verbatim(' '),
cmd('fn', '.DVI'),
verbatim("''"),
'. This stands for ',
group(cmd('it'), 'D', verbatim('\')),
'e',
group(cmd('it'), 'v', verbatim('\')),
'ice ',
group(cmd('it'), 'I', verbatim('\')),
"ndependent' ...")
```

Switching back to code review mode:

- The command `LaTeX` probably produces the logo.
- The lines with `fn` and `it` produce some formatting. But why is the usage different? `fn` has an argument, and `it` is enclosed in a group.
- Well, I think it switches to another formatting forever and the group limits this forever. But the construction `\` makes no sense to me.
- There is too much repetition, I don't like to code that way.

Unifying the `fn` and `it` interface, hiding the details and removing the repetitions, we get a better result:

```
sect.para(
'...You then get ', cmd('LaTeX'),
' to process the file, and it creates a
new file of typesetting commands; this
has the same name as your file but the',
fn('.TEX'), ' ending is replaced by ',
fn('.DVI'), '. This stands for ',
it('D'), 'e', it('v'), 'ice ',
it('I'), "ndependent' ...")
```

The functions `fn` and `it` can be considered as macros, which are expanded in Python, not in  $\LaTeX$  itself. Incidentally, `pdfscript` implements the `\let\def=\undefined` idea of Jonathan Fine [1].

### 3.3 Python active strings

There is still an inconvenience. What's easy in  $\LaTeX$ :  
`... before ... \LaTeX{} ... after ...`

is being written in Python as:

```
sect.para('... before ... ',
          cmd('LaTeX'),
          ' ... after ... ')
```

For one time use, it is ok. But when we have several  $\LaTeX$ s in a paragraph, the code looks spaghettiish, syntactically overcomplicated. A simple solution is to let the computer do the low-level job. We can write a function `subst_latex`, which finds the entries of the substring `LaTeX` in the source text and substitutes them with the corresponding commands. With help of this function, the code can be simplified:

```
sect.para(subst_latex(
'... before ... LaTeX ... after ... '))
```

Nearly all the paragraphs of “Essential  $\LaTeX$ ” contain the logo, therefore it is logical to redefine the function `para`, asking it to call `subst_latex` automatically. The final code is:

```
sect.para(
'... before ... LaTeX ... after ... ')
```

In this code, the string `LaTeX` can be considered as an active string (by analogue to the active characters), expanded in Python.

At this point, the code does not use `pdfscript` at all. Instead, it communicates only with the `esla` package, which encapsulates not only the formatting details, but also the details of PDF generation.

## 4 API reference

The previous sections have given enough examples to demonstrate how to use the `pdfscript` library. Here is a more formal description.

For brevity, instead of fully qualified names like `pdfscript.something` I use simple `something`.

### 4.1 Module functions

`TeXsubdoc newdoc(arg1, arg2, ..., argN)`

Creates a new in-memory document. If there are any arguments (of type `string` or `TeXsubdoc`), they are added to the document. The parts of the documents are constructed with the following functions:

`TeXsubdoc cmd(name, arg1, arg2, ..., argN)`

`TeXsubdoc env(name, arg1, arg2, ..., argN)`

`TeXsubdoc opt (arg1, arg2, ..., argN)`

`TeXsubdoc parm (arg1, arg2, ..., argN)`

`TeXsubdoc group (arg1, arg2, ..., argN)`

`TeXsubdoc text (arg1, arg2, ..., argN)`

`TeXsubdoc verbatim (arg1, arg2, ..., argN)`

These functions create the corresponding elements in  $\TeX$ ML. The library does not validate whether a combination of functions makes sense. Notes on the functions:

- `cmd` and `env` require at least one argument (of type `string`), which is the name.
- String arguments of `cmd` are wrapped with implicit calls of `parm`.

- The functions `text` and `verbatim` create an element `TeXML`. The latter function additionally sets the element's attributes in such a way that the text is passed to `TeX` without any changes.

#### 4.2 Methods of `TeXsubdoc`

```
xml.dom.minidom get_root(self)
```

Returns an XML subtree associated with the object `self` of type `TeXMLsubdoc`.

```
TeXMLsubdoc add(self, arg1, ..., argN)
```

Adds subdocuments `argX` of either type `string` or `TeXMLsubdoc` into the subdocument `self`. Returns the reference to the last added subdocument (`argN`, possibly cast to the type `TeXMLsubdoc`).

```
TeXMLsubdoc cmd (self, arg1, ..., argN)
```

```
TeXMLsubdoc env (self, arg1, ..., argN)
```

```
TeXMLsubdoc opt (self, arg1, ..., argN)
```

```
TeXMLsubdoc parm (self, arg1, ..., argN)
```

```
TeXMLsubdoc group(self, arg1, ..., argN)
```

```
TeXMLsubdoc text (self, arg1, ..., argN)
```

```
TeXMLsubdoc verbatim(self, arg1, ..., argN)
```

The first method is a shortcut for:

```
self.add(cmd(arg1, ..., argN))
```

In this definition, the object method `cmd` uses the module function `cmd` to create a document fragment, and then calls the object method `add` to attach the fragment. The other shortcuts are defined in the same way.

#### 4.3 Aliases

Some commands and environments can be accessed via aliases: `name('...')` instead of `cmd('name', ...)` or `env('name', ...)`. Such aliases are created by the following module functions:

```
register_cmd(name)
```

```
register_env(name)
```

#### 5 Conclusion and further work

Despite having no experience yet of `pdfscript` use in a production environment, the experiments so far already allow us to speculate how this tool affects different groups: `TeX` users, `TeX`-related developers and the world-outside-`TeX`.

`TeX` users can safely ignore `pdfscript`. It is dubious to stop typesetting in `TeX` and start doing it in Python. As demonstrated by the “Essential `LATEX`” example, such Python-`TeX` code is rather unreadable.

I expect that developers writing something-to-`LATEX` converters will find `pdfscript` useful. The library allows one to concentrate on the main point of the program and not worry about generating correct

`TeX` syntax. Further, representing a future `TeX` document in a tree simplifies adding refinements, such as changing penalties in the last paragraph of a section.

If `TeX` could be used as a library, what would its API look like? The `pdfscript` approach is a possibility. First, it is enough. Second, optimizations are possible. Commands could be converted to tokens directly, without serializing first to a string and then parsing this string in `TeX`. Similar, text content could immediately become `TeX` characters, without first escaping and then unescaping.

For me, the most important part, however, is how `pdfscript` affects the non-`TeX` world. In the final version of the “Essential `LATEX`” example, we saw that sections, paragraphs, inline markup and other document elements are represented as objects with properties and methods. This approach fits perfectly with modern programming practice, and therefore I hope that `pdfscript` will become a viable alternative to XSL-FO and other PDF creation tools. And when one uses `pdfscript`, one is actually using `TeX`.

The next step of the work is to move from the prototype to a first production version. In particular, I plan to make a PHP version of `pdfscript`, develop a few PHP stylesheets (document templates) and collect users' feedback to decide on the priorities of further development.

#### References

- [1] Jonathan Fine. `TeX` forever! In *EuroTeX 2005 (Pont-à-Mousson) Proceedings*, pages 140–149, 2006. <http://tug.org/TUGboat/Articles/tb27-0/fine.pdf>.
- [2] Alex Papadimoulis. The Daily WTF: Curious Perversions in Information Technology. <http://thedailywtf.com/>.
- [3] Oleg Parashchenko. API to generate `TeX` files, search for related work. `comp.text.tex`, [http://groups.google.com/group/comp.text.tex/browse\\_thread/thread/ba29ef069a47f00a/](http://groups.google.com/group/comp.text.tex/browse_thread/thread/ba29ef069a47f00a/).
- [4] Oleg Parashchenko. `TeXML`: Resurrecting `TeX` in the XML world. *TUGboat*, 28(1):5–10, March 2007. <http://tug.org/TUGboat/28-1/tb88parashchenko.pdf>.
- [5] Jon Warbrick. Essential `LATEX`. <http://mirror.ctan.org/info/latex-essential/>.

◇ Oleg Parashchenko  
bitplant.de GmbH  
Fabrikstr. 15  
89520 Heidenheim, Germany  
olpa (at) uucode dot com  
<http://uucode.com/>

## illumino: An XML document production system with a $\TeX$ core

Matteo Centonza and Vito Piserchia

### Abstract

XML is the state of the art in publishing technology. Publishers, through the “one source, multiple output” paradigm, are able to publish the same content to multiple media without much effort. In this paper we’ll investigate current scenarios for publishers adopting a  $\LaTeX$  workflow and introduce *illumino*, our fulltext XML production system built around  $\TeX$ .

### 1 Introduction

XML publishing in scholarly publications is nothing new. Publishers, through content/format separation, can leverage the many benefits of XML:

- Publish the same content to multiple media
- Store production data in a neutral format, the “*lingua franca*” of Internet applications
- Use XML as a neutral format for long-term archival of content
- Disseminate content through syndication
- Have content ready for data harvesting/mining (discussed in sect. 4.3)

With the term “XML publishing”, we are referring to procedures and methods generating final output media from XML sources. XML sources are authored to produce final output, ready to be published. On the other hand, XML publishing is a complex task since content should be structured to be valid XML, *i.e.*:

- Encoded with correct metadata granularity
- Follow an XML grammar

XML publishing tools are often complex content management systems (CMS). Users need to perform content authoring according to tool specifications. Import tools may be provided, but imported content needs to be reviewed. This is a time-consuming task.

Publishers interested in XML publishing and adopting a  $\LaTeX$  based workflow, are either supposed to develop complex in-house solutions or outsource most of the publishing chain. There are many outsource facilities more or less ready to do the job but the price to pay is losing control of the work.

In this paper we’d like to present *illumino*, our fulltext XML production system that is trying to change this scenario. We’ll present the ideas behind this technology, system capabilities and discuss future development.

### 1.1 illumino

*illumino* is a fulltext XML production system, built around  $(\LaTeX)\TeX$ , which integrates international standards such as:

- DocBook 5.0
- MathML 2.0
- SVG Tiny 1.2
- Unicode 5.0

*illumino* converts  $\LaTeX$  sources to its internal XML format (DocBook) and the publishing chain, starts from XML sources.

The process is similar to the one described in the seminal article by E. Gurari and S. Rahtz [3] but uses different XML technologies. For a graphical representation of the full process, please see figure 1.

*illumino* is a multiplatform application built around  $\TeX$  ( $\TeX$  Live and the embedded  $\TeX$ 4ht), XSLT 2.0, Java, `git` (as SCM) and once configured, has native support for publisher  $\LaTeX$  classes and generates publishers’ native production files as output. It is able to run unmodified in the old  $\LaTeX$  workflow.

*illumino* aims to integrate as smoothly as possible with any  $\LaTeX$  workflow, minimizing production changes to obtain fulltext XML publishing.

To achieve this goal, *illumino* performs automatic metadata enrichment through heuristic methods to match content granularity needed by a given XML grammar. In order to guarantee content safety while heuristically enriching unstructured information, *illumino* has been designed to produce output that perfectly matches that of the  $\LaTeX$  production source file the system is processing: we test for equal checksums of source and production output (currently PostScript output) to ensure this. When this perfect match (“*equivalence*”) applies we are sure that the system has not introduced any modification to document content, so there’s no need to review the article content.

*illumino* has embedded content checking (via SHA checksums) and the user is warned when the system outcome is not the perfect equivalence; in those cases, *illumino* is able to visually highlight differences found, so that visual validation can take place.

*illumino* is an incremental (à la Apache `ant`), client/server application and is able to run through the network with speed similar to that of a conventional  $\LaTeX$  workflow. By integrating SCM technologies, *illumino* can be used concurrently in a safe way. The complete list of features is given on the main *illumino* web page.

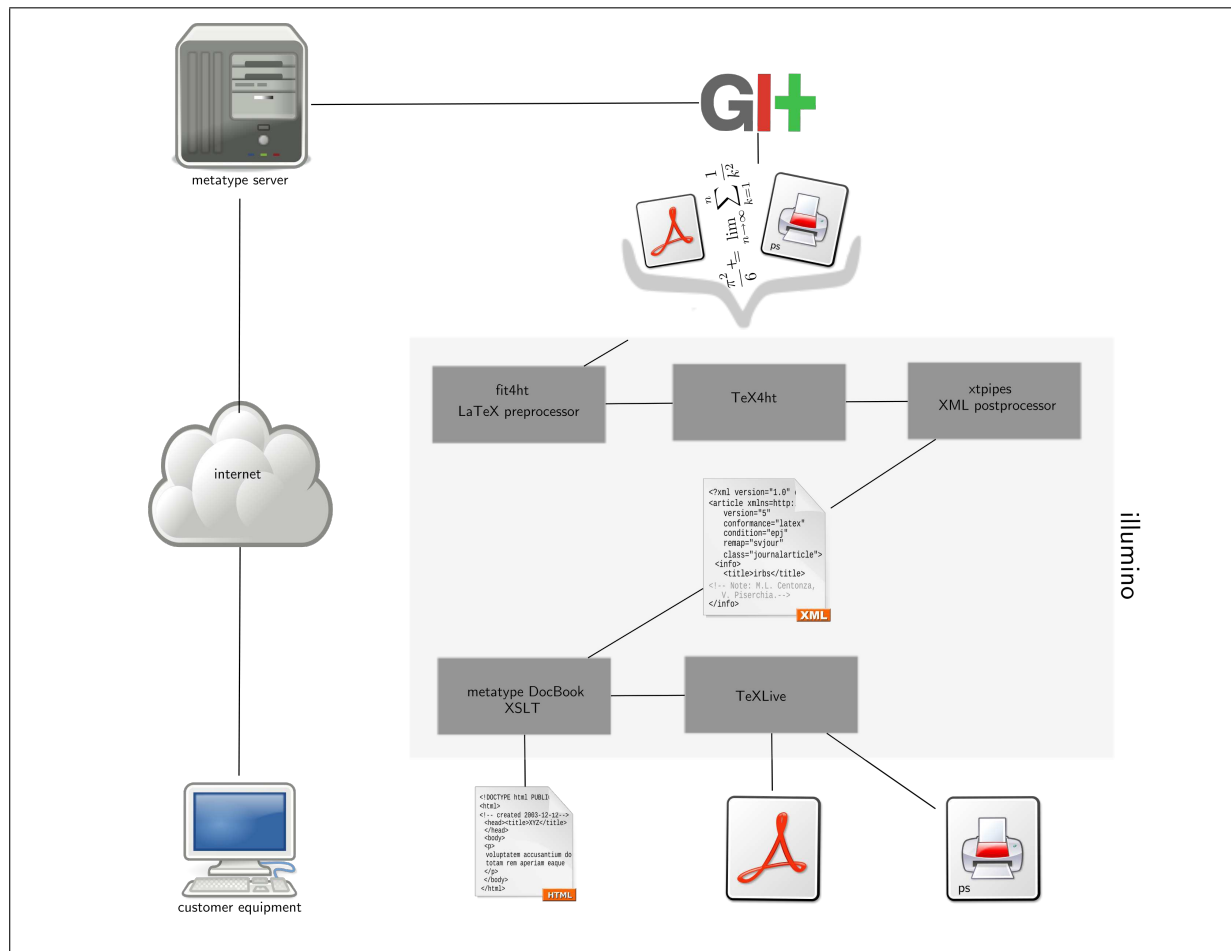


Figure 1: The illumino architecture

## 2 illumino architecture

Figure 1 shows current illumino client/server interaction. illumino uses standard components and implements standard and open protocols.

illumino has its foundations on just two main components:  $\text{\TeX}$  Live and Java.

From a technical point of view, illumino is based on Apache `ant` and is implemented as several custom `ant` tasks, through our `illuminant` library (`antlib`). By using `ant`, illumino is an incremental (through dependencies and timestamps calculations) and multithreaded application (Java).

The system is completely standalone<sup>1</sup> and `ant`, used also to build the whole stack, is able to update and rebuild all upstream dependencies.

What follows is a description of high-level processes of which illumino is made.

<sup>1</sup> With the exception of the Apache Tomcat servlet container (used to implement the caching XSLT engine) and `git` SCM program.

### 2.1 fit4ht

This part of illumino, as its name may suggest, is responsible for making the initial  $\text{\LaTeX}$  source file “fit” to be run under  $\text{\TeX}$ 4ht. This workflow segment parses  $\text{\LaTeX}$  document and by using heuristic algorithms performs:

- Automatic document cleanup (*e.g.* standardize misused  $\text{\TeX}$  primitives and sloppy constructs to  $\text{\LaTeX}$ )
- Enrich document metadata structure (split and tag content according to information patterns)
- Make some constructs ready to be correctly interpreted by  $\text{\TeX}$ 4ht

From a low-level point of view, `fit4ht` is implemented as an `ant` filterreader.

### 2.2 $\text{\TeX}$ 4ht

$\text{\TeX}$ 4ht is the heart of illumino and is the component taking care of  $\text{\LaTeX}$  to XML transformations.

We'll not delve into  $\text{T}\text{E}\text{X}4\text{ht}$  internals since this is out of scope for this article. For a more in-depth explanation of how  $\text{T}\text{E}\text{X}4\text{ht}$  works, the reader may refer to [2, 1].

$\text{T}\text{E}\text{X}4\text{ht}$ 's most notable difference with other similar technologies is the use of the real thing, the  $\text{T}\text{E}\text{X}$  parser, when converting a  $\text{T}\text{E}\text{X}$  file to another format.

For simplicity, we'll condense the  $\text{T}\text{E}\text{X}4\text{ht}$  workflow to three main steps:

1. Seed configurable (at the control sequence level) hooks in DVI output
2. Harvest the seeded hooks to generate a given markup representation
3. Post-process the outcome to undergo validation

We have heavily customized  $\text{T}\text{E}\text{X}4\text{ht}$ <sup>2</sup> mainly to:

- Implement a native backend for DocBook 5.0 output.
- Add support in the  $\text{T}\text{E}\text{X}4\text{ht}$  core for editorial fine tuning control sequences (*e.g.* supporting all tuning `toks`, vertical, horizontal, and math spaces, ...) as XML processing instructions.
- Enrich control sequence mapping in order to go from  $\text{L}\text{A}\text{T}\text{E}\text{X}\rightarrow\text{XML}$  and back without degradation in information quality.

By pre-processing input files and slightly modifying some  $\text{T}\text{E}\text{X}4\text{ht}$  internals, we have made the  $\text{L}\text{A}\text{T}\text{E}\text{X}\rightarrow\text{XML}$  conversion a completely automated process.

We have developed custom “ $(\text{L})\text{T}\text{E}\text{X}4\text{ht}$  compile” `ant` tasks to have automated compilation of sources. Compile reruns are handled automatically (*e.g.*  $\text{T}\text{E}\text{X}4\text{ht}$ , for complex tables have to run several times, and  $\text{L}\text{A}\text{T}\text{E}\text{X}$  needs to be rerun when labels are modified).

Through  $\text{T}\text{E}\text{X}4\text{ht}$ 's power and flexibility we've been able to have fine-grained content resolution and exactly remap a  $\text{L}\text{A}\text{T}\text{E}\text{X}$  file into its corresponding DocBook 5.0 counterpart, producing the same output (we call it “*equivalence*” and their outputs have identical checksums).

`illumino` testcases are made of “*equivalences*” with research papers in physics from different scholarly publications. This approximately 400 pages and 30 articles test suite is `illumino`'s internal certification system and is used to avoid regressions and to spot inconsistencies in the whole `illumino` application stack (including upstream dependencies). For every build, `illumino` must pass these test cases that are constantly updated as soon as we implement new features or fix bugs.

At present, `illumino` has been tested on approximately 4k pages of content from hard sciences.

<sup>2</sup> Thanks to the invaluable help of Eitan Gurari.

## 2.3 XML transformation phase

`illumino` uses XSLT to transform the raw XML document generated by the previous phase ( $\text{T}\text{E}\text{X}4\text{ht}$ ).

In more detail, `illumino`'s XML transformation phase is currently using XSLT 2.0 and takes advantages of its features, *e.g.* by using `xsl:function`, `xsl:character-map`, regular expressions and pattern matching features extensively.

The XSLT 2.0 phase must be seen as a multi-stage stack of stylesheets, where every filter accomplishes a different task.

XSLT stylesheets are organized in two main sets:

- `xtpipes`, an XSLT pre-phase, which takes care of space rearrangement and element positioning, and produces an enriched and valid DocBook document;
- Metatype DocBook XSLT, transforming the resulting DocBook document to all supported formats (including  $\text{L}\text{A}\text{T}\text{E}\text{X}$  with publisher's class).

### 2.3.1 `xtpipes` stylesheets

In this stage, the filter performs:

- Space rearrangements
- Element reordering and structure enrichment
- Validation fixes

Space rearrangements are strictly related to the design decision of aiming for full equivalence with source output.  $\text{L}\text{A}\text{T}\text{E}\text{X}$  and XML spaces obey completely different sets of rules in determining the output. In  $\text{L}\text{A}\text{T}\text{E}\text{X}$  spaces can appear almost anywhere in the source document but may be relevant to output in only some cases; conversely, an XML grammar strictly controls the allowed spaces in the document tree.

In order to achieve “*equivalence*” between source and production output, we have handled all corner situations in which the meaning of spaces from  $\text{L}\text{A}\text{T}\text{E}\text{X}$  and XML differ.

Regarding element reordering and enriching, we have to face the different nature of semi-structured and structured data. For example, in  $\text{L}\text{A}\text{T}\text{E}\text{X}$  documents, many commands can change the properties of the entire group or environment when specified inside that group. Almost all the alignment commands have this behaviour (*e.g.* `\centering` inside a floating environment). On the other hand, on the XML side we have to specify this behaviour with the tag that represents the  $\text{L}\text{A}\text{T}\text{E}\text{X}$  environment, with permitted attributes, if any (*i.e.* `align="center"` inside the `CALS` table element).

Keeping in mind that seeding of  $\text{T}\text{E}\text{X}4\text{ht}$  hooks is sequential and happens when  $\text{T}\text{E}\text{X}$  sees the commands, we have two possibilities:

- using elements and attributes suggested by the XML schema, when meaningful and close to  $\LaTeX$  counterparts (*e.g.* alignment in table environments)
- using a powerful transclusion and linking technique

`xtpipes` stylesheets follows the first approach where possible and in the remaining cases reverts to using a built-in `xlink/xpointer` processor, implemented with XSLT function extensions.

For example, the `xpointer` scheme can be used to link other elements in the document and the `xinclude` syntax can be used to transclude from other documents.

We have been able, with our XSLT 2.0 `xpointer` implementation, to point to any other element in the document and *e.g.* change attribute values. In short, we have XSLT transformations driven by the XML content, so in the final analysis governed by  $\TeX$ 4ht.

When the latter method is not applicable, we resort to bare XML processing instructions to render the construct.

Validation techniques are discussed in sect. 4.1.

### 2.3.2 metatype DocBook stylesheets

This phase produces the supported output formats, starting from valid DocBook 5.0 sources. Leveraging XML's strengths, we can generate several output documents (*e.g.* simple text, HTML,  $\LaTeX$  or documents in other XML markup languages) from the same XML source.

## 2.4 DocBook version 5

DocBook, developed by the OASIS consortium, is a semantic markup language for technical documentation. As a semantic language, DocBook is focused on content and meaning (DocBook has not been designed to visually format content).

DocBook offers several advantages over competing markup languages:

- Long history and schema stability
- Wide adoption and great availability of tools that support authoring of DocBook documents
- Capacity to generate output files in a wide variety of formats (HTML, XSL-FO and  $\LaTeX$  for later conversion into PDF or other document markup languages), lately `epub`
- Semantic similarities with  $\LaTeX$  commands
- Modular structure including widely adopted XML grammars (*e.g.* MathML and SVG)

For a more in-depth explanation of DocBook concepts, the reader may refer to [5].

## 2.5 illumino-remote

`illumino` is a client/server application built upon open protocols. `illumino` leverages SCM technologies, and the backend system exposes `git` (<http://git-scm.com/>) interfaces.

`illumino-remote`, the system client, interacts with the remote `illumino` server through the `git` protocol.

Whenever the `git` daemon receives new changesets (deltas) for a given article from a client, a new local (server) workflow run will be launched on the updated sources and results (*e.g.* XML, PDF deltas) will be sent back to the client.

Normally `git` roundtrips are very fast<sup>3</sup> in comparison to other SCM technologies and we are able, in combination with `ant` behind the scenes, to have `illumino` processing time be on the same order as a  $\LaTeX$  workflow run.

`illumino-remote` is a Java application with JMS message passing between client and server. We are waiting for the pure Java `git` (`jgit`) implementation to mature, in order to have a pure Java client.

`illumino-remote` can control all remote backend behaviour such as:

- Repository operations (add, delete article resources)
- Enable/disable output formats
- Choose the PDF output engine (`pdfTeX`, Adobe Distiller, `ghostscript`)
- Show output differences<sup>4</sup>
- Enforce output equivalence<sup>5</sup>
- Choose a secondary XML output format

## 3 Usage caveats

`illumino` has been designed to integrate as smoothly as possible into any existing  $\LaTeX$  workflow.

XML publishing, starting from unmodified  $\LaTeX$  production sources, while a cost-effective way for publisher to enable a full text XML workflow, is also a complex software task. Aspects of this complexity are:

- Automatic enrichment of semi-structured content to a more structured form
- Proper separation of content from presentational elements.

What follows is a list of production caveats.

<sup>3</sup> Deltas (differences) for storing changesets and fast merging/indexing algorithms let `git` compete with some native filesystem operations.

<sup>4</sup> Visual differences are presented when the transformation does not end with output equivalence.

<sup>5</sup> `illumino` will fail the transformation if the result is not equivalence.



### 3.1 Automated content tagging

Often  $\LaTeX$  sources are not sufficiently structured to permit a 1:1 mapping with the majority of XML schemata. To be able to fill all the data structures provided by an XML schema, we have to properly resolve pieces of information adhering to specific patterns. These patterns are able to take care of most of the production scenarios we have seen during the heavy test phase our product has undergone.

Out of the box, `illumino` is able to resolve correctly and to split various sparse information that in other semi-automatic systems users tag manually.

This process is by no means perfect since it is completely heuristic. In some corner cases, this approach may not be completely satisfactory and manual tagging is needed. If a new content pattern is found or highlighted, it will be added to existing filters.

In other cases, heuristic treatment is simply ineffective (such as affiliation splitting) and users must manually tag content to get the needed granularity (*e.g.* split into organization name, division, etc.).

Our long-term aim is to integrate `illumino` with the UIMA framework and leverage Bayesian annotators to automatically split what currently is done manually (see sect. 4.2).

### 3.2 Content/presentation separation

$\LaTeX$  has a plethora of commands, environments and class infrastructures which allow for a very high fraction of content separated from presentation.

Authors strictly adhering to  $\LaTeX$  and class instructions will provide a very good source base to transform to XML. Unfortunately this is not always true, and non-standard environments, low level  $\TeX$  code instead of standard  $\LaTeX$ ,  $\TeX$  font primitives, etc., are easily found.

We have done our best to automatically transform non-standard code to a more conformant form, preserving its original meaning. This again will probably not cover all possible cases. In a few cases, users should manually convert the non-standard code.

### 3.3 XML validation

A document, to be valid according to an XML grammar, should be checked not only at the structural level but also at the element content level (*i.e.* not only how elements nest but also what elements contain).

This streamlines further processing to other formats and *e.g.* long term archiving of content (one of the most interesting parts of an XML workflow).

This (not surprisingly) comes at a cost: content sometimes should be rearranged in order to adhere to

a given XML schema. The upside is that document overall quality will be increased.

In most situations, `TeX4ht` is able to produce valid XML documents, but some problematic cases exist. In our experiments, we have found at least two classes of problems in which validation should be refined at a later XML post-processing stage.

As already mentioned, this is due to the strict rules imposed on an XML document when compared with the weak structure imposed by the  $\LaTeX$  grammar:  $\LaTeX$  to XML transformation can produce XML chunks that do not fit in the XML structure (*e.g.* elements outside allowed parent).

We have solved these validation problems by using XSL context-aware `xpath` expressions, rearranging the offending chunk and folding it with the most appropriate parent element, whenever the XML schema allows this. With this approach we are able to solve most validation problems. In some remaining cases, users must resort to recoding  $\LaTeX$  sources to solve validation problems; a high fraction of problems come from offending XML chunks generated from a sloppy or invalid use of  $\LaTeX$  constructs.

## 4 “What the future brings”...

### 4.1 XML validation

Currently we validate XML documents through the Namespace-based Validation Dispatching Language (NVDL).

NVDL is able to route content coming from a given namespace in order to be validated by the correct namespace grammar. In this way, we are able (by using DocBook) to intermix content validated through DTD, RelaxNG, and XML Schema.

`oNVDL`, an open-source NVDL implementation based on `Jing`, is our choice.

In the future, we want to explore the opportunity to take advantages of other XML validation languages. In particular our attention and future efforts are focused on the Schematron validation language. By using Schematron rules we will be able to deal more easily with current validation constraints.

### 4.2 Improving unstructured content parsing through the UIMA framework

In section 2.1 we introduced `fit4ht` filters taking care of document metadata structure enrichment, information tagging and code cleanup.

`fit4ht` is a set of specialized modules taking care of enriching information structure by adding context metadata. The nature of `fit4ht` modules is heuristic: whenever document excerpts adhere to a given pattern, information can be split (safely, since “*equivalence*” or visual validation comes to help).

Since one of `illumino`'s tasks is to treat unstructured/partially structured information to convert into a more structured form, in the long term we'll port `fit4ht` modules to Apache UIMA (<http://uima.apache.org/>).

Unstructured Information Management applications are software systems that analyze large volumes of unstructured information in order to discover knowledge relevant to an end user. An example UIM application might ingest plain text and identify entities, such as persons, places, organizations; or relations, such as works-for or located-at.

The UIMA frameworks support configuring and running pipelines of Annotator components. These components do the actual work of analyzing the unstructured information. Users can write their own annotators, or configure and use pre-existing annotators. Some annotators are available as part of the UIMA project; others are contained in various repositories on the Internet.

By integrating `illumino` with the framework we will be able to leverage the software ecosystem built around UIMA and *e.g.* split information based on Bayesian inference or address other editorial tasks such as normalization of inflected forms.

### 4.3 Knowledge mining

Another interesting field for which scientific XML content is particularly suited is *knowledge mining*.

Several advances in computer science have been brought together under the rubric of “data mining” [4]. Techniques range from simple pattern searching to advanced data visualisation and neural networks. Since our aim is to extract comprehensible and communicable scientific knowledge, our approach should be characterised as “knowledge mining”.

Our idea is to create a network of links between research articles from various fields of science and accelerate research, scientific discovery and innovation.

The key point is that scientific papers, especially from the hard sciences, encode most of their content using mathematical expressions. Every mathematical expression has a unique meaning.

By indexing all occurrences of mathematical expressions present in research papers, it would be possible to build a network of links between research articles. Analyzing links between different fields of

knowledge would make it possible to deduce symmetries, patterns, and even similarities that could be used as research targets.

### 4.4 illumino GUI

We plan to develop a graphical interface in order to have a smooth interaction with the system. This graphical interface should integrate a  $\text{\LaTeX}$  editor and will handle remote interaction with the system.

In our plans, this will be done by developing an Eclipse plugin, in order to leverage the Eclipse ecosystem to have advanced functionalities such as:

- Real-time shared editing
- Context sensitive editing
- Seamless remote interaction
- Versioning and change management (à la `git`).

### References

- [1] G. Cevolani. Introduzione a  $\text{\TeX}$ 4ht, Proceedings of the 2004 Italian GuIT meeting (in Italian). <http://www.guit.sssup.it/guitmeeting/2005/articoli/cevolani.pdf>
- [2] M. Goossens and S. Rahtz with E. Gurari, R. Moore, and R. Sutor. *The  $\text{\LaTeX}$  Web Companion*, Addison-Wesley, 1999.
- [3] E. Gurari and S. Rahtz. “From  $\text{\LaTeX}$  to MathML and back with  $\text{\TeX}$ 4ht and Passive $\text{\TeX}$ ”. <http://www.cse.ohio-state.edu/~gurari/docs/mml-00/mml-00.html>
- [4] P. Langley and H.A. Simon. Applications of machine learning and rule induction. *Communications of the Association for Computing Machinery*, 38(11), 54–64, 1995.
- [5] N. Walsh. *DocBook: The Definitive Guide*, O'Reilly & Associates. <http://www.docbook.org/tdg/>

◇ Matteo Centonza  
metatype, Via Santacroce 13/5,  
I-40122 Bologna, Italy  
matteo (at) metatype.it

◇ Vito Piserchia  
metatype, Via Santacroce 13/5,  
I-40122 Bologna, Italy  
vito (at) metatype.it

## Managing printed and online versions of large educational documents

Jean-Michel Hufflen

### Abstract

We have developed a  $\text{\LaTeX} 2_{\epsilon}$  package, `pfa-macros`, usable for both presentational education, concerning ‘classical’ students, and distance education, where most of a curriculum is performed by means of online documents. First, we explain why requirements for educational documents are not the same for these two ways of teaching. Then we show why our package allows us to manage two versions — printed and online — of the same textbook.

**Keywords** Presentational education, distance education, course text, online course, case study,  $\text{\LaTeX}$ , PDF, pdf $\text{\LaTeX}$ , `pfa-macros` package.

### 1 Introduction

The Internet has revived *correspondence education*: now many network tools are widely used within this field: electronic mail, mailing lists, forums, online documents available *via* the Web, etc. The term ‘correspondence education’ seems to be quite old, since it appears to be related to ‘classical’ letters sent and delivered by post, so nowadays the term ‘*distance education*’ is preferred. As result of greater and greater interest in distance education, most universities in the world have increased such offerings. An example of a French academic institution delivering distance education is the CTU,<sup>1</sup> part of the University of Franche-Comté, located at Besançon. The CTU allows students to get all the units required for a master in Computer Science. Of course, the University of Franche-Comté still provides curricula in presentational education — for students who physically attend ‘classical’ lectures, exercises and lab classes — which remains the ‘traditional’ way of teaching. Obviously some teaching units are common to the two curricula of presentational and distance education.

In this article, we show how some new  $\text{\LaTeX}$  commands allow us to manage the different parts of a single document’s body, for presentational students as well as distance ones. In fact, these parts have been initially written as chapters and appendices of a textbook for presentational students. Later, they have been reused and maintained as we explain in Section 2. Then Section 3 goes thoroughly into requirements about educational documents and shows that requirements for textbooks for presentational

students and online documents for distance students are not the same. Our commands have been grouped into a package `pfa-macros`:<sup>2</sup> Section 4 describes the broad outlines of it. Finally, Section 5 discusses some alternative solutions.

A report about this work has already been published as [7], but within a general conference about computer-aided education, so there we reduced technical details about  $\text{\LaTeX}$ ’s features as far as possible. The present article gives a bit more detailed description of our package’s functionalities.<sup>3</sup> However, reading it only requires knowledge of  $\text{\LaTeX}$  as an end user.

### 2 History

One of the teaching units we are in charge of is devoted to *functional programming*.<sup>4</sup> In fact, it is entitled *Advanced Functional Programming*, PFA for short,<sup>5</sup> since it is attended by graduate students — 4th-year university degree in computer science — that is, students who already have experience in programming. The ‘philosophy’ and contents of this unit are described in [6]. Let us just recall briefly that students actually practise only one programming language within this unit — Scheme [20] — but alternative implementations of functional programming concepts are exemplified using other programming languages, such as Common Lisp<sup>6</sup> [21], Standard ML<sup>7</sup> [16], CAML<sup>8</sup> [12], and Haskell<sup>9</sup> [17]. Other comparisons with modern object-oriented languages such as Java [9], C++ [22], and C# [13] are also given. In addition, we show in [6] that some examples are demonstrated using  $\text{\TeX}$ ’s language. As a consequence, a textbook based on what is taught within this unit should include many excerpts of programs using various languages. Setting up this teaching unit PFA began in spring 1997 and the first version of our printed document [4] came out in August 1997, with a pre-version of a short additional document [5] devoted to an introduction to the  $\lambda$ -calculus [1], the common root of functional programming languages.

<sup>2</sup> Available online: <http://lifc.univ-fcomte.fr/home/~jmhufflen/latex-etc/pfa-macros.sty>.

<sup>3</sup> An extended version [8], more technical, is given in the proceedings of the 2010 conference of the  $\text{\TeX}$  (*Gruppo Utilizzatori Italiani di  $\text{\TeX}$* ), the Italian-speaking users group.

<sup>4</sup> *Functional programming* emphasises application of functions, whereas *imperative programming* — the paradigm implemented within more ‘traditional’ languages such as Pascal [25] or C [11] — emphasises changes in state.

<sup>5</sup> *Programmation Fonctionnelle Avancée*, in French. Our package’s name — `pfa-macros` — originates from this acronym.

<sup>6</sup> ‘Lisp’ stands for **L**IS**T** **P**ROCESSOR.

<sup>7</sup> ‘ML’ stands for **M**ETA**L**ANGUAGE.

<sup>8</sup> **C**ATEGORICAL **A**BSTRACT **M**ACHINE **L**ANGUAGE.

<sup>9</sup> Named after Haskell Brooks Curry (1900–1982).

<sup>1</sup> *Centre de Télé-enseignement Universitaire*, that is, *University Centre for Teleteaching*.

When the master's for distance students was launched, for the academic year 2004–2005, its curriculum obviously resembled the master's in presentational education. But a unit common to these two curricula was not necessarily handled by the same teacher. Concerning us, we have been in charge of the PFA unit within both presentational and distance education, but this arrangement does not hold true for all the units. So we were interested in a method that would allow us to derive the two versions — printed and online — from the same source files. Such a *modus operandi* would ease the maintenance of our documents. For example, some slight mistakes should be fixed once, and we wished to add more examples. More ambitiously, the version of standard Scheme changed, from [2] to [10], so we ought to adapt some existing texts and examples.<sup>10</sup>

### 3 Different requirements

The document [4] consists of six chapters. Each chapter includes exercises, given with model solutions. These chapters are followed by several appendices — making precise some extra information or devoted to lab class exercises done by students — and a rich ‘Bibliography’ section. The whole document is approximately 400 pages long. It can be viewed as a textbook, even if its dissemination is limited to this unit's students. The students are progressively given the successive parts of this document, but it is organised as a whole, with precise architecture: cross-references are widely used throughout it. Of course, it contains not only text — in the sense of successive paragraphs — but also many examples of programs and some mathematical formulas, even if it is not really a textbook in mathematics.

#### 3.1 Requirements about typography

When the first teaching units were launched in distance education, teachers were obviously asked to install online documents on the Web. Some teachers wrote documents using HTML.<sup>11</sup> However, such a choice seemed to us unsuitable for scientific documents: the look of resulting Web pages depends on the browser used; in addition, formatting mathematical formulas and program fragments often results in poor-quality output. We could have used some converters from L<sup>A</sup>T<sub>E</sub>X source texts to HTML pages,<sup>12</sup> which may use *images* to insert fragments whose conversion to HTML is difficult, e.g., mathematical formulas. However, even if these converters allow the

output's quality to be improved, in comparison with direct writing in HTML, authors have to adapt source texts in order for the conversion to work properly. In other words, it may be difficult to do such a task for a large document already written and formatted.

Concerning the insertion of program fragments, let us recall that this point was essential, especially about the fragments given in languages other than Scheme. We could perform some demonstrations during the lab classes of presentational students, so they could observe these other programs' behaviour. The same *modus operandi* was impossible for distant students, and it was difficult to ask them to install many compilers or interpreters. So the solution was to ask them for exercises only in Scheme — as done for presentational students — but the examples given throughout our text must be explicit, in order for these students to understand without running them. In addition, we paid much attention to the indentation of these programs and inserted some comments throughout them using special effects — e.g., slanted fonts — so they do not use *verbatim*-like environments, but are built by means of *tabbing* environments.

From our point of view, only PDF<sup>13</sup> [3, Ch. 2] offers some sufficient warranty about the quality of texts displayed on the Web. This point is also related to *communication*: when a teacher writes some formulas onto a blackboard, students see the result exactly as the teacher formats it. The same warranty is given by PDF files, not by HTML pages. So we decided to systematically use PDF files, generated by the pdfL<sup>A</sup>T<sub>E</sub>X program [3, § 2.4]. In addition, if the *hyperref* package [3, § 2.3] is used, PDF files produced by pdfL<sup>A</sup>T<sub>E</sub>X can support hyperlinks, as in HTML. Let us now come to the organisational differences between texts for presentational and distance students.

#### 3.2 Presentational vs. distance education

Of course, distance students could not be given a single document as a huge PDF file. It is preferable for distance students to get separate medium-sized files, according to the steps of their planning. Besides, let us not forget that these files are downloaded: students cannot be asked to download a huge file again if only some typing mistakes have just been fixed. Splitting this big document into separate files induces a precise organisation of cross-reference links throughout the original version. Information redundancy should be avoided, so all the parts should point to the same ‘Bibliography’ section, as a separate file.

<sup>10</sup> Later, in 2007, another change occurs, from [10] to [19].

<sup>11</sup> **H**yper**T**ext **M**arkup **L**anguage. A good introduction to it is [15].

<sup>12</sup> Some are described in [3, Ch. 3–4].

<sup>13</sup> **P**ortable **D**ocument **F**ormat.

Model solutions can be given after each exercise for presentational students, especially if this exercise has already been proposed in class. That cannot be the same for a document devoted to distance education: model solutions should be grouped at the end of each chapter, or provided in separate files.

#### 4 The pfa-macros package

Let us assume that the chapters, sections, etc. of the two versions — printed and online — are numbered identically. Besides, L<sup>A</sup>T<sub>E</sub>X allows each chapter of a document to be associated with its own auxiliary (.aux) file, containing information solving cross-references. So we can compile a chapter for the online version by using the auxiliary files of the document's other chapters of the 'presentational' version. A cross-reference written by L<sup>A</sup>T<sub>E</sub>X's `\ref` command is implemented in pdfL<sup>A</sup>T<sub>E</sub>X as an internal hyperlink, which is fine for cross-references within the same chapter. For external references, we define a new command:

```
\pfaexternalref[chapter-file]{label0}
```

If the big document for presentational education is generated, this works like `\ref{label0}`. If the chapter is generated as part of the online text, a link to the file `chapter-file.pdf` is put. In both cases, the same text is displayed or enlightened by a hyperlink. If the complete version has already been put on the site, it can be searched. Otherwise, it is a kind of *stub* whose contents reads 'This chapter will be put later' and when the complete version is put, the hyperlink will remain the same.<sup>14</sup> We use similar technique for cross-references to footnotes belonging to another chapter (commands `\pfacite`, `\pfaexternalref`, `\pfaexternalfootnoteref`, etc.).

#### 5 Other methods

There exists some work allowing a L<sup>A</sup>T<sub>E</sub>X document to refer to a label belonging to an external document. A first example is given by some commands of the `html` package [3, § 3.5.3], unsuitable for us, since this package is only interesting if you want to derive HTML pages. A second implementation of external references using hyperlinks is given by the `xr-hyper` package [14, § 2.4.6]. Nevertheless, this package has two drawbacks for us. First, it does not deal with bibliographical citations (`\cite` commands). Second, it cannot refer to an external label that will be defined

<sup>14</sup> Of course, when we started this task, such a choice led us to look for all the occurrences of the `\ref` command and change some into `\pfaexternalref` ones. In practice, that was not difficult, because a good technique is to prefix labels' name by an identifier for the corresponding chapter. So the file name to be put was not difficult to supply.

later. To explain that, let us consider that the first chapter refers to a section of the second chapter. As long as the second chapter is replaced by a stub, the hyperlink will fail; it will work only as soon as this chapter's complete text is made public.<sup>15</sup> Within our system, the hyperlink always points to the second chapter's PDF file, a stub or the complete text.<sup>16</sup>

If we had started from scratch, that is, if both the presentational and distance unit were launched at the same time, an interesting method could have been to specify our input files using XML,<sup>17</sup> and XSLT<sup>18</sup> [24] could have been used to derive texts for L<sup>A</sup>T<sub>E</sub>X, or in XSL-FO<sup>19</sup> [23]

#### 6 Conclusion

A first sketch of the present article was initially designed for the EuroT<sub>E</sub>X 2010 conference. The Web page announcing this event mentioned that L<sup>A</sup>T<sub>E</sub>X is still widely used, but 'the landscape is changing', and other word processors continue to emerge.

From our point of view, the present work shows that L<sup>A</sup>T<sub>E</sub>X is still unrivalled to 'intelligently' process texts for several purposes. As mentioned above, the first version of our course text came out in 1997. Then it has evolved deeply — chapters and appendices have been wholly revised — and continuously, since we have applied some changes each year. We did it successfully — in particular when we had to be conformant with new revisions of standard Scheme — so we can think that our system is reliable.

#### 7 Acknowledgements

I am grateful to the distance education students who addressed me very constructive criticisms; year after year, they indirectly helped me improve my tools. Thanks to Karl Berry and Barbara Beeton, who kindly proofread this article.

◇ Jean-Michel Hufflen  
LIFC (EA CNRS 6942)  
University of Franche-Comté  
16, route de Gray  
25030 Besançon Cedex  
France  
jmhufflen (at) lifc dot univ-fcomte dot fr  
<http://lifc.univ-fcomte.fr/home/~jmhufflen>

<sup>15</sup> From a pedagogical point of view, such a *forward reference* is often viewed as bad. But it can occur within a footnote, or a fragment that can be skipped at first reading.

<sup>16</sup> That could be improved in a future version: if the external label exists, the hyperlink directly points to the corresponding resource, if not, it points to a stub.

<sup>17</sup> eXtensible Markup Language. [18] is a good introduction to this meta-language.

<sup>18</sup> eXtensible Stylesheet Language Transformations.

<sup>19</sup> eXtensible Stylesheet Language — Formatting Objects.

## References

- [1] Alonzo CHURCH: *The Calculi of Lambda-Conversion*. Princeton University Press. 1941.
- [2] William D. CLINGER and Jonathan A. REES, with Harold ABELSON, Norman I. ADAMS IV, David H. BARTLEY, Gary BROOKS, R. Kent DYBVIK, Daniel P. FRIEDMAN, Robert HALSTEAD, Chris HANSON, Christopher T. HAYNES, Eugene Edmund KOHLBECKER, JR., Donald OXLEY, Kent M. PITMAN, Guillermo Juan ROZAS, Guy Lewis STEELE, JR., Gerald Jay SUSSMAN and Mitchell WAND: “Revised Report<sup>4</sup> on the Algorithmic Language Scheme”. *ACM Lisp Pointers*, Vol. 4, no. 3. July 1991.
- [3] Michel GOOSSENS and Sebastian RAHTZ, with Eitan M. GURARI, Ross MOORE and Robert S. SUTOR: *The L<sup>A</sup>T<sub>E</sub>X Web Companion*. Addison-Wesley Longman, Inc., Reading, Massachusetts. May 1999.
- [4] Jean-Michel HUFFLEN : *Programmation fonctionnelle avancée. Notes de cours et exercices*. Polycopié. Besançon. Juillet 1997.
- [5] Jean-Michel HUFFLEN : *Introduction au λ-calcul (version révisée et étendue)*. Polycopié. Besançon. Février 1998.
- [6] Jean-Michel HUFFLEN: “Using T<sub>E</sub>X’s Language within a Course about Functional Programming”. *MAPS*, Vol. 39, pp. 92–98. In EuroT<sub>E</sub>X 2009 conference. August 2009.
- [7] Jean-Michel HUFFLEN: “Recycling Previous Documents for Distance Education”. In: *Proc. CSEDU 2010*, Vol. 1, pp. 469–472. Valencia, Spain. April 2010.
- [8] Jean-Michel HUFFLEN: “Managing Printed and On-Line Versions of Large Educational Documents”. *ArsT<sub>E</sub>Xnica*. To appear. November 2010.
- [9] *Java Technology*. March 2008. <http://java.sun.com>.
- [10] Richard KELSEY, William D. CLINGER, and Jonathan A. REES, with Harold ABELSON, Norman I. ADAMS IV, David H. BARTLEY, Gary BROOKS, R. Kent DYBVIK, Daniel P. FRIEDMAN, Robert HALSTEAD, Chris HANSON, Christopher T. HAYNES, Eugene Edmund KOHLBECKER, JR., Donald OXLEY, Kent M. PITMAN, Guillermo J. ROZAS, Guy Lewis STEELE, JR, Gerald Jay SUSSMAN and Mitchell WAND: “Revised<sup>5</sup> Report on the Algorithmic Language Scheme”. *HOSC*, Vol. 11, no. 1, pp. 7–105. August 1998.
- [11] Brian W. KERNIGHAN and Dennis M. RITCHIE: *The C Programming Language*. 2nd edition. Prentice Hall. 1988.
- [12] Xavier LEROY, Damien DOLIGEZ, Jacques GARRIGUE, Didier RÉMY and Jérôme VOILLON: *The Objective Caml System. Release 3.12 Documentation and User’s Manual*. 2010. <http://caml.inria.fr/pub/docs/manual-ocaml/index.html>.
- [13] MICROSOFT CORPORATION: *Microsoft C# Specifications*. Microsoft Press. 2001.
- [14] Frank MITTELBAACH and Michel GOOSSENS, with Johannes BRAAMS, David CARLISLE, Chris A. ROWLEY, Christine DETIG and Joachim SCHROD: *The L<sup>A</sup>T<sub>E</sub>X Companion*. 2nd edition. Addison-Wesley Publishing Company, Reading, Massachusetts. August 2004.
- [15] Chuck MUSCIANO and Bill KENNEDY: *HTML & XHTML: The Definitive Guide*. 5th edition. O’Reilly & Associates, Inc. August 2002.
- [16] Lawrence C. PAULSON: *ML for the Working Programmer*. 2nd edition. Cambridge University Press. 1996.
- [17] Simon PEYTON JONES, ed.: *Haskell 98 Language and Libraries. The Revised Report*. Cambridge University Press. April 2003.
- [18] Erik T. RAY: *Learning XML*. O’Reilly & Associates, Inc. January 2001.
- [19] Michael SPERBER, William CLINGER, R. Kent DYBVIK, Matthew FLATT and Anton VAN STRAATEN, with Richard KELSEY, Jonathan REES, Robert Bruce FINDLER and Jacob MATTHEWS: *Revised<sup>5,97</sup> Report on the Algorithmic Language Scheme*. June 2007. <http://www.r6rs.org>.
- [20] George SPRINGER and Daniel P. FRIEDMAN: *Scheme and the Art of Programming*. The MIT Press, McGraw-Hill Book Company. 1989.
- [21] Guy Lewis STEELE, JR., with Scott E. FAHLMAN, Richard P. GABRIEL, David A. MOON, Daniel L. WEINREB, Daniel Gureasko BOBROW, Linda G. DEMICHEL, Sonya E. KEENE, Gregor KICZALES, Crispin PERDUE, Kent M. PITMAN, Richard WATERS and Jon L WHITE: *Common Lisp. The Language. Second Edition*. Digital Press, 1990. <http://www.cs.cmu.edu/Groups/AI/html/cltl/cltl2.html>.
- [22] Bjarne STROUSTRUP: *The C++ Programming Language*. 2nd edition. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts. 1991.
- [23] W3C: *Extensible Stylesheet Language (XSL). Version 1.1*. W3C Recommendation. Edited by Anders Berglund. December 2006. <http://www.w3.org/TR/2006/REC-xsl11-20061205/>.
- [24] W3C: *XSL Transformations (XSLT). Version 2.0*. W3C Recommendation. Edited by Michael H. Kay. January 2007. <http://www.w3.org/TR/2007/WD-xslt20-20070123>.
- [25] Niklaus WIRTH: “The Programming Language Pascal”. *Acta Informatica*, Vol. 1, no. 1, pp. 35–63. 1971.

## Aligning text in diagrams exported by Mathematica: A question about the PostScript infrastructure

Michael P. Barnett

### Abstract

I produce many L<sup>A</sup>T<sub>E</sub>X documents that contain diagrams exported by Mathematica\* graphics. Usually, these contain text. Often, this is misaligned horizontally. I think that getting correct alignment needs an understanding of PDF font encoding. This note describes the problem in hope of getting feedback.

### 1 Introduction

This note seeks advice from PostScript experts about certain details of font encoding. I need this information to align built-up text expressions that mix different fonts, in diagrams that are constructed by Mathematica graphics. I include these diagrams in L<sup>A</sup>T<sub>E</sub>X manuscripts on topics in the natural sciences, mathematics and the humanities. The text is aligned by the TMG (text in Mathematica) package that I wrote. Fig. 1 and nearly 30 similar diagrams are in a recent paper on nuclear magnetic resonance (NMR) that I wrote with István Pelczer [1]. The construction

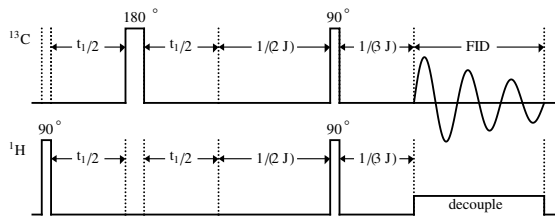


Figure 1: An NMR pulse sequence diagram.

of these diagrams prompted the work on TMG. The package contains an `encode` function that tries to position the contents of separate Mathematica `Text` commands precisely. This is a standard need when a set of related diagrams consists of varied selections of modules that contain text. The definitive description of the `Text` command in *The Mathematica Book* [2] states:

`Text[expr, coords, offset]` specifies an offset for the block of text relative to the coordinates given.”

The description goes on to mention sample offsets that include “{-1, 0} left-hand end at {x, y}” and “{0, -1} centered above {x, y}”. The obvious extension is that {-1, -1} puts the lower left corner of

\* MATHEMATICA is a registered trademark of Wolfram Research Inc.

the text at {x, y}. The description in [2] refers to the “bounding rectangle that surrounds the text”.

The idea of bounding rectangles that surround text has been inherent in the use of moveable type for millennia [3] and, more recently, in phototypesetting [4]. It is associated with the idea of a baseline, defined as “the line upon which most letters ‘sit’ and below which descenders extend” [5]. Fig. 2 shows a sequence of words and isolated characters in serif, sans-serif and Greek fonts, that were typeset by elementary L<sup>A</sup>T<sub>E</sub>X coding. The rectangles that surround the characters and the baseline were drawn by `\rule` commands. Typesetting software has customarily

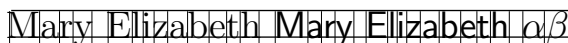


Figure 2: Bounding boxes — consistent baselines.

treated the vertical coordinate of a piece of text as the position of its baseline, since the inception of the field in the late 1950s [4]. Digital fonts were developed that treated each character as if it were contained in a rectangle, that had the point size as its height, with baselines positioned for consistency between characters in the same font and in different fonts. This paralleled the design of metal type slugs.

Fig. 3 shows some bad alignment produced by Mathematica `Text` commands. These all contain the offsets {-1, -1}, and the same y value is used in the `Text` and `Line` commands that produced each row. `Line[{x1, y1}, {x2, y2}]` draws a line from (x<sub>1</sub>, y<sub>1</sub>) to (x<sub>2</sub>, y<sub>2</sub>).

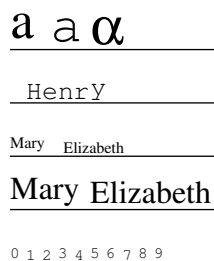


Figure 3: Examples of unexpected misalignment.

```
{Text[Style["a", FontFamily -> "Times-Roman",
  FontSize -> 20], {0, 600}, {-1, -1}],
Text[Style["a", FontFamily -> "Courier",
  FontSize -> 20], {15, 600}, {-1, -1}],
Text[Style["a", FontFamily -> "Symbol",
  FontSize -> 20], {30, 600}, {-1, -1}],
Line[{0, 600}, {80, 600}]}]
(* *)
Text[Style["H", FontFamily -> "Courier",
  FontSize -> 10], {6, 580}, {-1, -1}],
Text[Style["e", FontFamily -> "Courier",
```

```

FontSize -> 10], {12, 580}, {-1, -1}},
...
Text[Style["y", FontFamily -> "Courier",
FontSize -> 10], {30, 580}, {-1, -1}],
Line[{{0, 580}, {80, 580}}],
(* *)
Text[Style["Mary",
FontSize -> 6], {0, 560}, {-1, -1}],
Text[Style["Elizabeth",
FontSize -> 6], {20, 560}, {-1, -1}],
Line[{{0, 560}, {80, 560}}],
(* *)
Text[Style["Mary",
FontSize -> 12], {0, 540}, {-1, -1}],
Text[Style["Elizabeth",
FontSize -> 12], {30, 540}, {-1, -1}],
Line[{{0, 540}, {80, 540}}],
(* *)
Text[Style["0", FontFamily -> "Courier",
FontSize -> 6], {0, 520}, {-1, -1}],
Text[Style["1", FontFamily -> "Courier",
FontSize -> 6], {6, 520}, {-1, -1}],
...
Text[Style["9", FontFamily -> "Courier",
FontSize -> 6], {54, 520}, {-1, -1}],
Line[{{0, 520}, {80, 520}}];

```

An earlier version of these commands is shorter but needs more explanation. There are several reasons for the alignment effects in Fig. 3.

1. In the 1<sup>st</sup> row, the letter “a” in Times-Roman and Courier fonts and the  $\alpha$  do not line up because the different fonts are coded with different baselines in their respective bounding boxes.
2. In the 2<sup>nd</sup> row, the *bases* of the rectangles that fit tightly around the individual letters in “Henry” are aligned. This makes the “y” high relative to the other letters.
3. In the 3<sup>rd</sup> and 4<sup>th</sup> rows, the string “Mary” has consistent baselines. So does “Elizabeth”. But the “y” in “Mary” pushes the entire string up, relative to “Elizabeth”.
4. In the 5<sup>th</sup> row, I think that the digits do not line up because 0, 3, 5, 6, 8 and 9 were coded using one set of conventions, and 1, 2, 4 and 7 using a different set.

Fig. 4 shows a practical consequence of the alignment of personal names. I wrote a Mathematica script in the 1990s to display genealogies. Fig. 4 is a minimalistic display of relationships that dominated British society for half a century. The misalignment

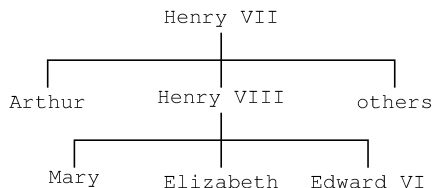


Figure 4: The simplified Tudor succession.

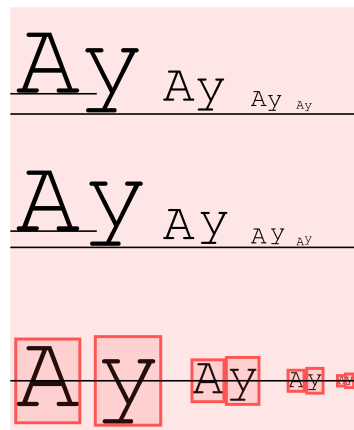


Figure 5: Forced demonstration of bounding box.

would be unacceptable in a scholarly journal that dealt with the substantive issues that this presents.

Some forms of undesirable alignment become more pronounced as font size decreases. This may be related to an apparent drift in the snugness of the bounding rectangle. Although I have not found relevant data directly, some simple syntactic errors make the system display the rectangles that it seems to use. This is done in the 3<sup>rd</sup> row of Fig. 5 by using null as the  $y$  offset. The system treats it as 0. In the 1<sup>st</sup> row, the string “Ay” is set successively in 40, 20, 10 and 5 point Courier type. The  $y$  coordinate in the `Text` expressions is 550, and a `Line` expression draws a line with  $y = 550$  across the page. The serif of the “y” touches this line in 40 point type, but not in the smaller sizes. A line drawn with  $y = 557.6$  shows the elevation of the serifs of the “A” relative to those of the “y” in 40 point type.

In the 2<sup>nd</sup> row, the two letters “A” and “y” are set by separate `Text` statements, with  $y = 500$ . The serif of the 40 point “y” touches a line with this coordinate, but the relative elevation of the “A” has dropped to 6.1 points. As the size decreases, the “y” continues to move up relative to the “A”.

In the 3<sup>rd</sup> row the rectangles surrounding the “y” have moved down slightly, with decreasing point size, relative to the serif. Fig. 6 shows the 5 point example, magnified 8-fold by the `scale` parameter in the  $\text{\LaTeX}$



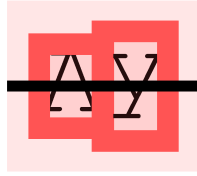


Figure 6: 8-fold magnification of 5 point example.

`\includegraphics` command. The position of the horizontal line at  $y = 450$  emphasizes the change.

## 2 The TMG encode function

I try to achieve the horizontal alignment of a body of text that is displayed on a single line by putting the  $i$ -th character, denoted here by  $c_i$ , into a separate expression of the form

```
Text[Style[ $c_i$ , FontFamily-> $f_i$ , FontSize-> $s_i$ ],
      { $x_0 + h_i$ ,  $\bar{y}$ }, {-1,  $\sigma(f_i, s_i, c_i)$ }]
```

where

$$1. h_i = \sum_{j=1}^{i-1} \frac{s_j}{10} w(f_j, c_j),$$

2.  $c_i$ ,  $f_i$  and  $s_i$  are the  $i$ -th character and the font style and font size in which it is set,
3.  $w(f_i, c_i)$  is the width of  $c_i$  in 10 point font  $f_i$  (giving  $w(\text{"Courier"}, c)$  the value 6 for the entire character set),
4.  $\sigma(f_i, s_i, c_i)$  is the offset that puts the baseline of the character, in the specified size and style, onto the line  $y = \bar{y}$ , that is specified in the coordinates part of the `Text` statement,
5.  $x_0$  is the starting  $x$  coordinate of the text.

I developed methods to find widths and offsets by trial and error. Using these, I found the widths for the Courier, Times-Roman and Symbol fonts with ease and accuracy. I found the offsets for the Courier and Symbol fonts for point sizes 4 to 10, and Times-Roman for size 12, with considerable difficulty and tedium and some uncertainty.

---

### I would like advice on finding the offsets algorithmically from the font tables.

---

The information may be in the chapter on fonts in the *PostScript Language Reference* manual [6]. The learning curve for this seems non-trivial, and I do not want to climb it unnecessarily.

The TMG expression

```
encodeString[font, size, x, y, string]
```

sets *string* in the specified font face and font size, starting with the left edge of the bounding box of the 1<sup>st</sup> character at  $x$ , and the baseline at  $y$ . In the more general expression

```
encodeSequence[item1, item2, ...]
```

a a α

Henry

Mary Elizabeth

Mary Elizabeth

0123456789

Figure 7: Output of encode expressions.

each item is either

1. a character string, *e.g.* “delay”, that is set in uniform font and size on a common baseline;
2. a character sequence with no quote marks, *e.g.* delay, that the system envelops in quote marks and treats as just described (this 2<sup>nd</sup> kind of item actually is restricted to objects that in Mathematica syntax are symbols);
3. one of the following commands
  - (a) `ps[n]`: changes font to size  $n$  without altering the baseline,
  - (b) `tf[f]`: changes the font to style  $f$ ,
  - (c) `tf[f, n]`: changes the font to style  $f$  and size  $n$ ,
  - (d) `sub[s]`: sets the string  $s$  in the decoration size, sunk to subscript level,
  - (e) `sup[s]`: sets  $s$  in decoration size, raised to superscript level,
  - (f) `subSup[s1, s2]`: sets  $s_1$  and  $s_2$  as subscript and superscript, left aligned,
  - (g) `lSubSup[s1, s2]`: sets  $s_1$  and  $s_2$  as right aligned subscript and superscript,
  - (h) `tab[ $\bar{x}$ ]`: changes the  $x$  coordinate for the next displayed object to  $\bar{x}$ ,
  - (i) `vtab[ $\bar{y}$ ]`: changes the  $y$  coordinate for the next displayed object to  $\bar{y}$ ,
  - (j) `hs[ $\bar{n}$ ]`: increases  $x$  by  $n$ ,
  - (k) `vs[ $\bar{n}$ ]`: increases  $y$  by  $n$ .

I hope to extend this set of commands to provide algorithmic formatting capabilities.

The file `alignedByEncode.pdf` that produced Fig. 7 for comparison with Figs. 3 and 4 was written by the following statement, that contains `encode` and `encodeString` expressions.

```
export[alignedByEncode =
  {AbsoluteThickness[.1],
   encode[ps[20], vtab[620], tab[20],
         tf["Times-Roman"], "a", tab[40],
         tf["Courier"], "a", tab[60],
         tf["Symbol"], "a"],
   Line[{{20, 620}, {80, 620}}],
```

```

encodeString["Courier", 10, 20, 600,
  "Henry"],
Line[{{20, 600}, {50, 600}}],
encodeString["Times-Roman", 6, 20, 580,
  "Mary"],
encodeString["Times-Roman", 6, 40, 580,
  "Elizabeth"],
Line[{{20, 580}, {70, 580}}],
encodeString["Times-Roman", 12, 20, 560,
  "Mary"],
encodeString["Times-Roman", 12, 50, 560,
  "Elizabeth"],
Line[{{20, 560}, {100, 560}}],
encode[tf["Courier"], ps[6], tab[20],
  vtab[540], "0", "1", "2", "3", "4", "5",
  "6", "7", "8", "9"],
Line[{{20, 540}, {60, 540}}]]]

```

The file `refinedTudors.pdf` that produced Fig. 8 was written by the following statements.

```

tudorTreeEdges =
{Line[{{200, 600}, {200, 590}}],
  Line[{{135, 590}, {265, 590}}],
  Line[{{135, 590}, {135, 580}}],
  Line[{{200, 590}, {200, 580}}],
  Line[{{265, 590}, {265, 580}}],
  Line[{{200, 570}, {200, 560}}],
  Line[{{145, 560}, {255, 560}}],
  Line[{{145, 560}, {145, 550}}],
  Line[{{200, 560}, {200, 550}}],
  Line[{{255, 560}, {255, 550}}]}

encodedTudorNames =
{encodeString[
  "Courier", 8, 178.4, 605, "Henry VII"],
  encodeString[
  "Courier", 8, 120.6, 575, "Arthur"],
  encodeString[
  "Courier", 8, 186.0, 575, "Henry VIII"],
  encodeString[
  "Courier", 8, 250.6, 575, "others"],
  encodeString[
  "Courier", 8, 135.4, 545, "Mary"],
  encodeString[
  "Courier", 8, 178.4, 545, "Elizabeth"],
  encodeString[
  "Courier", 8, 233.4, 545, "Edward VI"]}

export[refinedTudors =
  {tudorTreeEdges, encodedTudorNames}]

```

The alignment is imperfect but I believe it can be improved by fine tuning the offsets. I think that each letter has a range of offsets that are acceptable in one context, and a different range in another context, with very narrow overlap. I have been using just one or two contexts to determine the offsets for each letter, and picking an offset within the range of acceptability in a somewhat arbitrary manner.

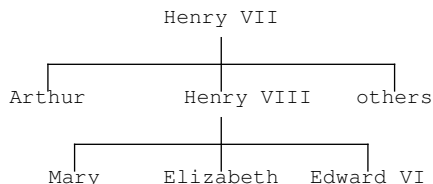


Figure 8: More output of encode expressions.

The present technique supports the alignment of short expressions with each other, as needed in the pulse sequence diagram of Fig. 1. That diagram, and the other diagrams in [1], were produced by *ad hoc* coding before I started TMG. Using TMG, I will be able to extend the options for including explanatory text in the diagrams, even in its present crude form. An algorithmic basis for TMG would enable many other applications of Mathematica graphics in the kernel mode.

### Acknowledgements

I thank Barbara Beeton, Karl Berry and Andrew Roberts for advice on coding and on sources of information.

### Supplementary material

Accompanying this article on the *TUGboat* web site is a collection of additional material:

1. the TMG software described in this note,
2. an account of how to measure widths and offsets,
3. software that I used to explore Mathematica fonts, with a detailed explanation.

### References

- [1] M. P. Barnett and I. Pelczer, Pulse sequence editing by symbolic calculation, *J. Magn. Reson.* 204 (2010) 189–195.
- [2] S. Wolfram, *The Mathematica Book*, 2nd ed. Addison-Wesley, New York, 1991, or later editions.
- [3] Movable type. [http://en.wikipedia.org/wiki/Movable\\_type](http://en.wikipedia.org/wiki/Movable_type).
- [4] M. P. Barnett, *Computer Typesetting, Experiments and Prospects*, MIT Press. 1965.
- [5] Baseline (typography). [http://en.wikipedia.org/wiki/Baseline\\_\(typography\)](http://en.wikipedia.org/wiki/Baseline_(typography)).
- [6] PostScript Language Reference, Adobe Systems Incorporated. <http://www.adobe.com/products/postscript/pdfs/PLRM.pdf>.

◇ Michael P. Barnett  
 Meadow Lakes  
 Hightstown, NJ 08520, USA  
 michaelb (at) princeton dot edu  
<http://www.princeton.edu/~michaelb/nmr/>



## The Treasure Chest

This is a list of selected new packages posted to CTAN (<http://ctan.org>) from May 2010 through October 2010, with descriptions based on the announcements and edited for brevity.

Entries are listed alphabetically within CTAN directories. A few entries which the editors subjectively believed to be of especially wide interest or otherwise notable are starred; of course, this is not intended to slight the other contributions.

We hope this column and its companions will help to make CTAN a more accessible resource to the T<sub>E</sub>X community. Comments are welcome, as always.

◇ Karl Berry  
<http://tug.org/ctan.html>

---

### fonts

**adorn** in **fonts**  
 Ornaments font.

**adfsymbols** in **fonts**  
 Includes arrows and bullets.

**auto1** in **fonts**  
 L<sup>A</sup>T<sub>E</sub>X support for the Underware Auto 1 fonts.

**baskervaldadf** in **fonts**  
 Font family based on Baskerville.

**berenisadf** in **fonts**  
 Berenis Pro ADF.

**cm-unicode** in **fonts**  
 Computer Modern Unicode.

**electrumadf** in **fonts**  
 Slab serif font family.

**gillcm** in **fonts**  
 Unslanted italic CM fonts following Eric Gill's ideas.

**jamtimes** in **fonts**  
 Expanded Times Roman fonts with math based on Belleek.

**mdputu** in **fonts**  
 Unslanted digits in Adobe Utopia italics.

**oldstandard** in **fonts**  
 Unicode font for classical and medieval studies.

**poltawski** in **fonts**  
 Extensive font family; replaces **antp**.

**punknova** in **fonts**  
 OpenType version of Don Knuth's Punk font.

**romandeadf** in **fonts**  
 Font family somewhat based on Caslon.

**bonita** in **fonts/softmakerfreefont**  
 L<sup>A</sup>T<sub>E</sub>X support files for Softmaker Bonita.

**\*stix** in **fonts**  
 Unicode mathematics font collection.

**tfrupee** in **fonts**  
 Font with the new rupee symbol.

**xits** in **fonts**  
 STIX with additional OpenType math support.

---

### graphics

**bodegraph** in **graphics/pgf/contrib**  
 Draw Bode, etc., plots with Gnuplot and TikZ.

**duotenzor** in **graphics**  
 Draw circuit and duotensor diagrams via TikZ.

**numericplots** in **graphics**  
 Plot numeric data using PSTricks.

**pst-electricfield** in **graphics/pstricks/contrib**  
 Draw electric field and equipotential lines.

**pst-magneticfield** in **graphics/pstricks/contrib**  
 Draw magnetic field lines of Helmholtz coils.

---

### info

**Math-E** in **info/examples**  
 Examples from *Typesetting Mathematics with L<sup>A</sup>T<sub>E</sub>X*.

**pstricks\_calcnodes** in **info**  
 Illustrates using PSTricks for calculus lecture notes.

**svg-inkscape** in **info**  
 Including SVG images in L<sup>A</sup>T<sub>E</sub>X via Inkscape.

**tex-font-errors-cheatsheet** in **info**  
 Cheat sheet for the most common T<sub>E</sub>X font errors.

---

### macros/generic

**lecturer** in **macros/generic**  
 Slide support for any format.

**\*texapi** in **macros/generic**  
 Writing format-independent packages.

**yax** in **macros/generic**  
 Yet Another Key System.

---

### macros/latex/contrib

**acroflex** in **macros/latex/contrib**  
 Use SWF file to create a graphing screen.

**aeb\_mlink** in **macros/latex/contrib**  
 Multi-line link support.

**annot\_pro** in **macros/latex/contrib**  
 Text, stamp, and file attachment annotations.

**arrayjobx** in **macros/latex/contrib**  
 Array data structures.

**\*cals** in **macros/latex/contrib**  
 Typeset multipage tables with headers, footers, cell spanning and decorations.

**calxxxx-yyyy** in **macros/latex/contrib**  
 Printing calendars for chosen years and languages.

**macros/latex/contrib/calxxxx-yyyy**

- `chemfig` in `macros/latex/contrib`  
Draw molecules with an easy syntax.
- `chextras` in `macros/latex/contrib`  
Companion package for Swiss typesetting.
- `chronology` in `macros/latex/contrib`  
Horizontal timeline with day granularity.
- `cntdwn` in `macros/latex/contrib`  
Support short and long countdowns, and clocks in any time zone.
- `colordoc` in `macros/latex/contrib`  
Color braces in doc lists.
- `cutwin` in `macros/latex/contrib`  
Create a “window” in a paragraph.
- `drawstack` in `macros/latex/contrib`  
Draw execution stacks.
- `drs` in `macros/latex/contrib`  
Draw Discourse Representation Structures.
- `elteikthesis` in `macros/latex/contrib`  
Thesis class for ELTE University Informatics.
- `equell` in `macros/latex/contrib`  
Fine exclamation, question, and ellipsis marks.
- `esk` in `macros/latex/contrib`  
Encapsulate Sketch files in L<sup>A</sup>T<sub>E</sub>X source.
- `fjodor` in `macros/latex/contrib`  
Layout options for small books.
- `hrefhide` in `macros/latex/contrib`  
Display but do not print a hyperlink.
- `imakeidx` in `macros/latex/contrib`  
Produces indices during a typesetting run.
- `inputtrc` in `macros/latex/contrib`  
Tracing which file loads which.
- `jmlr` in `macros/latex/contrib`  
Class for the *Journal of Machine Learning Research*.
- `linegoal` in `macros/latex/contrib`  
Length remaining on the line.
- `locality` in `macros/latex/contrib`  
Implementation of basic scoping.
- `logreq` in `macros/latex/contrib`  
Log requests to run external files in a machine-readable format.
- `marginfix` in `macros/latex/contrib`  
Patch `marginpar` routines to prevent overflowing or misalignment.
- `mylatexformat` in `macros/latex/contrib`  
Dump a `.fmt` based on any preamble.
- `pagesLTS` in `macros/latex/contrib`  
Define labels for last pages.
- `papermas` in `macros/latex/contrib`  
Compute mass of printed form of document.
- `progressbar` in `macros/latex/contrib`  
Visualize shares of a total amount as a bar.
- `realscripts` in `macros/latex/contrib`  
Use OpenType features to replace `\textsuperscript` and `\textsubscript` where possible.
- `rmannot` in `macros/latex/contrib`  
Rich media annotations.
- `russ` in `macros/latex/contrib`  
Russian letters in T<sub>E</sub>X control sequences, Russian hyphenation, and more, independent of `babel`.
- `rvwrite` in `macros/latex/contrib`  
Help for insufficient `\write` registers.
- `serbianpart` in `macros/latex/contrib`  
Part numbers in Serbian.
- `simplecd` in `macros/latex/contrib`  
CD/DVD covers for printing.
- `skb` in `macros/latex/contrib`  
Build document repository for long-lived documents.
- `skeycommand` in `macros/latex/contrib`  
Create commands using parameters and keys together.
- `spot` in `macros/latex/contrib/beamer-contrib`  
Spotlight highlighting for Beamer.
- `suftesi` in `macros/latex/contrib`  
Typesetting theses, especially in the humanities.
- `unicode-math` in `macros/latex/contrib`  
Unicode math support for X<sub>Y</sub>T<sub>E</sub>X and LuaT<sub>E</sub>X.
- `undolabl` in `macros/latex/contrib`  
Override existing labels, especially automatically generated ones.
- `uowthesis` in `macros/latex/contrib`  
University of Wollongong thesis class.
- `yt4pdf` in `macros/latex/contrib`  
Play YouTube videos in a PDF.
- 
- macros/luatex**
- `lualibs` in `macros/luatex/generic`  
Lua modules useful for general programming.
- `luaotfload` in `macros/luatex/generic`  
OpenType support for LuaT<sub>E</sub>X (based on but outside of ConT<sub>E</sub>Xt).
- \*`luatexbase` in `macros/luatex/generic`  
Basic facilities for LuaT<sub>E</sub>X macro programmers.
- `luatextrace` in `macros/luatex/generic`  
User-level LuaT<sub>E</sub>X macro goodies.
- 
- macros/plain**
- `present` in `macros/plain/contrib`  
Customizable presentations in plain T<sub>E</sub>X.
- 
- support**
- `adobemapping` in `support`  
Collected Adobe cmap and pdfmapping files.
- `dktools` in `support`  
Image-related tools and libraries.
- `ltxfileinfo` in `support`  
Print information about a L<sup>A</sup>T<sub>E</sub>X package to stdout.
- `texlog_extract` in `support`  
Colored summary of messages from a log file.

**ArsTeXnica #9 (October 2010)**

*ArsTeXnica* is the journal of G<sub>J</sub>T, the Italian T<sub>E</sub>X user group (<http://www.guit.sssup.it/>).

GIANLUCA PIGNALBERI, Editoriale [From the editor]; pp. 3–4

A short overview of the present issue.

GIANGIACOMO BRAVO, Reciprocità e attaccamento al gruppo nel forum G<sub>J</sub>T [Reciprocity and appreciation for the G<sub>J</sub>T forum group]; pp. 5–14

This paper studies the provision of public goods in open-source software support forums. Data from the G<sub>J</sub>T were analyzed to find individual motives for offering help. Using this methodology, we were able to split the forum participants into a small intrinsically motivated core group and a much larger group motivated mainly on the basis of reciprocity. The motives of the two groups were largely complementary and jointly produced a situation where the overwhelming majority of questions received an appropriate answer. At the same time, the core group played a fundamental role and was the key in explaining the forum's success. Without this group, the forum's performance would have been considerably diminished, probably down to a level that would not justify its existence.

TOMMASO GORDINI, Scrivere un indirizzo postale [How to write a postal address]; pp. 15–23

We'll describe all the rules to write postal addresses exactly according to Italian standard. You can find here also a simple document class to print addresses directly on an envelope with L<sup>A</sup>T<sub>E</sub>X.

GIANLUCA PIGNALBERI, Cicli, test e calcoli angolari per disegni non banali con METAPOST [Loops, tests and angular computations for non-trivial drawings in METAPOST]; pp. 24–30

A fair number of introductory guides to METAPOST are available online; a good selection comes along with the T<sub>E</sub>X distributions. Unfortunately, sometimes the authors don't succeed in treating the topics fully: some details get hidden, lost or left to other similar documents. In this paper we'll see how some non-trivial drawings for a short thesis on Galileo Galilei were done, having the chance to study in detail some of the manuals' explanations.

GUSTAVO CEVOLANI, Composizione automatica dell'indice dei nomi con biblatex [Automatic composition of a list of names with biblatex]; pp. 31–38

Most academic and specialist publications are required to contain an *index of names*. The **biblatex**

package offers, for the first time, a simple and direct way of *automatically* generating the index of names. This paper briefly explains how to generate the index of names using **biblatex**, with reference to some minimal working examples. The paper assumes that the reader is familiar with **BIBTEX** and the **makeidx**, **index** and **biblatex** packages.

IVAN VALBUSA, Creare stili bibliografici con biblatex: l'esperienza del pacchetto biblatex-philosophy [How to create bibliographic styles with biblatex: the experience of the package biblatex-philosophy]; pp. 39–50

The aim of this article is to describe the genesis and the main features of the bibliography and citation styles provided by the **biblatex-philosophy** package; moreover, it provides the basic concepts to create a style for use with Philipp Lehman's **biblatex** package. This article requires basic knowledge of **BIBTEX** and **biblatex**.

LUIGI SCARSO, Fell Types in ConT<sub>E</sub>Xt; pp. 51–56

In this paper we will briefly show how to install and use an OpenType font with ConT<sub>E</sub>Xt MkIV. We will use the *Fell Types* fonts as in M. Dominici's paper "Utilizzo di caratteri TrueType con L<sup>A</sup>T<sub>E</sub>X. Un esempio pratico: i *Fell Types*". A problem with an unusual font parameter is described and a solution offered by ConT<sub>E</sub>Xt MkIV is discussed.

ENRICO GREGORIO, L'arte esoterica di scrivere in cirillico con L<sup>A</sup>T<sub>E</sub>X [The esoteric art of writing in Cyrillic with L<sup>A</sup>T<sub>E</sub>X]; pp. 57–73

Writing words in the Cyrillic script with L<sup>A</sup>T<sub>E</sub>X is easy once we know some small tricks of the trade. With **babel** it's also easy to write a document with longer parts in a language using the Cyrillic script. We describe also some small defects of **babel** in this area and some ways to correct them.

CLAUDIO BECCARI and HEINRICH FLECK, I mark, questi sconosciuti [Marks, those unknowns]; pp. 74–78

Marks are useful for typesetting headers, but their inner workings are rather mysterious. We try to uncover their secrets with an important example: the composition of a dictionary.

[Received from Gianluca Pignalberi.]

**MAPS 40 (2009)**

MAPS is the publication of NTG, the Dutch language T<sub>E</sub>X user group (<http://www.ntg.nl>).

**MAPS 40 (Spring 2010)**

TACO HOEKWATER, Redactioneel [From the editor]; p. 1

Overview.

HANS HAGEN, The font name mess; pp. 2–8

Font names as well as file names of fonts are highly inconsistent across vendors, within vendors and platforms. As we have to deal with this matter, in ConT<sub>E</sub>Xt MkIV we have several ways to address a font: by file name, by font name, and by specification. In this article I describe all three.

KEES VAN DER LAAN, Circle Inversions; pp. 9–65

Circle inversions are exercised and drawn with PostScript operators which are also included in this plain T<sub>E</sub>X article. Interesting pictures will be shown, resulting from inversion of straight line pieces and other procedures.

I demonstrate a way to calculate the circle of anti-similitude, by which two circles are inverses of each other. Furthermore, I show how one can transform two distinct circles into two concentric circles, and how to draw a circle orthogonal to a circle which passes through one or two points within the circle is done via the circle inversion technique.

The above is generalized into finding the circle which cuts the boundary at an arbitrary angle, e.g. 80 degrees, and passes through a point within the circle. Orthogonal circular arcs can form an Escher-like grid, as he used in his *Circle Limit* drawings. Four variants of the grid of *Circle Limits III* have been included. The first cuts the boundary at 80 degrees, the second at 90 degrees, and the third with a mixture of both. The fourth is Coxeter's solution.

A smiley pattern is inverted in (orthogonal) circular arcs within a circle with the aid of PostScript's `pathforall` by (repeated use of) circle inversion. How to draw a circle orthogonal to 1, 2 or 3 other distinct circles is shown. Apollonius' problem is solved by the use of the circle inversion transformation and also by transforming the three quadratic equations into one non-linear equation and a 2x2 system of linear equations, and then solving these equations in PostScript and MetaPost. A closer look yields that we only have to solve one quadratic equation in  $r$ , the radius of the wanted circle, in order to obtain the solution of Apollonius' problem.

Coding problems in MetaPost will be mentioned and circumvented. I demonstrate the way one can

create and use a PostScript library. A plea is made for creating and maintaining a PostScript library of operators, graphics and utilities. A snapshot of this growing library is included. A few tiny but handy PostScript operators are given next to a (numerical) PostScript operator to solve a 3x3 linear system of equations, where partial pivoting is implemented and the calculations are done with the accuracy of the underlying computer arithmetic, which is much better than MetaPost's accuracy at present. How to overload a PostScript operator, e.g. `length`, is given. The question of whether the PostScript library can be used in MetaPost is answered.

The core of the paper is twofold: first the rediscovery that Apollonius' problem is solved by the solution of a quadratic equation, and second the Apollonius operator, which reflects this rediscovery and can be used to obtain all 8 solutions of Apollonius' problem. Another gem is `Apollonius2`, which is suited for the case that one circle contains the other two. The culmination of it all is the operator `radical` for drawing the radical circle of three given distinct circles.

EDITORIAL, EuroT<sub>E</sub>X 2010 announcement; p. 66  
(Cancelled.)

HANS HAGEN, Grouping in hybrid environments; pp. 67–71

[Enhancements to groups for, e.g., background colors and underlining, in ConT<sub>E</sub>Xt MkIV.]

EDITORIAL, Fourth ConT<sub>E</sub>Xt meeting announcement; pp. 72–72

September 13–18, Břejlov (Prague), Czech Republic.

LUIGI SCARSO, OpenType PostScript fonts with unusual units-per-em values; pp. 73–79

OpenType fonts with PostScript outlines are usually defined in a dimensionless workspace of 1000 x 1000 units per em (upm). Adobe Reader exhibits strange behaviour with PDF documents which embed an OpenType PostScript font with unusual upm. This paper describes a solution implemented by LuaT<sub>E</sub>X that resolves this problem.

PIET VAN OOSTRUM, Een uittreksel uit de recente bijdragen in het CTAN archief [Selected recent contributions to CTAN]; pp. 80–83

This article describes some recent contributions to the CTAN archive (and other Internet sources). The selection reflects what interests me and what I think others may be interested in. It is, therefore, a personal choice, not a comprehensive review.

[Received from Wybo Dekker.]

---

**The *PracTeX* Journal 2010-1**

*The PracTeX Journal* is an online publication of the TeX Users Group. Its web site is <http://tug.org/pracjourn>. All articles are available there.

Issue theme: L<sup>A</sup>T<sub>E</sub>X academic work bench.

FRANCISCO REINALDO, From the Editor  
Editorial; next issue: L<sup>A</sup>T<sub>E</sub>X for teachers.

THE EDITORS, News from Around  
Knuth volume 4 and Google talk; CSI typeface; other journals; one million L<sup>A</sup>T<sub>E</sub>X math formulas at [latexsearch.com](http://latexsearch.com).

CLAUDIO BECCARI, Some PDF/A tricks  
This short contribution explains how to fix some font problems when creating PDF/A documents, the new standard for archival PDF documents.

ALAN BRASLAU, Chemical structures with PPCH<sub>T</sub>E<sub>X</sub>  
Chemical formulas and chemical structures can be included in a L<sup>A</sup>T<sub>E</sub>X or a ConT<sub>E</sub>Xt document easily using the PPCH<sub>T</sub>E<sub>X</sub> macros. We present here a simple introduction to their use. Additionally, a more extensive tutorial is available in the documentation of the package.

KLAUS DOHMEN, Dual screen presentations with the L<sup>A</sup>T<sub>E</sub>X beamer class under X  
We show how the ‘X Resize, Rotate and Reflect Extension’ of the X Window System can be used to display a L<sup>A</sup>T<sub>E</sub>X beamer presentation on one or two beamers while simultaneously displaying the output of both beamers on the lecturer’s display. If only one beamer is used, the lecturer’s display might show both the output of the beamer and hidden notes.

MASSIMILIANO DOMINICI, L<sup>A</sup>T<sub>E</sub>X e CSV [L<sup>A</sup>T<sub>E</sub>X and CSV]  
In this paper we will present some techniques and a few examples about handling data in comma-separated-value format. We will focus mainly on two packages specifically aimed at this purpose: `datatool` and `pgfplots`. (In Italian.)

ARACELE GARCIA and ARTHUR BUCHSBAUM, Sobre as ferramentas em L<sup>A</sup>T<sub>E</sub>X que os estudantes de Lógica deveriam conhecer [About L<sup>A</sup>T<sub>E</sub>X tools that students of logic should know]

In this article, we share our experience with Prac<sub>T</sub>E<sub>X</sub> readers about L<sup>A</sup>T<sub>E</sub>X and the toolbox that students of Formal Logic of the Master in Computer Science from the Federal University of Santa Catarina (UFSC) in Brazil are using to prepare handouts, books, articles, dissertations and solving exercises.

We present some tools we have found useful for students who are developing projects in formal logic: proof styles, useful sites, styles of numbering and referencing of proclamations, references in BIB<sub>T</sub>E<sub>X</sub> format and suggestions of reading. The work done in this area requires a certain formality and rigor, thus we believe that such features can be successfully aimed at by the use of L<sup>A</sup>T<sub>E</sub>X. (In Portuguese.)

MARCO ANTONIO GOMEZ-MARTIN and PEDRO PABLO GOMEZ-MARTIN, Continuous integration in L<sup>A</sup>T<sub>E</sub>X

Have you ever co-written a paper using L<sup>A</sup>T<sub>E</sub>X together with some version control system such as SVN? Have you ever updated your local copy and the compilation become broken due to a previous bad commit? Continuous integration avoids this problem using an auxiliary server that constantly checks the sanity of the repository, compiling the L<sup>A</sup>T<sub>E</sub>X documents after each commit, and notifying authors of possible problems. This paper describes how to configure this environment. Although the configuration effort is detailed, it is done only once and provides many benefits. In addition to doing compilation tests, all authors can be automatically informed by email when a new version is committed, and the current .pdf version can be made available to third parties on the Web.

IVAN GRIFFIN and ITA RICHARDSON, Using L<sup>A</sup>T<sub>E</sub>X for qualitative data analysis

L<sup>A</sup>T<sub>E</sub>X, in addition to its typesetting role, has considerable potential as a tool to assist in workflow automation for *Qualitative Data Analysis* (QDA) of collected research data.

RICHARD HARDWICK, Automatic report generation with your text editor, Perl, and L<sup>A</sup>T<sub>E</sub>X

I describe a simple system for producing standard evaluation reports. The evaluator writes plain text files. A Perl script reads the text files and uses the Perl module `Template.pm`, with a ready-made L<sup>A</sup>T<sub>E</sub>X template, to generate the final L<sup>A</sup>T<sub>E</sub>X report.

JIM HEFFERON, Giving away a book  
[Reprinted with revisions in this *TUGboat*.]

TOMAS MORALES DE LUNA, Useful vector graphic tools for L<sup>A</sup>T<sub>E</sub>X users

This paper presents some useful tools for creating vector graphics that can be included in L<sup>A</sup>T<sub>E</sub>X documents. Of all the tools available, we focus on those that can produce graphics easily, and that can include any L<sup>A</sup>T<sub>E</sub>X math formula. In particular, we present three useful tools: `Xfig`, `LaTeXDraw`, and `Matplotlib`. While the two first are intended to

produce sketches and figures, the last will produce graphs, charts and contours.

FRANCISCO REINALDO ET AL., Gerando Certificados Acadêmicos e inserindo Assinaturas Digitalizadas [Generating academic certificates]

In this paper we present how ordinary users can generate academic certificates with scanned signatures automatically by using CSV and a few instructions in  $\LaTeX 2_{\epsilon}$ . (In Portuguese.)

FRANCISCO REINALDO ET AL., Doxygen e  $\LaTeX 2_{\epsilon}$ : As definitivas ferramentas para documentar seu código-fonte [Developing software with Doxygen &  $\LaTeX$ ]

In this paper we present how programmers can document source code and have updated reports during the elaboration/implementation phase. We focus mainly on two tools specifically aimed at this purpose: Doxygen and  $\LaTeX 2_{\epsilon}$ . (In Portuguese.)

FRANCISCO REINALDO ET AL., Projeto Interdisciplinar (PI) em  $\LaTeX 2_{\epsilon}$ : Um modelo de relatório para a academia [A student report template]

In this paper we present an example of a technical report commonly used by students to present their academic research. (In Portuguese.)

FRANCISCO REINALDO ET AL., Guia Visual Definitivo para Instalação de  $\LaTeX 2_{\epsilon}$  e suas Ferramentas de Apoio [Six  $\LaTeX$  tools (with videos)]

In this paper we present the most promising  $\LaTeX 2_{\epsilon}$  tools for common users and how these tools should be fine-tuned. We focus mainly on six heterogeneous tools specifically aimed at this purpose: Mi $\TeX$ , GSview, e $\text{X}$ Per $\text{T}$  PDF Reader, Texmaker, JabRef, and LaTable. (In Portuguese.)

LUIGI SCARSO, Playing with Flash in Con $\TeX$ t MkIV

A first attempt to adapt `flashmovie.sty` to Con $\TeX$ t MkIV to produce a flash movie with MetaPost and `swftools`.

HERBERT SCHULZ, Enhancing command completion for TeXShop

$\LaTeX$  environments and commands are rather wordy markup. These make the intentions of the author easy to determine but more difficult to write. Using command completion, authors can write a few letters and trigger an expansion into complete environments and commands along with ways of going between arguments of those commands. In this paper I present an enhancement to command completion in TeXShop that allows more consistent completions and inclusion of short comments to help

authors remember the order and contents of the arguments to those environments and commands.

FRANCESC SUÑOL, Tools for creating  $\LaTeX$ -integrated graphics and animations under GNU/Linux

This paper describes how to easily create graphics and animations that can be included in  $\LaTeX$  documents. This article discusses three kinds of figures: plots, schematics, and pictures. The tools presented here can quickly generate plots, and are based on simple Gnuplot and Bash scripts that display the final result on the screen. Ipe is an excellent program to deal with complex figures and schematics, and the `animate` package is used to make a series of figures change over time to simulate a movie. All the programs used in this article are free software.

EVAN WESSLER, An argument for learning  $\LaTeX$ : Benefits beyond typesetting  
[Published in *TUGboat* 31:1.]

DAVID WALDEN, Travels in  $\TeX$  Land: memoir, TtH, and a booklet signature

In this column in each issue I have mused on my wanderings around the  $\TeX$  world. In this issue I describe three efforts. In section 1, I describe my first attempt to use the *memoir* class to produce a book. In section 2, I describe my first time using TtH to convert from  $\LaTeX$  to HTML. In section 3, I describe creating a 16-page booklet signature using a method described by another author in an earlier issue of this journal.

This will be my final  $\TeX$  Land column in this journal. I am pleased to have provided a column for every previous issue, but it is now time for me to focus on other things. I won't stop using  $\TeX$ , however, and probably will continue to write about  $\TeX$  once in a while, but without the concerns of a regular column. I wish the editors of this journal "all the best" as they continue publication of *The Prac $\TeX$  Journal*.

FRANCESCO REINALDO ET AL., Book review:  *$\LaTeX$  Quick Start*

In this paper we review the book  *$\LaTeX$  Quick Start: A first guide to document preparation* from a user's viewpoint, and give a candid assessment of its contents. (In Portuguese and English; the book reviewed is in English.)

THE EDITORS, Ask Nelly

How can I have the author name for a quotation set on the same line as the quotation or on a new line, according to space requirements?

THE EDITORS, Distractions: Typesetting a fancy curriculum vitae



**Zpravodaj 20(1–2) and 20(3), 2010**

Editor's note: *Zpravodaj* is the journal of  $\mathcal{C}\mathcal{S}\mathcal{T}\mathcal{U}\mathcal{G}$ , the  $\text{\TeX}$  user group oriented mainly but not entirely to the Czech and Slovak languages (<http://www.cstug.cz>).

**Zpravodaj 20(1–2), 2010**

PAVEL STRÍŽ, Z letních dnů roku 2010 [Greetings from the Summer holidays 2010], p. 1.

MARTIN BUDAJ, Divide et impera: program `findhyph` [Divide and conquer — the `findhyph` program], pp. 2–5

The article presents a simple computer program, `findhyph`, which generates a list of all words hyphenated in documents processed by  $\text{\TeX}$ . This program can be downloaded from CTAN.

JIRÍ RYBIČKA, PETRA TALANDOVÁ, JAN PŘICHYSTAL, Počítačová podpora výběru optimálních programů pro zpracování textů [Computer-aided optimal program selection for document processing], pp. 6–13

This article deals with generalization of possibilities for preparing electronic documents of various types. Computer support is proposed for optimization of program equipment selection. It takes into account user requirements for different programs and document properties.

PAVEL STRÍŽ, RADEK BENDA, Editace PDF souboru aneb O jednom dnu [Editing PDF files], pp. 14–22

The real-world problem of deleting specific text parts in PDF files of hundreds of pages occurred out of the blue sky and the deadline was to finish the task within 24 hours. This article presents our experience with editing PDF files using different proprietary software and trial versions as well as tools and programs from the world of open source software.

DENIS ROEGEL, Kulové plochy, hlavní kružnice a rovnoběžky [Spheres, great circles and parallels], pp. 23–38

[Czech translation of the English article from *TUGboat* 30:1. Translation by Pavel Stríž.]

HERBERT VOSS, The current state of the PSTricks project [Současný vývoj a novinky v balíčce PSTricks], pp. 39–67

[Published in *TUGboat* 31:1.]

DENIS ROEGEL, Anatomy of a macro (tutorial) [O rozboru jednoho makra (tutoriál)], pp. 68–76

[Published in *TUGboat* 22:1/2.]

J. H. SILVERMAN,  $\text{\TeX}$  reference card (for plain  $\text{\TeX}$ ) [Syntaxe jazyka  $\text{\TeX}$  (formátu plain  $\text{\TeX}$ )], pp. 77–78

Reprint of <http://refcards.com/docs/silvermanj/tex/tex-refcard-a4.pdf>.

KLAAS BALS, TONY GRAHAM, Extensible stylesheet language requirements, version 2.0, working draft, 26 March 2008 [Požadavky na XSL-FO verze 2.0], pp. 79–120

The XSL 1.1 specification defines the features and syntax for the Extensible Stylesheet Language (XSL), a language for expressing stylesheets. This paper enumerates the collected requirements for a 2.0 version of XSL. There are two parts to XSL: XSL Transformations (XSLT) for transformation of documents and XSL Formatting Objects (XSL-FO) for formatting of documents. This is the requirements document for XSL-FO and not for XSLT.

This article is approximately a printed version of <http://www.w3.org/TR/xslfo20-req/>.

PAVEL STRÍŽ, VÍT ZÝKA, MICHAL MÁDR, Nové a staronové knihy [New and older books], pp. 121–126

Michel Goossens, Frank Mittelbach, Sebastian Rahtz, Denis Roegel, Herbert Voß: *The L<sup>A</sup>T<sub>E</sub>X Graphics Companion*, second edition, Addison-Wesley Professional, 2007, in English.

Herbert Voß: *PSTricks – Grafik mit PostScript für  $\text{\TeX}$  and L<sup>A</sup>T<sub>E</sub>X*, fifth edition, DANTE e.V., 2008, in German.

Herbert Voß: *PSTricks: Graphics and PostScript for  $\text{\TeX}$  and L<sup>A</sup>T<sub>E</sub>X*, first edition, Cambridge, 2010, in English.

Bill Casselman: *Mathematical Illustrations: A Manual of Geometry and PostScript*, first edition, 2005, in English. Also available online at <http://www.math.ubc.ca/~cass/graphics/manual/>.

František Štorm: *Eseje o typografii* [Essays on typography], first edition, Revolver Revue, Prague 2008, in Czech.

Radana Lencová: *Rozhovory o písmu rukopisném* [Interviews on Handwriting], first edition, Svet Publishers, Prague, 2007, in Czech. Interviews with 24 leading figures of Czech typography and graphic design, including samples of their handwriting and work.

Radana Lencová: *Comenia Script, praktický manuál* [Comenia Script, Manual], first edition, Svet Publishers, Prague, 2008, in Czech. Practical Manual for a new handwriting style for schools.

Jan Jeřábek: *Grafologie – více než diagnostika osobnosti* [Graphology — more than a personality

diagnosis], fifth edition, Argo, Prague, 2003, in Czech.

Vilém Schönfeld: *Učebnice vědecké grafologie pro začátečníky* [Textbook on scientific graphology for beginners], fourth edition, Elfa, Prague, 2007, in Czech.

JANO KULA, PAVEL STRÍŽ,  $\zeta$ TUG čtenáře srdečně zve! [An invitation to the fourth ConT $\text{\TeX}$ t meeting and the third T $\text{\TeX}$ perience conference], p. 127–128

EDITORS,  $\zeta$ TUG členem CrossRef [ $\zeta$ TUG is a new member of CrossRef], p. 129–131

EDITORS, Redakční poznámky a pokyny autorům [Notices and instructions for authors], pp. 132–136  
In Czech and English.

### **Zpravodaj 20(3), 2010**

PAVEL STRÍŽ, Úvodníček [Welcoming letter from the editors], p. 137.

DENIS ROEGEL, Jednoduché makro `suanpan` na kreslení čínského a japonského abaku [Simple macros for drawing Chinese and Japanese abaci], pp. 138–151

[Czech translation of the English article from *TUGboat* 30:1. Translation by Pavel Stríž.]

TIMOTHY EYRE, Typesetting Japanese with p $\text{\TeX}$  [Sazba japonštiny pomocí p $\text{\TeX}$ u], pp. 152–173

p $\text{\TeX}$  is a  $\text{\TeX}$ -like typesetting system that is specifically designed for typesetting Japanese and is widely used in Japan. This article describes how to acquire, set up and use a p $\text{\TeX}$  system in practice, with an emphasis on font management. It also provides basic background information on Japanese text processing and alternatives to p $\text{\TeX}$ .

Latest news: p $\text{\TeX}$  and p $\text{\LaTeX}$  formats are now available for your convenience in T $\text{\TeX}$  Live, starting in 2010.

KAZUOMI KUNIYOSHI, Japanese formatting rules for X $\text{\TeX}$  [Pravidla sazby japonštiny v X $\text{\TeX}$ u], pp. 174–175

This is a two-page report with information about the reasons and the existence of the `genzi` package which sets Japanese formatting rules for X $\text{\TeX}$ . The package, samples and more comments can be viewed and downloaded from the author's web site, <http://kuniyoshi.fastmail.fm/xetex/>.

KEN LUNDE, OpenType Japanese Font Tutorial: Kazuraki [Kazuraki: tutoriál k japonskému OTF písmu], pp. 176–198

Adobe Systems' Type Engineering & Design team in Japan has developed a ground-breaking and innovative new typeface design that breaks the

mold that has constrained Japanese typefaces for decades. The typeface design, created by Adobe's own Ryoko Nishizuka, was inspired by the calligraphy of the 12th century Japanese calligrapher and writer Fujiwara-no-Teika, and its final production to produce a functional OpenType font leveraged three powerful AFDKO (Adobe Font Development Kit for OpenType) tools, `tx`, `mergeFonts`, and `rotateFont`, to implement its complex metrics.

Kazuraki is unique among mainstream Japanese typefaces in that it is fully proportional, in both writing directions. Some glyphs are wider than they are tall, and some are taller than they are wide, and this is reflected in their metrics. For this reason, and because subtle shifting is required for correct positioning of each glyph, there are separate glyphs for both writing directions. In other words, for the 1,082 kanji that are supported in the current version, the font contains 1,082 glyphs for horizontal use, and 1,082 glyphs for vertical. In addition, Kazuraki also includes a significant number of two-, three-, and four-character hiragana ligatures for vertical use.

The tutorial that is reprinted here in its entirety is designed to guide font developers in building special-purpose OpenType fonts, using Kazuraki as an example of how to build a fully-proportional Japanese font. The current version is always available at [http://www.adobe.com/devnet/font/pdfs/5901.Kazuraki\\_Tutorial.pdf](http://www.adobe.com/devnet/font/pdfs/5901.Kazuraki_Tutorial.pdf).

The Kazuraki specimen book, which demonstrates how this font can be used, is available at [http://store4.adobe.com/type/browser/pdfs/Kazuraki\\_SPN.pdf](http://store4.adobe.com/type/browser/pdfs/Kazuraki_SPN.pdf).

TIMOTHY EYRE, Creating a kanji stroke order font [Jak na výrobu písma kandži s pořadím tahů], pp. 199–207

This article describes how a font that displays kanji stroke orders can be created from thousands of SVG files containing this information.

The latest version of Kanji Stroke Orders Font (KSOF) can be downloaded from the author's site, <http://sites.google.com/site/nihilistorguk/>.

TIMOTHY EYRE, PDFdiff: A PDF file comparison Script [PDFdiff: skript srovnávající PDF soubory], pp. 208–214

A Python script that can be used to take two PDF files and automatically process them with `pdftk`, `Ghostscript`, `ImageMagick` and X $\text{\TeX}$  to produce a PDF file that shows the differences between the two input files.

JJGOD JIANG, Chinese T $\text{\TeX}$  typesetting: Past and present [Sazba čínštiny v T $\text{\TeX}$ u: historie a současnost], pp. 215–219

The article introduces and gives an overview of Chinese  $\TeX$  typesetting from its early beginnings to the present day.

DENIS ROEGEL, Sudoku s vepsanými kandži: integrace čínských glyfů s grafikou na úrovni  $\text{METAPOSTu}$  [Kanji-Sudokus: Integrating Chinese and graphics], pp. 220–226

[Czech translation of the English article from *TUGboat* 29:2. Translation by Pavel Stríž.]

PETER WILSON, The sudoku bundle [Balíček `sudokubundle`], pp. 227–241

The sudoku bundle provides a coordinated set of packages for displaying, solving, and generating Sudoku puzzles. This article describes some of the internal aspects of the packages.

PAVEL STRÍŽ, MICHAL MÁDR, Nové a staronové knihy [New and older books], pp. 242–249

Ken Lunde: *CJKV Information Processing: Chinese, Japanese, Korean & Vietnamese Computing*, second edition, O'Reilly Media, 2008.

Jukka K. Korpela: *Unicode Explained: Internationalize Documents, Programs, and Web Sites*, first edition, O'Reilly Media, 2006.

The Unicode Consortium: *The Unicode Standard, Version 5.0*, fifth edition, Addison-Wesley Professional, 2006.

Richard Gillam: *Unicode Demystified: A Practical Programmer's Guide to the Encoding Standard*, first edition, Addison-Wesley Professional, 2002.

Joe Becker, Richard Gillam, Mark Davis: *Unicode Guide: The Ultimate Reference Guide to the Universal Character Encoding Standard*, first edition, Barcharts, 2006.

Ulrik Vieth: Book review: *Fonts & Encodings* by Yannis Haralambous, first edition, O'Reilly Media, 2007. Translation to Czech by Marcel Svitalský.

KAREL HORÁK, Osmnáctý Bacho $\TeX$ : Typografové a programátoři – vzájemné inspirace (April 30 – May 5, 2010) [Typographers and programmers: mutual inspirations — Bacho $\TeX$  2010], <http://www.gust.org.pl/bachotex/2010/>, pp. 250–252

PAVEL STRÍŽ, Denní pásmo seminářů TypeTalks 2010 [Impressions from TypeTalks 2010], <http://typetalks.com/Symposium2010/>, pp. 253–259.

MILOŠ BREJCHA, Svět knihy Praha 2010 [16th International Book Fair and Literary Festival, Prague Exhibition Grounds, 2010], <http://www.bookworld.cz/en/> and <http://www.svetknihy.cz/en/>, pp. 260–261

THE EXECUTIVE BOARD OF  $\zeta$ TUG, Zápis z valné hromady  $\zeta$ TUG [A report from an annual  $\zeta$ TUG meeting], pp. 262–264

### TypeTalks 2010 Symposium

The theme of this conference in Brno, Czech Republic, was type. This is a broad area embracing the history of type, the design of type, type education, the use of type (typography) and much more. The key criteria for the acceptance of a talk was that it have educational value.

There were seven invited speakers:

- Florian Hardwig (D): Localize! The dialects of handwriting in type design;
- Rob Keller (US/D): Font technology is crazy!;
- Michael Hochleitner (AT) A contemporary view on the relationship of lettering and type;
- Tomáš Brousil (CZ): A new font family Tabac;
- Dan Reynolds (US/D) The passion of the young, multi-script type designer;
- Dan Rhatigan (US/UK): How I learned to stop worrying and love bad type; and
- Veronika Burian (CZ/D): Typographic match-making.

The conference web site is <http://www.typetalks.com>.

[Received from Pavel Stríž.]

---

### *Die T<sub>E</sub>Xnische Komödie 2010/3*

*Die T<sub>E</sub>Xnische Komödie* is the journal of DANTE e.V., the German-language T<sub>E</sub>X user group (<http://www.dante.de>). (Editorial items are omitted.)

ACHIM SCHAFFRINNA, Anatomie der Buchstaben [The anatomy of letters]; pp. 11–15

Compared to the other article this article is not T<sub>E</sub>X-related but rather offers basic knowledge about typography. It is a work in progress, the author encourages all readers to participate in explaining the introduced terms and their graphical representation. Certainly there are more terms perfectly fitting into this list.

HEIKO OBERDIEK AND CHRISTINE RÖMER, Anzeigen der Trennstellen [Showing hyphenated words]; pp. 16–16

Sometimes it may be of interest to see how T<sub>E</sub>X will potentially hyphenate words. With the macro `\hyphenated{...text...}` the output shows all possible hyphenations of every word.

MARCO DANIEL, Das Paket `mdframed` [The `mdframed` package]; pp. 18–21

What might another frame package be good for? I asked myself this question as well, since so far I had been satisfied by the `framed` package written by Donald Arseneau. But when I realized I could not avoid the closing line on the first and the beginning horizontal line on the second page and searching the web also revealed no results for this issue I decided to implement this, based on the `framed` package with the help of `listings.sty` which offers this option.

UWE ZIEGENHAGEN, In Tabellen rechnen mit `spreadtab` [Calculating in tables with `spreadtab`]; pp. 22–26

With a syntax comparable to common spreadsheet applications the `spreadtab` package by Christian Tellechea offers simple calculations inside L<sup>A</sup>T<sub>E</sub>X tables. In this article the package is introduced and used in a more complex example to typeset invoices.

UWE ZIEGENHAGEN, PocketMods mit L<sup>A</sup>T<sub>E</sub>X erstellen [Creating Pocketmods with L<sup>A</sup>T<sub>E</sub>X]; pp. 27–32

Pocketmods are small booklets which consist of a single piece of paper that is cut and folded in a special way. In this article I show several ways to create such a Pocketmod.

ROLF NIEPRASCHK, Zierlinien [Trimlines]; pp. 33–34

In the following it is shown with the example of trim lines (also called “English lines”) how freely

available graphics files found on the Internet can be used in documents.

DOMINIK WAGENFÜHR, Unicode-Zeichen in L<sup>A</sup>T<sub>E</sub>X nutzen [Using Unicode characters in L<sup>A</sup>T<sub>E</sub>X]; pp. 35–37

The time when special characters such as German umlauts had to be encoded as e.g., “a are long gone. Thanks to UTF-8 support it is possible today to even use other special characters with L<sup>A</sup>T<sub>E</sub>X.

DOMINIK WAGENFÜHR, L<sup>A</sup>T<sub>E</sub>X-Symbole: Einfügen mit LSS [Inserting L<sup>A</sup>T<sub>E</sub>X symbols with LSS]; pp. 38–41

In the previous article we explained how to use Unicode characters with L<sup>A</sup>T<sub>E</sub>X documents. Another alternative for finding symbols is the L<sup>A</sup>T<sub>E</sub>X Symbols Selector, LSS.

[Received from Herbert Voß.]

### **This TUGboat issue’s epigraph**

The quotes on the title page of this *TUGboat* issue come from email between the editors and Chuck Bigelow in the course of discussing future Lucida projects. Chuck suggested the following references from *The Journal of Typographic Research* for anyone who is curious about the slashed-zero debate:

Dirk Wendt, “O or 0?”, vol. 3, no. 3, July 1969, pp. 241–248.

Allen G. Vartabedian, “A Proposed Fontstyle for the Graphic Representation of the Oh and Zero”, vol. 3, no. 3, July 1969, pp. 249–258.

Hermann Zapf, “Letter to the Editor” re: Vartabedian, “A Proposed Fontstyle ...”, vol. 4, no. 2, Spring 1970, pp. 179–180.

Allen G. Vartabedian, “Reply to Zapf”, vol. 4, no. 2, Spring 1970, pp. 181–183.

Chuck adds:

There’s doubtless a lot more of such stuff, especially if you include screeds on-line, but these thoughtful papers and letters were published early in the era of computerized typography and were written by an illustrious designer (Zapf), a good academic psychologist studying typography (Wendt), and an engineer working on related problems at Bell Labs (Vartabedian), so they show the diversity of views when such issues were emerging.

Enjoy!

## Calendar

### 2010

Nov 5–  
Mar 20 “Marking Time”: A traveling juried exhibition of books by members of the Guild of Book Workers. Dartmouth College, Hanover, New Hampshire. Sites and dates are listed at [palimpsest.stanford.edu/byorg/gbw](http://palimpsest.stanford.edu/byorg/gbw)

Nov 6–8 The Eighth International Conference on the Book, University of St. Gallen, St. Gallen, Switzerland. [booksandpublishing.com/conference-2010](http://booksandpublishing.com/conference-2010)

Nov 19 “Letterpress: Forward thinking”, St Bride Library, London, England. [stbride.org/events](http://stbride.org/events)

Jun 19–22 Digital Humanities 2011, Alliance of Digital Humanities Organizations, Stanford University, Palo Alto, California. [www.digitalhumanities.org](http://www.digitalhumanities.org)

Jul TypeCon 2011, New Orleans, Louisiana. [www.typecon.com](http://www.typecon.com)

Jul 14–17 SHARP 2011, “The Book in Art & Science”, Society for the History of Authorship, Reading & Publishing. Washington, DC. [www.sharpweb.org](http://www.sharpweb.org)

Aug 7–11 SIGGRAPH 2011, Vancouver, Canada. [www.siggraph.org/s2011](http://www.siggraph.org/s2011)

Sep 14–18 Association Typographique Internationale (ATypI) annual conference, Reykjavik, Iceland. [www.atypi.org](http://www.atypi.org)

Sep 19–24 The fifth ConT<sub>E</sub>Xt user meeting, Porquerolles, France. [meeting.contextgarden.net/2011](http://meeting.contextgarden.net/2011)

Oct 14–15 American Printing History Association’s 36<sup>th</sup> annual conference, “Printing at the Edge”, University of California, San Diego, California, [www.printinghistory.org/about/calendar.php](http://www.printinghistory.org/about/calendar.php)

Oct 14–16 The Ninth International Conference on the Book, University of Toronto, Ontario, Canada. [booksandpublishing.com/conference-2011](http://booksandpublishing.com/conference-2011)

---

### 2011

Jan 13–16 College Book Art Association Biennial Conference, “Word, Image, Text, Object”, University of Indiana, Bloomington, Indiana. [www.collegebookart.org](http://www.collegebookart.org)

Jan 28 “The Design of Understanding”, St Bride Library, London, England. [stbride.org/events](http://stbride.org/events)

Feb 1 **TUG election:** nominations due. [tug.org/election](http://tug.org/election)

Apr 29–  
May 3 EuroBach<sub>T</sub>E<sub>X</sub> 2011: 19<sup>th</sup> Bach<sub>T</sub>E<sub>X</sub> Conference, “Aesthetics and effectiveness of the message, cultural contexts”, Bachotek, Poland. [www.gust.org.pl/bachotex/bachotex2011-en](http://www.gust.org.pl/bachotex/bachotex2011-en)

---

### TUG 2011 Cairo, Egypt.

Nov 14–17 The 32<sup>nd</sup> annual meeting of the T<sub>E</sub>X Users Group. [tug.org/tug2011](http://tug.org/tug2011)

---

*Status as of 1 November 2010*

For additional information on TUG-sponsored events listed here, contact the TUG office (+1 503 223-9994, fax: +1 206 203-3960, e-mail: [office@tug.org](mailto:office@tug.org)). For events sponsored by other organizations, please use the contact address provided.

A combined calendar for all user groups is online at [texcalendar.dante.de](http://texcalendar.dante.de).

Other calendars of typographic interest are linked from [tug.org/calendar.html](http://tug.org/calendar.html).

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

The information here comes from the consultants themselves. We do not include information we know to be false, but we cannot check out any of the information; we are transmitting it to you as it was given to us and do not promise it is correct. Also, this is not an official endorsement of the people listed here. We provide this list to enable you to contact service providers and decide for yourself whether to hire one.

TUG also provides an online list of consultants at <http://tug.org/consultants.html>. If you'd like to be listed, please see that web page.

### Aicart Martinez, Mercè

Tarragona 102 4<sup>o</sup> 2<sup>a</sup>  
08015 Barcelona, Spain  
+34 932267827  
Email: [m.aicart \(at\) ono.com](mailto:m.aicart@ono.com)  
Web: <http://www.edilatex.com>

We provide, at reasonable low cost, T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X typesetting services to authors or publishers worldwide. We have been in business since the beginning of 1990. For more information visit our web site.

### Dangerous Curve

PO Box 532281  
Los Angeles, CA 90053  
+1 213-617-8483  
Email: [typesetting \(at\) dangerouscurve.org](mailto:typesetting@dangerouscurve.org)  
Web: <http://dangerouscurve.org/tex.html>

We are your macro specialists for T<sub>E</sub>X or L<sup>A</sup>T<sub>E</sub>X fine typography specs beyond those of the average L<sup>A</sup>T<sub>E</sub>X macro package. If you use X<sub>Y</sub>T<sub>E</sub>X, we are your microtypography specialists. We take special care to typeset mathematics well.

Not that picky? We also handle most of your typical T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X typesetting needs.

We have been typesetting in the commercial and academic worlds since 1979.

Our team includes Masters-level computer scientists, journeyman typographers, graphic designers, letterform/font designers, artists, and a co-author of a T<sub>E</sub>X book.

### Hendrickson, Amy

Brookline, MA, USA  
Email: [amyh \(at\) texnology.com](mailto:amyh@texnology.com)  
Web: <http://www.texnology.com>

L<sup>A</sup>T<sub>E</sub>X macro writing our speciality for more than 25 years: macro packages for major publishing companies, author support; journal macros for American Geophysical Union, Proceedings of the National Academy of Sciences, and many more.

Scientific journal design/production/hosting, e-publishing in PDF or html.

### Hendrickson, Amy (cont'd)

L<sup>A</sup>T<sub>E</sub>X training, at MIT, Harvard, many more venues. Customized on site training available.

Please visit our site for samples, and get in touch. We are particularly glad to take on adventurous new uses for L<sup>A</sup>T<sub>E</sub>X, for instance, web based report generation including graphics, for bioinformatics or other applications.

### Peter, Steve

295 N Bridge St.  
Somerville, NJ 08876  
+1 732 306-6309  
Email: [speter \(at\) mac.com](mailto:speter@mac.com)

Specializing in foreign language, linguistic, and technical typesetting using T<sub>E</sub>X, L<sup>A</sup>T<sub>E</sub>X, and ConT<sub>E</sub>Xt, I have typeset books for Pragmatic Programmers, Oxford University Press, Routledge, and Kluwer, among others, and have helped numerous authors turn rough manuscripts, some with dozens of languages, into beautiful camera-ready copy. I have extensive experience in editing, proofreading, and writing documentation. I also tweak and design fonts. I have an MA in Linguistics from Harvard University and live in the New York metro area.

### Shanmugam, R.

No. 38/1 (New No. 65), Veerapandian Nagar, Ist St.  
Choolaimedu, Chennai-600094, Tamilnadu, India  
+91 9841061058  
Email: [rshanmugam92 \(at\) yahoo.com](mailto:rshanmugam92@yahoo.com)

As a Consultant, I provide consultation, training, and full service support to individuals, authors, typesetters, publishers, organizations, institutions, etc. I support leading BPO/KPO/ITES/Publishing companies in implementing latest technologies with high level automation in the field of Typesetting/Prepress, ePublishing, XML2PAGE, WEBTechnology, DataConversion, Digitization, Cross-media publishing, etc., with highly competitive prices. I provide consultation in building business models & technology to develop your customer base and community, streamlining processes in getting ROI on our workflow, New business opportunities through improved workflow, Developing eMarketing/E-Business Strategy, etc. I have been in the field BPO/KPO/ITES, Typesetting, and ePublishing for 16 years, handled various projects. I am a software consultant with Master's Degree. I have sound knowledge in T<sub>E</sub>X, L<sup>A</sup>T<sub>E</sub>X<sub>2 $\epsilon$</sub> , XMLT<sub>E</sub>X, Quark, InDesign, XML, MathML, DTD, XSLT, XSL-FO, Schema, ebooks, OeB, etc.

**Sievers, Martin**

Im Treff 8, 54296 Trier, Germany  
+49 651 4936567-0

Email: [info \(at\) schoenerpublizieren.com](mailto:info@ schoenerpublizieren.com)

Web: <http://www.schoenerpublizieren.com>

As a mathematician with ten years of typesetting experience I offer  $\text{\TeX}$  and  $\text{\LaTeX}$  services and consulting for the whole academic sector (individuals, universities, publishers) and everybody looking for a high-quality output of his documents.

From setting up entire book projects to last-minute help, from creating individual templates, packages and citation styles ( $\text{\BIBTeX}$ ,  $\text{\biblatex}$ ) to typesetting your math, tables or graphics — just contact me with information on your project.

**Veytsman, Boris**

46871 Antioch Pl.  
Sterling, VA 20164

+1 703 915-2406

Email: [borisv \(at\) lk.net](mailto:borisv@lk.net)

Web: <http://www.borisv.lk.net>

$\text{\TeX}$  and  $\text{\LaTeX}$  consulting, training and seminars. Integration with databases, automated document preparation, custom  $\text{\LaTeX}$  packages, conversions and much more. I have about fifteen years of experience in  $\text{\TeX}$  and twenty-eight years of experience in teaching & training. I have authored several packages on CTAN, published papers in  $\text{\TeX}$  related journals, and conducted several workshops on  $\text{\TeX}$  and related subjects.

## TUG Institutional Members

American Mathematical Society,  
*Providence, Rhode Island*

Aware Software, Inc.,  
*Midland Park, New Jersey*

Banca d'Italia,  
*Roma, Italy*

Center for Computing Sciences,  
*Bowie, Maryland*

Certicom Corp.,  
*Mississauga, Ontario, Canada*

CSTUG, *Praha, Czech Republic*

Florida State University,  
School of Computational Science  
and Information Technology,  
*Tallahassee, Florida*

IBM Corporation,  
T J Watson Research Center,  
*Yorktown, New York*

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

MacKichan Software, Inc.,  
*Washington/New Mexico, USA*

Marquette University,  
Department of Mathematics,  
Statistics and Computer Science,  
*Milwaukee, Wisconsin*

Masaryk University,  
Faculty of Informatics,  
*Brno, Czech Republic*

MOSEK ApS,  
*Copenhagen, Denmark*

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

Springer-Verlag Heidelberg,  
*Heidelberg, Germany*

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*

## 2011 T<sub>E</sub>X Users Group election

Jim Hefferon for the Elections Committee

The positions of TUG President and nine members of the Board of Directors will be open as of the 2011 Annual Meeting, which will be held in November 2011 in Cairo, Egypt.

The directors whose terms will expire in 2011: Barbara Beeton, Jon Breitenbucher, Kaja Christiansen, Susan DeMeritt, Ross Moore, Cheryl Ponchin, and Philip Taylor. Two additional director positions are currently unoccupied.

Continuing directors, with terms ending in 2013, are: Jonathan Fine, Steve Grathwohl, Jim Hefferon, Klaus Höppner, Steve Peter, and David Walden.

The election to choose the new President and Board members will be held in Spring of 2011. Nominations for these openings are now invited.

The Bylaws provide that “Any member may be nominated for election to the office of TUG President/to the Board by submitting a nomination petition in accordance with the TUG Election Procedures. Election . . . shall be by written mail ballot of the entire membership, carried out in accordance with those same Procedures.” The term of President is two years.

The name of any member may be placed in nomination for election to one of the open offices by submission of a petition, signed by two other members in good standing, to the TUG office at least two weeks (14 days) prior to the mailing of ballots. (A candidate’s membership dues for 2011 will be expected to be paid by the nomination deadline.) The term of a member of the TUG Board is four years.

A nomination form follows this announcement; forms may also be obtained from the TUG office, or via <http://tug.org/election>.

Along with a nomination form, each candidate must supply a passport-size photograph, a short biography, and a statement of intent to be included with the ballot; the biography and statement of intent together may not exceed 400 words. The deadline for receipt of nomination forms and ballot information at the TUG office is **1 February 2011**. Forms may be submitted by FAX, or scanned and submitted by e-mail to [office@tug.org](mailto:office@tug.org).

Ballots will be mailed to all members within 30 days after the close of nominations. Marked ballots must be returned no more than six (6) weeks following the mailing; the exact dates will be noted on the ballots.

Ballots will be counted by a disinterested party not affiliated with the TUG organization. The results of the election should be available by early June, and will be announced in a future issue of *TUGboat* as well as through various T<sub>E</sub>X-related electronic lists.

## 2011 TUG Election — Nomination Form

Only TUG members whose dues have been paid for 2011 will be eligible to participate in the election. The signatures of two (2) members in good standing at the time they sign the nomination form are required in addition to that of the nominee. **Type or print** names clearly, using the name by which you are known to TUG. Names that cannot be identified from the TUG membership records will not be accepted as valid.

The undersigned TUG members propose the nomination of:

**Name of Nominee:** \_\_\_\_\_

Signature: \_\_\_\_\_

Date: \_\_\_\_\_

for the position of (check one):

**TUG President**

**Member of the TUG Board of Directors**

for a term beginning with the 2011 Annual Meeting, **November 2011**

1. \_\_\_\_\_  
(please print)

\_\_\_\_\_ (signature) \_\_\_\_\_ (date)

2. \_\_\_\_\_  
(please print)

\_\_\_\_\_ (signature) \_\_\_\_\_ (date)

Return this nomination form to the TUG office (forms submitted by FAX or scanned and submitted by e-mail will be accepted). Nomination forms and all required supplementary material (photograph, biography and personal statement for inclusion on the ballot) must be received in the TUG office no later than **1 February 2011**.<sup>1</sup> It is the responsibility of the candidate to ensure that this deadline is met. Under no circumstances will incomplete applications be accepted.

- nomination form
- photograph
- biography/personal statement

T<sub>E</sub>X Users Group **FAX:** +1 206 203-3960  
**Nominations for 2011 Election**  
 P. O. Box 2311  
 Portland, OR 97208-2311  
 U.S.A.

<sup>1</sup> Supplementary material may be sent separately from the form, and supporting signatures need not all appear on the same form.





## Introductory

- 158 *Barbara Beeton* / Editorial comments  
• typography and *TUGboat* news
- 158 *Karl Berry* / From the President  
• conferences; interviews; software
- 175 *Jim Hefferon* / Giving it away  
• writing and distributing a free (libre) linear algebra text
- 191 *Luca Merciadri* / Some misunderstood or unknown L<sup>A</sup>T<sub>E</sub>X<sub>2 $\epsilon$</sub>  tricks II  
• float references; spreadsheets; QED symbols; counting pages and tables; messages on blank pages; dots in matrices; logic gates; circled-text enumerations
- 194 *L<sup>A</sup>T<sub>E</sub>X Project Team* / L<sup>A</sup>T<sub>E</sub>X<sub>3</sub> news, issue 4  
• `expl3` in practice; new `xpackages`; developments with `expl3`; T<sub>E</sub>X Users Group 2010 reflections

## Intermediate

- 227 *Karl Berry* / The treasure chest  
• new CTAN packages from April through October, 2010
- 161 *Christophe Caignaert* / A story of *kpfonts*: Reaching the limits of NFSS  
• extensive article on an extensive font family
- 219 *Jean-Michel Hufflen* / Managing printed and online versions of large educational documents  
• adapting texts to distance vs. presentational education
- 177 *Peter Wilson* / Glisterings  
• meandering miniature books
- 195 *Joseph Wright* / From `\newcommand` to `\DocumentNewCommand` with `xparse`  
• L<sup>A</sup>T<sub>E</sub>X<sub>3</sub> extensions to defining macros for users

## Intermediate Plus

- 213 *Matteo Centonza* and *Vito Piserchia* / `illumino`: An XML document production system with a T<sub>E</sub>X core  
• an Apache `ant` workflow for perfect L<sup>A</sup>T<sub>E</sub>X to Docbook conversion
- 184 *Paul Isambert* / Three things you can do with LuaT<sub>E</sub>X that would be extremely painful otherwise  
• LuaT<sub>E</sub>X introduction, and representing page color, underlining, margin notes
- 208 *Oleg Parashchenko* / Generate T<sub>E</sub>X documents using `pdfscript`  
• Python library for conveniently generating correct T<sub>E</sub>X documents

## Advanced

- 223 *Michael Barnett* / Aligning text in diagrams exported by Mathematica: A question about the PostScript infrastructure  
• help requested in finding proper baseline alignments in PDF output
- 197 *Hans Hagen* / Tagged PDF in ConT<sub>E</sub>Xt  
• towards tagged PDF support in ConT<sub>E</sub>Xt
- 203 *Luigi Scarso* / Introduction to colours in ConT<sub>E</sub>Xt MkIV  
• theoretical, practical, and ConT<sub>E</sub>Xt colors

## Contents of publications from other T<sub>E</sub>X groups

- 230 *ArsT<sub>E</sub>Xnica*: Issue 9 (October 2010); *MAPS*: Issue 40 (2010); *The PracT<sub>E</sub>X Journal*: Issue 2010-1; *Zpravodaj*: Issues 20(1-2), 20(3) (2010); *Die T<sub>E</sub>Xnische Komödie*: Issue 2010/3;

## Reports and notices

- 160 *Barbara Beeton* / Hyphenation exception log  
• update for missed and incorrect U.S. English hyphenations
- 237 Calendar
- 238 T<sub>E</sub>X consulting and production services
- 239 Institutional members
- 240 *Jim Hefferon* / T<sub>E</sub>X Users Group 2011 election

# TUGBOAT

Volume 31, Number 3 / 2010

|                                      |     |  |
|--------------------------------------|-----|--|
| <b>General Delivery</b>              | 158 | From the president / <i>Karl Berry</i>   |
|                                      | 158 | Editorial comments / <i>Barbara Beeton</i><br>Matthew Carter named MacArthur Fellow;<br>Indie Excellence Awards for self-published books;<br>City maps made entirely of type; U&lc on line;<br>Some “under-the-covers” uses of T <sub>E</sub> X; Beyond literate programming |
|                                      | 160 | Hyphenation exception log / <i>Barbara Beeton</i>  |
| <b>Fonts</b>                         | 161 | A story of <i>kpfonts</i> : Reaching the limits of NFSS / <i>Christophe Caignaert</i>  |
| <b>Publishing</b>                    | 175 | Giving it away / <i>Jim Hefferon</i>   |
|                                      | 177 | Glisterings: Meandering miniature books / <i>Peter Wilson</i>  |
| <b>Software &amp; Tools</b>          | 184 | Three things you can do with LuaT <sub>E</sub> X that would be extremely painful otherwise /<br><i>Paul Isambert</i>   |
| <b>L<sup>A</sup>T<sub>E</sub>X</b>   | 191 | Some misunderstood or unknown L <sup>A</sup> T <sub>E</sub> X <sub>2<math>\epsilon</math></sub> tricks II / <i>Luca Merciadri</i>  |
| <b>L<sup>A</sup>T<sub>E</sub>X 3</b> | 194 | L <sup>A</sup> T <sub>E</sub> X3 news, issue 3 / <i>L<sup>A</sup>T<sub>E</sub>X Project Team</i>   |
|                                      | 195 | From <code>\newcommand</code> to <code>\DocumentNewCommand</code> with <code>xparse</code> / <i>Joseph Wright</i>  |
| <b>ConT<sub>E</sub>Xt</b>            | 197 | Tagged PDF in ConT <sub>E</sub> Xt / <i>Hans Hagen</i>   |
|                                      | 203 | Introduction to colours in ConT <sub>E</sub> Xt MKIV / <i>Luigi Scarso</i>   |
| <b>Electronic Documents</b>          | 208 | Generate T <sub>E</sub> X documents using <code>pdfscript</code> / <i>Oleg Parashchenko</i>  |
|                                      | 213 | <code>illumino</code> : An XML document production system with a T <sub>E</sub> X core /<br><i>Matteo Centonza and Vito Piserchia</i>  |
|                                      | 219 | Managing printed and online versions of large educational documents /<br><i>Jean-Michel Hufflen</i>  |
| <b>Problems</b>                      | 223 | Aligning text in diagrams exported by Mathematica: A question about the<br>PostScript infrastructure / <i>Michael Barnett</i>  |
| <b>Hints &amp; Tricks</b>            | 227 | The treasure chest / <i>Karl Berry</i>   |
| <b>Abstracts</b>                     | 229 | <i>ArsT<sub>E</sub>Xnica</i> : Contents of issue 9 (October 2010)  |
|                                      | 230 | <i>MAPS</i> : Contents of issue 40 (2010)  |
|                                      | 231 | <i>The PracT<sub>E</sub>X Journal</i> : Contents of issue 2010-1   |
|                                      | 233 | <i>Zpravodaj</i> : Contents of issues 20(1–2), 20(3) (2010)  |
|                                      | 236 | <i>Die T<sub>E</sub>Xnische Komödie</i> : Contents of issue 2010/3   |
| <b>News</b>                          | 237 | Calendar   |
| <b>Advertisements</b>                | 238 | T <sub>E</sub> X consulting and production services  |
| <b>TUG Business</b>                  | 239 | TUG institutional members  |
|                                      | 240 | TUG 2011 election  |

## Introductory

- 158 *Barbara Beeton* / Editorial comments  
 • typography and *TUGboat* news
- 158 *Karl Berry* / From the President  
 • conferences; interviews; software
- 175 *Jim Hefferon* / Giving it away  
 • writing and distributing a free (libre) linear algebra text
- 191 *Luca Merciadri* / Some misunderstood or unknown L<sup>A</sup>T<sub>E</sub>X<sub>2 $\epsilon$</sub>  tricks II  
 • float references; spreadsheets; QED symbols; counting pages and tables; messages on blank pages; dots in matrices; logic gates; circled-text enumerations
- 194 *L<sup>A</sup>T<sub>E</sub>X Project Team* / L<sup>A</sup>T<sub>E</sub>X<sub>3</sub> news, issue 4  
 • `expl3` in practice; new `xpackages`; developments with `expl3`; T<sub>E</sub>X Users Group 2010 reflections

## Intermediate

- 227 *Karl Berry* / The treasure chest  
 • new CTAN packages from April through October, 2010
- 161 *Christophe Caignaert* / A story of *kpfonts*: Reaching the limits of NFSS  
 • extensive article on an extensive font family
- 219 *Jean-Michel Hufflen* / Managing printed and online versions of large educational documents  
 • adapting texts to distance vs. presentational education
- 177 *Peter Wilson* / Glisterings  
 • meandering miniature books
- 195 *Joseph Wright* / From `\newcommand` to `\DocumentNewCommand` with `xparse`  
 • L<sup>A</sup>T<sub>E</sub>X<sub>3</sub> extensions to defining macros for users

## Intermediate Plus

- 213 *Matteo Centonza* and *Vito Piserchia* / `illumino`: An XML document production system with a T<sub>E</sub>X core  
 • an Apache `ant` workflow for perfect L<sup>A</sup>T<sub>E</sub>X to Docbook conversion
- 184 *Paul Isambert* / Three things you can do with LuaT<sub>E</sub>X that would be extremely painful otherwise  
 • LuaT<sub>E</sub>X introduction, and representing page color, underlining, margin notes
- 208 *Oleg Parashchenko* / Generate T<sub>E</sub>X documents using `pdfscript`  
 • Python library for conveniently generating correct T<sub>E</sub>X documents

## Advanced

- 223 *Michael Barnett* / Aligning text in diagrams exported by Mathematica: A question about the PostScript infrastructure  
 • help requested in finding proper baseline alignments in PDF output
- 197 *Hans Hagen* / Tagged PDF in ConT<sub>E</sub>Xt  
 • towards tagged PDF support in ConT<sub>E</sub>Xt
- 203 *Luigi Scarso* / Introduction to colours in ConT<sub>E</sub>Xt MkIV  
 • theoretical, practical, and ConT<sub>E</sub>Xt colors

## Contents of publications from other T<sub>E</sub>X groups

- 230 *ArsT<sub>E</sub>Xnica*: Issue 9 (October 2010); *MAPS*: Issue 40 (2010); *The PracT<sub>E</sub>X Journal*: Issue 2010-1; *Zpravodaj*: Issues 20(1-2), 20(3) (2010); *Die T<sub>E</sub>Xnische Komödie*: Issue 2010/3;

## Reports and notices

- 160 *Barbara Beeton* / Hyphenation exception log  
 • update for missed and incorrect U.S. English hyphenations
- 237 Calendar
- 238 T<sub>E</sub>X consulting and production services
- 239 Institutional members
- 240 *Jim Hefferon* / T<sub>E</sub>X Users Group 2011 election