Your suggestion to publish my typography programs tends to solve the vexing question of how this should be written up, because it would be difficult to put the ideas into a normal technical paper without losing too much detail. . . .

One problem is that such a book would delay volume 4 yet again, but on the other hand this research will need to be properly published if it turns out as well as I hope.

D.E. Knuth, letter to C.A.R. Hoare
16 November 1977
This regular issue (Vol. 34, No. 1) is the first issue of the 2013 volume year. TUGboat is distributed as a benefit of membership to all current TUG members. It is also available to non-members in printed form through the TUG store (http://tug.org/store), and online at the TUGboat web site, http://tug.org/TUGboat. Online publication to non-members is delayed up to one year after print publication, to give members the benefit of early access.

Suggestions and proposals for TUGboat articles are gratefully accepted. Please submit contributions by electronic mail to TUGboat@tug.org.

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

TUGboat editorial board
Barbara Beeton, Editor-in-Chief
Karl Berry, Production Manager
Boris Veytsman, Associate Editor, Book Reviews

Production team
William Adams, Barbara Beeton, Karl Berry, Kaja Christiansen, Robin Fairbairns, Robin Laakso, Steve Peter, Michael Sofka, Christina Thiele

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

TUGboat editorial information

Submitting items for publication
The deadline for receipt of final papers for the next issue is July 8, and for the proceedings issue, November 4.

The TUGboat style files, for use with plain TeX and LATEX, are available from CTAN and the TUGboat web site. We also accept submissions using ConTeX. Deadlines, tips for authors, and other information: http://tug.org/TUGboat/location.html

TUGboat advertising

TUG Institutional Members

American Mathematical Society, Providence, Rhode Island
Aware Software, Inc., Midland Park, New Jersey
Center for Computing Sciences, Bowie, Maryland
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
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
StackExchange, New York City, New York
Stanford University, Computer Science Department, Stanford, California
Stockholm University, Department of Mathematics, Stockholm, Sweden
University College, Cork, Computer Centre, Cork, Ireland
Université Laval, Ste-Foy, Québec, Canada
University of Oslo, Institute of Informatics, Blindern, Oslo, Norway
VTeX UAB, Vilnius, Lithuania
TUGboat, Volume 34 (2013), No. 1

Ab Epistulis
Steve Peter

TUG is only as strong and vibrant as its members, and I’d like to encourage as many to get involved as possible. 2013 is an election year for TUG, both for president and for several directors’ positions. The deadline for nominations is May 1, and an official announcement appears elsewhere in this issue.

CTAN
As described in Gerd Neugebauer’s article in this issue, the main http://www.ctan.org website has been updated and relaunched. The new site provides the information and functions of the old site in a new look & feel. This is a first step towards an improved user experience. Some of the features of the new site:

- Informative landing page
- Improved upload form with forwarding to the master hosts
- Browsing of the CTAN tree
- Listing, registering and monitoring of the CTAN mirrors
- Browsing the Catalogue (packages, authors, and topics)
- Availability of different skins to suit your taste

Most existing URLs have been preserved. If you encounter any problems, please see http://www.ctan.org/contact. Thanks to Gerd and all for their hard work.

Another aspect of CTAN has changed recently as well: the tug.ctan.org URL now points to the University of Utah, which provides improved connectivity. We recommend that CTAN mirrors in North America use it as their master source. Thanks to Nelson Beebe and colleagues, Jim Hefferon, and the CTAN maintainers, for making the move possible.

We continue to use and recommend the “multiplexor“ URL http://mirror.ctan.org for CTAN references, to make use of nearby mirrors.

Conferences
Several major conferences are planned for this year.

In chronological order:

- 7th Con\TeX{} meeting and \TeX{}perience 2013: Brejlov, Czech Republic, Sept. 23–29, 2013.

Regarding the TUG 2013 conference: registration is now open, and the website has the usual call for papers and other organizational information, in both English and Japanese. July 15 is the combined deadline for presentation proposals, bursary requests, and the early registration discount. Please see http://tug.org/tug2013.

Interviews and reviews
The TUG website features several new interviews since the last time I wrote. Board member David Walden continues to highlight some of the most creative people in the \TeX{} world on an ongoing basis. If you haven’t spent some time recently poking around the interview corner, you owe it to yourself to do so (http://tug.org/interviews/).

In addition to interviews, the website also has new book reviews, coordinated by board member Boris Veytsman. PSTricks: Graphics and PostScript for \TeX{} and \LaTeX{} by Herbert Voss was reviewed by Boris, and Just My Type: A Book About Fonts by Simon Garfield was reviewed by Dave Walden. See http://tug.org/books/#reviews for a complete listing of all reviews.

In addition to reviews, the TUG website has a section (http://tug.org/books/) that offers a listing of books of either \TeX{} or typography interest. A small portion of the sales benefits TUG so that we may continue to support various projects.

Also on our books site, CSLI Publications is now offering a 20% discount for TUG members on all their books, including the newly reissued (and corrected) Digital Typography by Donald E. Knuth. Such discounts are only one of the benefits available to you as a TUG member. Check out http://tug.org/books/#discounts for all the currently-available discounts.

Finally
On a lighter note, Don Knuth was featured in the web comic xkcd. Go to http://xkcd.com/1162/ and hover your mouse over the comic until you see the popup message.

Lastly, Marc van Dongen has kindly created and made available a nicely illustrated calendar for 2013 of \TeX{} material. Links are on the main TUG website.

Happy \TeX{}ing!

⋄ Steve Peter

president (at) tug.dot.org

http://tug.org/TUGboat/Pres
Editorial comments

Barbara Beeton

This is the year for \TeX bug reports

As noted on Don Knuth’s \TeX web pages, www-cs-faculty.stanford.edu/~knuth/abcde.html, he “intend[s] to check on purported bugs again in the years 2013, 2020, 2028, 2037, etc.” I expect to be asked for the accumulation in late autumn. So, fair warning, if you have any questions, please submit them soon — they have to be vetted before they can be sent to Don, and that takes time.

If you are submitting a report, please provide minimal, but thorough, documentation, using only plain \TeX for your examples.

Anything that can be documented as “not a bug” will be excluded from what is sent on; the bug checkers are very thorough and trustworthy, and if there’s any question, they will ask for more evidence. But as already pointed out, this takes time. Since the next review isn’t scheduled until 2020, you don’t want to miss this one.

Don Knuth in the news (again)

In a list of the “20 most influential scientists alive today” (www.superscholar.org/features/20-most-influential-scientists-alive-today/), Don appears as number eight. It’s not a surprise to find him in such good company, but the photo that accompanies the entry is surprising to anyone familiar with his \TeX pronouncements. The background image is the logo from his “\TeX of the future” talk presented at the San Francisco TUG meeting in 2010. For anyone who missed the talk, see TUGboat 32:2, pages 121–124, or watch the video at river-valley.tv/tug-2010/an-earthshaking-announcement.

A new \TeX calendar

Marc van Dongen has created a 2013 calendar with images that are mostly pictures drawn by TiKZ, and dates for \TeX and other typesetting-related events (as listed in the TUGboat calendar). A downloadable PDF file (A4 size) can be found at csweb.ucc.ie/~dongen/TeX-SX/12-13/TUGCalendar.pdf. A letter-size version is also available, as TUGCalendar-Letter.pdf

Marc (dongen (at) cs dot ucc dot ie) says, “I’m happy to update the calendar when people send me birthdays of \TeX celebrities and dates of major \TeX events. I also welcome emails about typos and suggestions for improvements.” (Please keep the descriptions short — space is limited.) He is also open to suggestions for images to be used next year; he suggests a showcase of \TeX typography, utilizing different languages and typefaces.

Compulsive Bodoni / the Parmigiano Typographic System

Go to www.compulsivebodoni.com/ for a look at a new font project, undertaken in honor of Giambattista Bodoni (1740–1813), the noted printer and amazingly prolific punchcutter, and timed to mark the 200th anniversary of his death.

The name of the project, the Parmigiano Typographic System, derives from the city, Parma, where Bodoni spent most of his life. The project aims to create “the most extended family of fonts ever to have been inspired by the greatest Italian punchcutter.”

The site opens with an excerpt from a short play highlighting some aspects of Bodoni’s personality. One doesn’t have to understand Italian to appreciate the fire and forceful presence expressed by the performer. One click takes you to a visual index of the site. Clicking on the element in the middle brings up a page of attractive posters advertising the project. Other pages highlight different fonts in the family, which includes (in addition to the familiar western alphabets) Armenian, Devanagari, Thai, and others. This is a rather large site, well structured, and fun to explore — an expedition which (for me) will have to be delayed until after this issue goes to press.

A more textual introduction to the project, with a good historical overview, can be found at ilovetypography.com/2013/03/14/a-compulsive-tribute-to-giambattista-bodoni/.

And to continue with the Bodoni theme, an unrelated project: www.typographyserved.com/gallery/Bodoni-in-red/3789729

Printing technology, old and new

An Encyclopedia Britannica film from 1947, “Making Books”, has been recirculated as a video by The Atlantic, at www.theatlantic.com/video/index/267036/. This is how books used to be made (and how films used to explain technology). Both have come a long way! But don’t stop there — the “more” link will take you to a page with another video that profiles a contemporary inventor (or hacker) who merges antique typewriters with computers and tablets “to create functioning writing machines”. The result is a hybrid that your parents certainly wouldn’t recognize.

Another current video shows how printing ink is made (www.broadsheet.ie/2013/03/10/how-ink-is-made/). This process matches Pantone colors for
use on modern presses; although highly automated, it still requires considerable intervention by skilled craftsmen to ensure a uniform and reproducible product.

Interactive and collaborative on-line \LaTeX

There are quite a few reasons why one would want to have access to an up-to-date \LaTeX compiler besides one on their own computer. For one thing, it doesn’t need to be carried around; for another, it can be used to collaborate with co-authors, assuring use of the same versions of required packages.

The number of on-line resources is increasing rapidly. Here’s a list of the ones I’ve learned about. No recommendations are implied; you should check them out for yourself to see if they’re suitable for your needs.

- Collaborative \LaTeX editor with preview in your web browser: it.slashdot.org/story/13/02/14/1814217/
- latex-lab, Web based \LaTeX editor for Google Docs: code.google.com/p/latex-lab/
- Scrib\TeX, “Create, share and compile your \LaTeX documents from anywhere”: www.scribtex.com
- ShareLaTeX, an online \LaTeX editor: www.sharelatex.com
- SpanDeX, “a collaborative solution for \LaTeX authors”: spandex.io
- Verb\TeX, “a free, collaborative LaTeX Editor for your Android device”: play.google.com/store/apps/details?id=verbosus.verbtex
- write\LaTeX: www.writelatex.com
- The Common \LaTeX Service Interface: github.com/scribtex/clsi. Somewhat different from the others, this is an API that attempts to provide a standard interface for multiple services.

A posting on the TeX.stackexchange forum discusses the features that should be included in a good on-line service: meta.tex.stackexchange.com/questions/3164/

Mapping math and scientific symbols to their meanings

In the TeX.stackexchange forum, a question has been posed regarding whether there exist any lists that provide mappings between math and scientific symbols and their meanings (tex.stackexchange.com/questions/101805).

Specifically, many (sub)fields have established notation, but there seems to be no by-field reference that can be accessed by potential users, symbol font designers, package writers, and others. Scott Pakin’s Comprehensive symbol list and the on-line tool Detexify are very helpful resources, but the first is often too broad, and the second, not yet “complete”.

I’ve seen this request numerous times, but when I inquired whether such lists existed, or if this was something that might be sponsored by the AMS, the answer to the first question was no, and the suggestion to provide one was rejected as “not practical” or “not needed”.

There are indeed some difficulties in creating such a resource; let’s look at mathematics, the area with which I’m most familiar.

- Many common symbols have different meanings in different areas.
- Established mathematicians will already know the notation in common use in the target field.
- A mathematician is free to define the notation to be used in a paper, and if there’s not already a well established symbol for a concept, a new one will often be selected based on its physical shape relative to that of symbols already used for related concepts, regardless of the new symbol’s meaning in other areas.
- An established mathematician will have little interest in making the effort to create such a list, and a graduate student will most likely be too busy with research on a dissertation to take the time to create a resource whose existence will garner nothing more than appreciation, when what is really important to the student is the degree.

The answer posted for the TeX.SX question describes how the STIX symbols collection was compiled—from pre-existing “needs” lists with no identification of why or relevance to any particular topic. But surely this knowledge does exist. If there’s interest in pursuing the creation of topical symbol lists, two possible places to start a discussion are TeX.sx or the mailing list forum math-font-discuss@tug.org.

Footnote: Detexify will be superseded by Sketch-A-Char (sketch-a-char.kirelabs.org/This is a work in progress, and at the moment, recognizes only some greek letters, although when it’s complete, it’s intended to identify all symbols in Unicode. To follow its progress, check out the blog at detexify.posterous.com/update-on-detexify.

◊ Barbara Beeton
http://tug.org/TUGboat
tugboat (at) tug dot org
CTAN: Relaunch of the Web portal

Gerd Neugebauer

Abstract
If you want to contribute something to the \TeX\ world you will find in CTAN the first place to drop your contribution. Conversely, CTAN is a valuable source of systems, packages, and information. The Web portal of CTAN has deserved a renewal. Now the relaunch of the CTAN portal is online.

1 Introduction
The Comprehensive Archive \TeX\ Network (CTAN) is the central repository for \TeX\-related material. For many users CTAN acts behind the scenes. They simply use the material from CTAN as available via the various \TeX\ distributions. CTAN has had a Web site (http://www.ctan.org, see figure 1) for a long time. The Web site serves as a landing place if you want to place material in the public repository, or if you want to search for packages or information.

![Figure 1: www.ctan.org](image1)

CTAN is the major repository of \TeX\-related material. This material can be browsed via the Web portal. Here you find the material organized in directories and files. You can navigate into the directories or download the files.

When you hit the directory associated with a package of the \TeX\ Catalogue then additional information is presented (see figure 2).

![Figure 2: Browsing a directory with Catalogue information](image2)

In some of the directories the functionality is presented to download the directory with all contents in a zip archive. This makes it simpler to get your hands on the files.

For packages which are prepared for the distributions \TeX\ Live or MiK\TeX\ the names of the package in those distributions are shown. Thus you can install the package without downloading and installing it manually.

2 Browsing CTAN

3 Uploading material

CTAN lives on its contributions. Those contributions can be submitted to CTAN via the Web portal (see figure 3). Here a form allows you to enter all relevant information about your contribution. Your contribution is then processed manually and usually appears on CTAN within a short period of time.

Formerly you had to manually select one of the main servers of CTAN to perform an upload. This is now performed through the portal. An appropriate server is selected and the upload forwarded to it. If one of the primary servers is not available then the other server is used automatically.

CTAN can only be as good as the \TeX\ community makes it. Thus, we urge you to strongly consider uploading your useful packages to CTAN.

Gerd Neugebauer
4 Package classification

The CTAN team maintains the “\TeX{} Catalogue”, a database of information about the packages found on CTAN or other places. The Catalogue information for each package includes the package name, the author, keywords for classification, and more. The CTAN portal allows you to browse alphabetic lists of the Catalogue information. For instance you can request a list of packages beginning with the letter A (see figure 4) or a list of authors beginning with the letter N. From there you can navigate to the detail page for the information you are looking for.

Figure 3: The upload page, http://www.ctan.org/upload

Figure 4: Browsing packages beginning with ‘A’, http://www.ctan.org/pkg

Of special interest are the topics. They can be used to find packages for a certain purpose, or to find packages similar to one you have at hand. This use case starts at a package page. From there you select one of the topics and receive a list of all packages with this topic.

5 Searching on CTAN

No Web site is complete without a search function of some kind. Thus the new CTAN portal has search too; it currently covers the information in the catalogue and the static pages.

The quick search on each page searches all available information. The result page is shown when you start a search. There you can refine your search and adjust more search parameters. Especially you can select in which kinds of information the search should be performed (see figure 5).

Figure 5: Search results and search parameters, http://www.ctan.org/search

For a future release of the portal it can be imagined to extend the search to the full contents of CTAN. In this, we could provide better differentiation than the general purpose search engines like Google; for example, we can indicate control sequences separately from regular text — \texttt{\textbackslash section} is not the same as \texttt{section}. In addition we could provide a link to the packages, enabling you to find the packages where a control sequence is defined, used, or just mentioned.

6 CTAN mirrors

The ‘N’ in CTAN stands for “Network”. This means that several servers are involved — 97 at the time of this writing. The two primary servers are used to maintain the data on CTAN. Based on these two servers, the coordinated mirrors duplicate the data and provide it to you.

CTAN provides a URL to select an appropriate mirror server: http://mirror.ctan.org. It is a good practice to use this to start a CTAN reference instead of naming an individual server whenever you
cite a package on CTAN. This has the advantage that the chances are good that a reader who uses this URL is directed to a server close by.

The CTAN portal provides information on the servers in the network, allows a new volunteer to register a server (see figure 6), and makes use of the distributed servers for downloading files.

Figure 6: Registering a CTAN mirror, http://www.ctan.org/mirroring

7 Behind the scenes

In this section I don’t want to describe all the details of the implementation. We can just have a look at a few interesting issues.

7.1 Hyphenation on the Web

The rendering of Web pages is done by the user’s browser. Many parameters influence the final appearance, e.g., the font used, the size of the font, and the width of the output window. These parameters can be influenced but not strictly determined on the server side.

Let’s first have a look at the width of the window. Many Web pages use a layout which uses a portion of the window with a fixed width. This results in empty space to either the left or the right. On smaller devices a horizontal scroll bar appears immediately. Neither variant is optimal for readers. Thus the CTAN portal tries to adapt the layout to the space available. Sometimes this is called responsive Web design. More on that later.

From typesetting on paper we know that long words can result in a sloppy right margin or large white holes in the text block. Thus we usually use hyphenation patterns to add appropriate places where words can be hyphenated and split across lines. Fortunately TeX does this for us. Unfortunately the browser is generally not so helpful.

For the CTAN portal I have used a module extracted from the ctext project (www.extex.org).

Gerd Neugebauer

Figure 7: Same text — different widths

The Comprehensive TeX Archive Network

The Comprehensive TeX Archive Network (CTAN) is the central place for all kinds of material around TeX. CTAN has many thousands of items which can be downloaded and used immediately.

Download a TeX System

CTAN provides complete ready to run TeX systems for various platforms:

- TUG Luca a cross-platform TeX system. It includes support for most Unix-like systems, including GNU/Linux, Mac OS X, and Windows.
- MacTeX is a TeX system for Mac OS X. It is also included in a native Mac installer. The MacTeX front-end is installed as well.
- MiKTeX is an easy to install TeX system for Windows. It is installed on MiKTeX with a .msi file to guide your installation and additional Windows specific tools.
- Miktex is an easy to install TeX system for Windows. It is based on MiKTeX with a .msi file to guide your installation and additional Windows specific tools.

Figure 8: Rendering the TeX logo in different ways

This module implements Liang’s algorithm (which is also used in TeX). This module has been wrapped in a tag library and used in the general layout definition for all pages. Thus it is possible to insert hyphenation points (\shy;) into the HTML source of the pages on the fly. This procedure makes direct use of the hyphenation patterns contained in hyphen.tex for US English. Now the browser can hyphenate words as TeX would do (see figure 7).

7.2 Skinning and logos

Different people have different opinions about typesetting the name ‘TeX’ — especially on Web pages. Donald Knuth has designed the logo ‘\TeX’ as we know it. In addition he has declared that the text version “TeX” is acceptable. From this inspiration, many more logos have originated: \LaTeX, \LaTeXe, etc.

The CTAN portal supports the drop-character variant as well as the text representation and enables you to select the preferred variant. The portal allows you to select a so-called skin. This skin determines the appearance of the portal. In figure 8 the default skin and the dual skin with text logos are shown.

The skin can be changed via the settings page; several skins are provided. Besides the multi-column skins, plain skins (in dark or light) can be selected. The plain skins omit the use of most fancy formatting
and reduce the appearance to the essentials (see figure 9). These are fine skins for purists. The plain skins also typeset the logos in text form.

![Figure 9: The plain skin](image)

Other skins use different colors or background textures. For instance the skin “sketch” (see figure 10) provides a sketchy look which should appear as a kind of design drawing for the not completely finished layout.

![Figure 10: The sketch skin](image)

At the time of writing, 13 skins are available. Thus it is best to try out the skins yourself. And don’t be surprised if you find some more.

7.3 Responsive Web design

The CTAN portal is based on a design principle known as responsive Web design. This means that the Web pages are tailored towards the (horizontal) space available. If the browser window is wide then the full width is used—to a certain extent. If it is narrow then the width of the page is reduced as well. This respects as much as possible the expressed or implicit expectations of the user.

For the Web, techniques similar to those already known from typesetting are applied: We do not hard-wire dimensions. Instead relative sizes are used whenever possible. For instance this means that widths are given as a percentage of the browser width or in em or ex. These last two adapt an element to the font size chosen by the user.

We can go even further. With CSS3 it is possible to adapt the overall layout of the page. Usually a two-column layout is used if there is enough space. On small devices this reduces to a single column layout. The specification of minimal and maximal widths avoids lines which are too long or too small. Thus readability is improved.

A minor point to mention is the image—the \TeX{} lion—displayed in the upper right corner by default. If space is tight, this image is suppressed.

Thus we have seen that we can pass on several concepts from the typesetting world to the Web in order to improve readability. Even in this radically different technology, the long-known and well-established rules can be applied to improve the user experience.

A snapshot is shown in figures 1 and 11. You can easily see these effects live just by resizing your browser window on your desktop PC, laptop, or smartphone. Take the time and experiment a bit.

![Figure 11: The landing page in a narrow window](image)

8 Visit www.ctan.org

You have seen some of the features of the new CTAN portal. Nevertheless the best way to experience the Web is to use a browser and click your way through it. And so you are invited to try the new experience. Enjoy www.ctan.org and keep on \TeX{}ing.

○ Gerd Neugebauer
  Im Lerchelsböhl 5
  64521 Groß-Gerau, Germany
  gene (at) gerd-neugebauer dot de
  http://www.gerd-neugebauer.de

CTAN: Relaunch of the Web portal
Fonts! Fonts! Fonts!
Bob Tennent

Abstract
Discussion of four new font packages and a revamped font package, with notes on the implementation of the support packages.

1 Introduction
Several new font-support packages (with fonts included) have been installed at CTAN recently and adopted by distributions such as TeX Live and MiKTeX. The primary reason for this outburst is that the Google Web Fonts (GWF) site¹ has provided a focal point for both amateur and professional font developers to distribute liberally-licensed fonts. A second reason is that the freely-available fontforge² font editing software and the ldftypetools and autoinst packages now provide the tools necessary to provide PDF support for new modern fonts relatively easily; this technology has also been used to revamp the widely-used libertine package, which had been abandoned by its original developer.

This article will discuss the following packages:
- quattrocento
- cabin
- librebaskerville
- ebgaramond
- libertine

but it should be noted that there are two other important GWF-derived packages: opensans (supporting the Open Sans family, designed by Steve Matteson of Monotype Imaging) and sourcesanspro (supporting the Source Sans Pro family, designed by Paul D. Hunt of Adobe Systems).

2 Font packages
2.1 Fonts by Pablo Impallari
Pablo Impallari is a young Argentinian typeface designer and font developer. He is a professional but believes in “open-doors” type design, and encourages participation in font development.

2.1.1 Quattrocento and Quattrocento Sans
Impallari describes Quattrocento as a classic, elegant, sober and strong typeface; the wide and open letterforms, and great x-height, make it very legible for body text at small sizes, and the tiny details that only show up at bigger sizes make it also great for display use. Only regular and bold variants are currently available; for now, the quattrocento package activates artificially slanted variants.

Quattrocento Sans is described as warm, readably and not intrusive; it is said to be the perfect sans-serif companion for Quattrocento. It is the main body font at Impallari’s own website.³ Quattrocento Sans currently has regular, bold, italic and bold-italic variants. The quattrocento package activates both of the Quattrocento families by default, but options allow selecting just one of them.

2.1.2 Cabin and Cabin Condensed
CABIN
A Humanist Sans with a Touch of Modernism
Impallari describes Cabin as a humanist sans inspired by Edward Johnston’s and Eric Gill’s typefaces, with a touch of modernism; it incorporates modern proportions, optical adjustments, and some elements of the geometric sans.

Cabin currently has four weights (regular, bold, medium, and semibold) and designed italic variants of all of these; furthermore there are four condensed variants. All of these have designed small capitals.

2.1.3 Libre Baskerville
Libre Baskerville is apparently based on 1941 specimens produced by the American Type Founders Company, but has a taller x height, wider counters and minor contrast to allow it to work at small sizes on any screen.

There is a designed italic and a bold, but currently there is no bold-italic variant; an artificially

¹ http://www.google.com/webfonts
² http://fontforge.org/
³ http://www.impallari.com

Bob Tennent
LIBRE BASKERVILLE


Georg Duffner is a Viennese graduate student of Romance philology. He has begun a project of digitizing fonts by Claude Garamond and Robert Granjon on a famous type specimen issued in 1592 by the Egenolff-Berner foundry in Frankfurt. At present, only regular and italic variants are available, but they include designed small-caps and old-style figures, both tabular and proportional. Also, some swash italics and decorative initials are available.

2.2 Egenolff-Berner Garamond

EGENOLFF-BERNER GARAMOND


3 Implementation notes

3.1 Introduction

Traditionally, font-support packages have relied on fontinst; this package assumes Type 1 (Postscript) font format, which commercially is increasingly considered to be a legacy format. It is possible to use fontforge or other software to convert a TrueType or OpenType font to Type 1 format and re-encode it to, say, Adobe encoding, but incorporating features such as old-style figures or small capitals is a rather painful process, described in full detail in the fontinstallationguide document available at CTAN.

The otftotfm program of the lcdfypetools package will convert an OpenType font to Type 1 format and generate font metrics, virtual fonts, and encoding vectors for use with conventional \LaTeX\ engines, including support for small capitals, old-style figures, titling glyphs, superior figures, swash glyphs, and so on, when these features are provided by the font. And the autoinst script in the fontspec package will process an entire family of fonts using otftotfm, producing also the fd files (in any choice of encodings) needed by \LaTeX.\n
It is true that emerging technologies (Xe\LaTeX, Lua\LaTeX) make it possible for users to access all the features of modern fonts directly, but using radically different font-specification mechanisms provided by the fontspec package. This is not a viable approach for processing legacy documents.

A solution to this dilemma is to implement a support package that, as much as possible, compatibly supports both traditional processing engines (\LaTeX, \pdf\LaTeX) and new technologies based on fontspec. For example, any current \LaTeX\ engine produces the Quattrocento sample of the preceding section from the following input:

\begin{verbatim}
\documentclass{article}
\usepackage{quattrocento}
\begin{document}
\thispagestyle{empty}
\begin{center}\huge Q\,U\,A\,T\,T\,R\,O\,C\,E\,N\,T\,O
\end{center}
\begin{center}\Large A Classic Roman Typeface
\end{center}
\par
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ...
\end{document}
\end{verbatim}

As we shall see, it is relatively straightforward to implement this approach.
For concreteness, we give detailed instructions for re-constructing the quattrocento package. We assume a Unix-like system and that current versions of fontools, lpdfonttools and fontforge (or comparable font-editing software) are available.

3.2 Accessing and converting the fonts
The “source” fonts may be downloaded from the GWF site (or others\footnote{http://www.fontsquirrel.com/}); one should get complete fonts rather than subsets. The fonts distributed for Quattrocento and its Sans counterparts are in TrueType format; to support \texttt{latex} → \texttt{dvips} processing, they should be converted to \texttt{otf} format in fontforge as follows:

File → Generate Fonts → OpenType (CFF) → Save

There are “missing” variants for Quattrocento (no italics); generate an artificially slanted font as a substitute as follows:

Edit → Select → Select All
Element → Style → Oblique → OK

Then change the FontName to, for example,

\texttt{Quattrocento-Italic}

and the “Name For Humans” to

\texttt{Quattrocento\ Italic}

as follows:

Element → Font Info

Finally, set the italic angle as follows:

General → Italic Angle Guess → OK

Then generate the corresponding OpenType font.

Note that some discretion is advisable in generating artificial substitutes. My attempt to produce artificially emboldened variants for \texttt{ebgaramond} was (justifiably) vetoed as undesirable by the designer, whereas artificially slanted or emboldened variants of a \texttt{monospaced} font should be acceptable.

3.3 Generating \TeX\ support files
To generate support files in a \texttt{texmf} tree for Quattrocento, put the relevant \texttt{otf} files in a directory and execute

\begin{verbatim}
autoinst -target=./texmf \
-encoding=OT1,T1,LY1,TS1 \
-vendor=impallari -typeface=quattrocento \n-noupdmap \
*.otf
\end{verbatim}

Then create the directory

\texttt{texmf/fonts/opentype/impallari/quattrocento/}

and move the \texttt{otf} files there. Repeat as above with the \texttt{otf} files for Quattrocento Sans.

3.4 Renaming the encoding files
The \texttt{otftotfm} program generates encoding files with filenames of the form \texttt{a_xxxxx}; to avoid possible filename conflicts with other packages, the files in

\texttt{texmf/fonts/enc/dvips/quattrocento}

should be re-named (use a small script) to have a distinctive prefix, such as \texttt{qtrcnt_}. Then, in the two map files in

\texttt{texmf/fonts/map/dvips/quattrocento}

all instances of \texttt{a_} should be changed to \texttt{qtrcnt_}. The map files may then be merged into a single file, say, \texttt{quattrocento.map}.

3.5 Generating Type 1 fonts
The \texttt{otftotfm} function called by \texttt{autoinst} will use \texttt{cfftot1} to create \texttt{pfb} files with appropriate internal names and filenames, and \texttt{autoinst} will install these in

\texttt{texmf/fonts/type1/impallari/quattrocento/}

but if more than one font family has been processed or if \texttt{cfftot1} runs into trouble with some glyphs, this may not happen. In that case, one must do the conversion font-by-font using either \texttt{cfftot1} or \texttt{fontforge}, which is less sensitive than \texttt{cfftot1} to bad glyph programs. The internal names and filenames must be those specified in the corresponding map file or \texttt{dvips} will fail.

3.6 Editing \TeX\ support files
The \texttt{autoinst} script will generate a large number of files with \texttt{.fd} extensions in the

\texttt{texmf/tex/latex/quattrocento/}

directory. Recent versions of \texttt{autoinst} will generate “silent substitution” rules for mapping \texttt{sl} to \texttt{it} and \texttt{bx} to \texttt{b}; if not, these should be added by hand.

The \texttt{autoinst} script will also have generated a file with \texttt{.sty} extension for each of the font families; however, these do not support direct use of OpenType fonts by X\TeX{} and Lua\TeX{}, and it is necessary to generate a style file suitable for all \TeX{} engines “by hand”. The basic idea is to use traditional settings such as

\begin{verbatim}
\renewcommand*{\rmdefault}{Quattrocento-TLF}
\renewcommand*{\sfdefault}{QuattrocentoSans-TLF}
\end{verbatim}

for Type 1 support, and compatible fontspec settings such as

\begin{verbatim}
\defaultfontfeatures { \
Ligatures=TeX , \
Extension = .otf 
}
\setmainfont{ UprightFont = * , \nItalicFont = *-Italic ,
\end{verbatim}
for OpenType support.

Initially, the choice between Type 1 and OpenType support is determined by the processing engine:

```latex
\ifxetex\quattrocento@otftrue
\else\ifluatex\quattrocento@otftrue
\else\quattrocento@otffalse % [pdf]LaTeX
\fi\fi
```

however, some users of Xe\LaTeX{} or Lua\LaTeX{} may prefer to avoid fontspec, so an option is provided to allow this to be changed:

```latex
\DeclareOptionX{type1}{\quattrocento@otffalse}
```

After all the options have been processed, the choice of settings may be made as follows:

```latex
\ifquattrocento@otf
...
\else
...
\fi
```

The full \texttt{quattrocento.sty} file may be viewed by installing \texttt{quattrocento}, or at CTAN. Here we briefly discuss some issues.

- \texttt{autoinst} generates support files for “superior” (i.e., superscript) figures, but the Quattrocento fonts provide only figures 1, 2 and 3, so the style file should ignore these. See \texttt{ebgaramond.sty} for an example of support of superior, old-style and proportional figures and swash italics.

- The \texttt{\ldots@scale} commands are invoked in the \texttt{fd} files or when specifying fonts with fontspec; but only the scale factor for Quattrocento Sans is adjustable using an option parameter.

- If the \texttt{sfdefault} option has been invoked, \texttt{\let} is used to set \texttt{\familydefault} to the \emph{current} value of \texttt{\sfdefault} (without change to the value of \texttt{\rmdefault}).

- The final step is to remove all \emph{default} font features in fontspec, in case other fonts will be activated by the user.

### 4 Linux Libertine and Biolinum redux

These fonts were fully supported for both traditional and emerging processing engines for some time and have been very popular; however, the \LaTeX{} and pdffonts support used \texttt{fontinst}, and when the original developer abandoned the project and the upstream fonts were updated, it became impractical to maintain the original package.

It has been possible to use \texttt{autoinst} as described above to create a new \texttt{libertine} package which provides reasonable support for traditional engines (including the display and initial fonts), and fairly complete support for emerging engines (including commands to generate arbitrary glyphs). Complete details and notes on the implementation may be found in an Appendix of the package documentation.\textsuperscript{9} The last version of the original \texttt{libertine} package (now called \texttt{libertineotf}, for Lua\LaTeX{} and Xe\LaTeX{} users only) is still available at CTAN.

### 5 Discussion

To conclude, here are links on CTAN to package information for the above fonts and software:

- \url{http://ctan.org/pkg/quattrocento}
- \url{http://ctan.org/pkg/cabin}
- \url{http://ctan.org/pkg/ebgaramond}
- \url{http://ctan.org/pkg/librebaskerville}
- \url{http://ctan.org/pkg/libertine}
- \url{http://ctan.org/pkg/libertineotf}
- \url{http://ctan.org/pkg/fontools (for autoinst)}
- \url{http://ctan.org/pkg/fontspec}
- \url{http://ctan.org/pkg/lcdf-typetools}

I hope the reader will find some of the newly available font packages of interest for their \LaTeX{} documents and may also be inspired to add to the repertoire of \LaTeX{}-ready fonts with support packages suitable for both traditional and emerging processing engines.

### Acknowledgements

I would like to thank Karl Berry, Georg Duffner, Silke Hofstra, Khaled Hosny, Eddie Kohler, Marc Penninga, Michael Sharpe, and Herbert Voss for their assistance and suggestions.

\begin{itemize}
  \item Bob Tennent  
  School of Computing  
  Queen’s University  
  Kingston, K7K3S7  
  Ontario, Canada  
  rdtennon (at) gmail dot com
  \end{itemize}

\textsuperscript{8} \url{http://mirror.ctan.org/fonts/quattrocento/latex/quattrocento.sty}

\textsuperscript{9} \url{http://mirror.ctan.org/fonts/libertine/libertine.pdf}

Fonts! Fonts! Fonts!
1 Font installer

A long time ago I wrote a shell script to install PostScript Type 1 fonts from the old Bitstream 500–font CD. Using a combination of parsing the .afm files, matching the fontname map files, and some low animal cunning, it created .fd, .sty, and .map files, put them and the generated .tfm and .vf files in the ‘right places’ (according to the TDS), and ran updmap to leave the user with an immediately usable typeface. The fontinst package does this too, but I was never able to make it work.

The cdvf script is still available at http://latex.silmaril.ie/fonts/cdvf (it was never robust enough to upload to CTAN), but I had always planned to rewrite it to handle other pairs of .pfb and .tfm files acquired elsewhere.

My university recently got rebranded, as part of which the default text typeface was made Gotham, which I hadn’t had occasion to use before. I therefore created a revised script for them, psfi, optimistically standing for the PostScript Font Installer. This now seems to do the job for any .pfb/.tfm pairs, grouping them into separate faces if required, each with its own .sty and .fd files, which is needed for large font families like Gotham (Figure 1).

What it revealed (which I should have known if I had been paying attention) is that the font-naming scheme which has served well for many years is now running out of space, especially in the Foundry department, hence the x in the fontname. This problem will eventually go away, of course, when everyone switches to using XpdfTEX, but I still have many clients who won’t be making that journey in the foreseeable future.

I’m not a huge fan of sans-serifs for body copy in long documents, although having read some test settings, I found Gotham to be easier on the eyes in quantity than, for example, Futura or Gill Sans (but that may just be my ageing eyes). There don’t appear to be any math fonts in Gotham, however, so there will either have to be some trickery done with faking it in MathDesign with something that looks similar, or the standard will need a variant to allow TEXicnal TExTs to be set in a serif face.

This time I will upload the script to CTAN, in the hope that others can test it and show me what goes wrong. Look for it at a server near you soon.

2 Class and package creation

The one thing that psfi doesn’t do is wrap the whole result up as a .dtx/.ins pair of files for redistribution, although technically it could do that for everything except the proprietary font files themselves. But that will have to wait.

What has not been able to wait is some work I had to do on some XML-based software for assisting the creation of class and package files for clients. This is still an in-house development, although I used it for the decorule package, referred to it in an aside in my last column, and am hoping to be able to show some of it at a suitable TUG or Balisage meeting.

Among the typographic tweaks are some extensions using the dox package for marginal tags for more than just macros and environments; a by-product of the preprocessing is that the margin can be dynamically re-set for the relevant chapters, to accommodate the widest tag referenced, which improves the usability. Whether a margin-change between the user documentation chapter and the code documentation chapter is good typography is moot, although using a narrow font variant for the tags minimises the disruption.

3 Grids

One of the packages was for a client whose designer used the traditional grid system for the document layout. Coercing LATEX into a rigid grid isn’t particularly easy, as the underlying convention is that every block object on the page is separated from its neighbours by rubber space. This drives designers into screaming fits; when they wake sweating at 3am, it’s LATEX’s breaches of the grid they were having a nightmare about.

In practice, tables, figures, sidebars, and display math rarely come in nice neat multiples of the baseline height, even when everything else (the title page, headings, paragraphs, lists, etc.) can be re-spaced to do so (Figure 2).

One of the methods is to set the object into a box, measure its height and depth, and then add white-space in increments to make it an integer number of baselines high. This works, at the expense of some extra cycles (usually irrelevant on modern machines); and the same principle can be used for any repetitive layout where the baselines of certain objects have to be multiples of a specific depth from the top of the page or from each other.

However, what turned out to be more difficult was getting subsection headings to have no extra vertical white-space below them, before the first line of the paragraph. This is no problem when \parskip is zero, despite \@startsection placing a \par after
the heading, because you can set the relevant argument of \@startsection to 1sp (it cannot be zero or negative because that is used to specify run-in headings). But when the layout requires a non-zero paragraph space, done in this case with the \parskip package, that amount gets added by the \par after a heading. One solution — a kludge — is to add an embedded negative skip at the end of the heading, which would drag the following paragraph up; and then to add code to \@dottedtocline so that the negative skip in the heading did not affect the Table of Contents. Running heads would not normally occur at the subsection level, otherwise the same method would have been needed there.

### 4 Business cards

I haven’t used very many of these but I do have a few for meetings with people who need them. Some cultures use them more than others: I am told they are de rigeur in Japan and virtually non-existent in Silicon Valley. Graduate students don’t immediately spring to mind as a market, but in fact they’re rather important. These are people who meet funding agencies, attend conferences where they need to get their name and their institution known, and do business with authors and publishers who may need reminding of the person and the field.

But they’re far too expensive for a department to have them professionally printed for every student,
5 Running ragged

Ragged-right setting is commonplace for many applications, often on aesthetic grounds because the column width is narrow, or because the line-spacing is tight, or because the document is informal (and probably lots of other reasons, too). One place where it has a specific practical use is in bibliographies, and yet I constantly see the References in articles and longer documents set justified because that’s the default. You can be lucky: if all the names and words in your entries are short, the H&J engine won’t have any problem, even in narrow columns [2].

However, the majority of technical documents with references would seem naturally to involve specialist words and phrases that even \TeX’s hyphenation algorithm may not handle, or long URIs that don’t linebreak easily [1]. In this case, as below, you may get the author and title justified, but have trouble with very long URIs. I see no reason why bibliographies shouldn’t be set ragged-right, especially where the measure is narrow: it certainly looks better than having a mix of justified and ragged. (Bibliographies in TUGboat often use ragged-right for precisely these reasons; the present bibliography is left as the default for purposes of example.)

References


Ο Peter Flynn
Textual Therapy Division, Simlaril Consultants
Cork, Ireland
Phone: +353 86 824 5333
peter (at) simlaril dot ie
http://blogs.simlaril.ie/peter
Entry-level MetaPost: On the grid
Mari Voipio

1 Running MetaPost
The basic method for running MetaPost is to create a file with extension .mp and run MetaPost directly on the command line: mpost yourfile.mp. In the .mp file, the code for making graphics is enclosed within beginfig ... endfig; the result is a PostScript file. The file extension is the figure number. With this example, the output would be a file yourfile.1.

beginfig (1);
% Draw a line
draw (1cm,2cm) -- (3cm,5cm);
endfig;
end.

If you add outputformat:="svg"; at the very beginning of the file, the output will be in SVG format. However, the resulting file will still be named yourfile.1; depending on your system, you may have to rename it to yourfile.svg for it to be recognized.

Another method, available in any decent TeX distribution, is to run MetaPost inside a ConTeXt file. (I am a ConTeXt user, so I find it handiest to do it this way.) The file extension is the usual .tex, and the MetaPost code (the same MetaPost code as when running standalone) is enclosed inside \startMPpage ... \stopMPpage. I run my files within an editor with a menu command Typeset, but you can also typeset the file on the command line with context yourfile.tex.

\startMPpage
% Draw a line
draw (1cm,2cm) -- (3cm,5cm);
\stopMPpage

The output from ConTeXt is (by default) a PDF file, yourfile.pdf. Graphics code can be included in any .tex file; you can have text and MP code in one and the same ConTeXt file.

2 Building up a graphic
To draw anything with MetaPost, you have to have some kind of idea of where your drawings end up and why. Everything is related to a base grid; it doesn’t usually show, but in our examples here I’ve put a grid in the background to help you figure out how to place objects in MetaPost. (A blank grid is shown in fig. 1. We’ll see how to draw the grid itself later.)

The MP grid extends to all directions, e.g., coordinates (1,2), (-1,2), (-1,-2) and (1,-2) all exist. However, I find it easiest to stay on the positive side when coding a drawing or an element; objects can be easily shifted later to their proper locations.

The basic unit of MetaPost is a “big point” (bp), and that one is tiny (72 bp = 1 in). There are several ways of changing the scale: in the examples below we’ll use a user-defined unit u. That way we only have to change u if we want to adapt the drawing to a different size/scale. Here, we’ll define the unit u at the beginning of the MetaPost file to fit TUGboat:
numeric u ; u := .5cm ;

As a first example, let’s draw a small red dot — a filled circle — at the origin. The output is in fig. 2. (If we had nothing else in the picture, we’d get a red dot in the middle of the page with some white around and that’s it; our dot here isn’t in the middle because of the grid in the background.)

% Set the scale
numeric u ; u := .5cm ;

% Draw a red dot at origin
fill fullcircle scaled (1/5u) withcolor red ;

% Add whitespace around the drawing
Next, let’s add another dot in blue at $(3,5)$:

```latex
fill fullcircle scaled (1/5u)
    shifted (3u,5u) withcolor blue;
```

This command goes after the previous fill, above the setbounds command (as do our subsequent additions). Output in fig. 3.

We don’t have to use whole numbers; MetaPost is just as handy with decimals and fractions (fig. 4):

```latex
fill fullcircle scaled (1/5u)
    shifted ((4u+1/8u),3.25u) withcolor green;
```

When something is repeated at regular intervals, a loop can be used (fig. 5):

```latex
for i = 6u step 1/2u until 10u :
    draw fullcircle scaled (1/5u) shifted (8u,i) withcolor red;
endfor;
```

We can change both coordinates inside one loop:

```latex
for i = 0 step 1/2u until 3u :
    draw fullcircle scaled (1/5u)
        shifted (2*i,(-i+9u)) withcolor green;
endfor;
```

Lines and objects are drawn by connecting two or more sets of coordinates. Either a straight line (specified with `--`) or a curved line `(..)` is drawn between the given coordinates. A line is turned into a closed object with the final command `cycle`. Only a closed object can be filled with colour! (See fig. 7.)

```latex
fill (5u,1u) -- (7u,5u) -- (7.5u,5u) .. (8u,3u) .. (9u,3u) .. (7u,1/2u) .. cycle;
```

In the next picture (fig. 8) these curve coordinates have been marked with small red dots. If you are not familiar with Bezier curves, the behaviour of the curved lines can be surprising.

A line or an object always has a bounding box, a rectangular frame. MetaPost knows where the corners of the bounding box are; in fig. 9, the box is drawn as a line from corner to corner on the black object (red dots are added later) and the dark blue

Mari Voipio
dot is placed on the lower-right corner (lrcorner) of the black object.

All of the code we’ve built up follows, and the output without the background grid is in fig. 10.

```plaintext
% -- begin full example --
numeric u ; u := .5cm ;
fill fullcircle scaled (1/5u) withcolor red ;
fill fullcircle scaled (1/5u) shifted (3u,5u) withcolor blue ;
fill fullcircle scaled (1/5u) shifted ((4u+1/8u),3.25u) withcolor green ;
for i = 6u step 1/2u until 10u :
  draw fullcircle scaled (1/5u) shifted (8u,i) withcolor red ;
endfor ;
for i = 0 step 1/2u until 3u :
  draw fullcircle scaled (1/5u) shifted (2*i,(-i+9u)) withcolor green ;
endfor ;
% Define the black object
picture curvy ;
curvy := image (fill (5u,1u) -- (7u,5u) -- (7.5u,5u).. (8u,3u) .. (9u,3u) .. (7u,1/2u).. cycle ;);
% Draw the black object
draw curvy ;
% Draw bounding box around the object
draw llcorner curvy -- lrcorner curvy -- urcorner curvy -- ulcorner curvy -- cycle withcolor blue;
% Add dot at lrcorner of object bounding box
fill fullcircle scaled (1/5u) shifted (lrcorner curvy) withcolor 0.5blue;
% Add red dots at object ‘turning points’
fill fullcircle scaled (1/10u) shifted (5u,1u) withcolor red;
fill fullcircle scaled (1/10u) shifted (7u,5u) withcolor red;
fill fullcircle scaled (1/10u) shifted (7.5u,5u) withcolor red;
fill fullcircle scaled (1/10u) shifted (8u,3u) withcolor red;
fill fullcircle scaled (1/10u) shifted (9u,3u) withcolor red;
fill fullcircle scaled (1/10u) shifted (7u,1/2u) withcolor red;
% Add whitespace
setbounds currentpicture
```

Entry-level MetaPost: On the grid
2.1 Troubleshooting

With MetaPost — as well as with T\TeX — I’ve found that it is a good idea to compile (typeset) the file fairly frequently, especially when doing something for the first time. I carefully “grow” my MetaPost graphics stage by stage, very much like the examples above. If my file suddenly stops working, I comment out the newest lines (with \% and recompile to check that I haven’t, say, accidentally deleted a semicolon in the older code, and then proceed to debug the newest code.

If your file doesn’t compile, the first thing to look for is an omission: a missing semicolon at the end of a command, a missing \endfig/\stopMPpage, missing parentheses. If the semicolon on the last line is missing, the file will compile — but only until something is added after that line, then it fails.

If you draw something with the fill command, the item to fill must be a closed path, either a line/curve closed with cycle, or a circle or square, e.g. fullcircle. If you try to fill a non-cyclical path, your file won’t compile because MetaPost hangs at the “impossible” command.

If your drawing is tiny and needs lots of zooming to be visible, you may be using MetaPost’s default unit, the big point. Try the above-mentioned trick of defining a unit \u to suit your taste, or use units with your coordinates. In the beginning I found it easiest to think on a millimeter grid, so I defined lines with, e.g., draw (8mm,20mm) -- (10mm,40mm);

MetaPost draws items in the order they are defined. If your file compiles, but you only get some of the objects, the missing ones may be underneath everything else. Our examples above are carefully spread out so they don’t overlap, but if you play with the coordinates, you may encounter this feature. If you draw several items at the same spot, the first (highest up) in the code is at the bottom in the drawing and the last in the file is on top in the final graphic; it may help to think of the lines and objects as pieces of paper piled up on top of each other, so the last piece ends up on top of everything else.

2.2 Drawing the background grid

I used Metafun to draw the gray grid in the illustrations above. Metafun comes with Con\TeXt, so if you have Con\TeXt, your \TeX installation already includes it. Besides a plain grid, Metafun also supports creating slanted or logarithmic grids (see pages 213–214 in the Metafun manual).

First we define the grid, then draw it:

\begin{verbatim}
picture grid ;
grid := image (width := 10 ; height := 10 ; draw vlingrid (0, 10, 1, width, height) withcolor .8white ; draw hlingrid (0, 10, 1, height, width) withcolor .8white ;)

draw grid withpen pencircle scaled 1/20 ;
\end{verbatim}

I put the grid code in the beginning of my MetaPost file(s), and thus it gets drawn first and ends up in the background, behind all other elements. The numbers are standard labels, set with the label ("\texttt{\textcolor{+red}{\texttt{\color{red}labelname}}, (x,y)) command. For example: label ("1", (-1/4\u,1\u)) ;

2.3 Further reading


MetaPost, a very brief tutorial: http://www.ursoswald.ch/metapost/tutorial.html. A more traditional approach to MetaPost. Click on filenames to get the code!

MetaPost home page: http://www.tug.org/metapost. Lots of links to other tutorials, examples, articles, packages, and more.

\begin{itemize}
\item Mari Voipio
\end{itemize}

Mari Voipio

mari dot voipio (at) lucet dot fi
http://www.lucet.fi
Recreating historical patterns with MetaPost
Mari Voipio

I’ve always been interested in history, and fascinated by repeated geometric patterns. This combination of interests has led me to try to recreate patterns from various historical periods for use in crafts, e.g. card making and quilting. Other patterns presented here were drawn just because they intrigued me and looked like a good way of practicing with MetaPost. I tend to prefer older European history, but the same motifs recur in later periods, so many look familiar — even modern — like the Greco-Roman pelta.

1 The Greek key
The Greek key and its variations have a multitude of names: meander, scroll, running dog, … The Greek key appears all over the Classical world, being very easy to repeat e.g. in mosaic. The basic square or rectangular form is also easy to recreate with MetaPost and allows for play with borders and positive/negative images.

```
% define one unit
picture gkey;
gkey := image (
    linecap := squared ; % crisp triangle corners
    linejoin := mitered ; % neat line ends
    draw (0,0) -- (0,5) -- (5,5) -- (5,1) -- (2,1)
       -- (2,3) -- (3,3) -- (3,2) -- (4,2)
       -- (4,4) -- (1,4) -- (1,0) -- (6,0);
);

% repeat to create a border
for i = 0 step 6 until 24 :
    draw gkey shifted (i,0) ;
endfor ;

% set final size of the meander
currentpicture := currentpicture xsized 6cm ;
% add .5 cm whitespace all over
setbounds currentpicture to boundingbox currentpicture enlarged 1/2cm ;

This “reversed” image draws the meander in white on a black background, the latter made with:
```
addbackground withcolor black ;
```

2 Roman mosaic floors
The Roman villas were richly decorated with paintings and mosaic floors — not only in Italy, but anywhere the Romans went, including the British Isles. The mosaics are laid with small cubes, tesserae, that are cut to points when needed. However, I’m more interested in the patterns that form when the mosaic has been laid. My first attempt at recreating mosaic patterns was a monochromatic floor border found in an older layer (1st century) at Fishbourne Roman villa in the south of England.

```
% define one unit
picture gkey;
gkey := image (
    linecap := squared ; % crisp triangle corners
    linejoin := mitered ; % neat line ends
    draw (0,0) -- (0,5) -- (5,5) -- (5,1) -- (2,1)
       -- (2,3) -- (3,3) -- (3,2) -- (4,2)
       -- (4,4) -- (1,4) -- (1,0) -- (6,0);
);

% repeat to create a border
for i = 0 step 6 until 24 :
    draw gkey shifted (i,0) ;
endfor ;

% set final size of the meander
currentpicture := currentpicture xsized 6cm ;
% add .5 cm whitespace all over
setbounds currentpicture to boundingbox currentpicture enlarged 1/2cm ;

This “reversed” image draws the meander in white on a black background, the latter made with:
```
addbackground withcolor black ;
```

% define one unit
picture gkey;
gkey := image (
    linecap := squared ; % crisp triangle corners
    linejoin := mitered ; % neat line ends
    draw (0,0) -- (0,5) -- (5,5) -- (5,1) -- (2,1)
       -- (2,3) -- (3,3) -- (3,2) -- (4,2)
       -- (4,4) -- (1,4) -- (1,0) -- (6,0);
);

% repeat to create a border
for i = 0 step 6 until 24 :
    draw gkey shifted (i,0) ;
endfor ;

% set final size of the meander
currentpicture := currentpicture xsized 6cm ;
% add .5 cm whitespace all over
setbounds currentpicture to boundingbox currentpicture enlarged 1/2cm ;

This “reversed” image draws the meander in white on a black background, the latter made with:
```
addbackground withcolor black ;
```

% define one unit
picture gkey;
gkey := image (
    linecap := squared ; % crisp triangle corners
    linejoin := mitered ; % neat line ends
    draw (0,0) -- (0,5) -- (5,5) -- (5,1) -- (2,1)
       -- (2,3) -- (3,3) -- (3,2) -- (4,2)
       -- (4,4) -- (1,4) -- (1,0) -- (6,0);
);

% repeat to create a border
for i = 0 step 6 until 24 :
    draw gkey shifted (i,0) ;
endfor ;

% set final size of the meander
currentpicture := currentpicture xsized 6cm ;
% add .5 cm whitespace all over
setbounds currentpicture to boundingbox currentpicture enlarged 1/2cm ;

This “reversed” image draws the meander in white on a black background, the latter made with:
```
addbackground withcolor black ;
```

% define one unit
picture gkey;
gkey := image (
    linecap := squared ; % crisp triangle corners
    linejoin := mitered ; % neat line ends
    draw (0,0) -- (0,5) -- (5,5) -- (5,1) -- (2,1)
       -- (2,3) -- (3,3) -- (3,2) -- (4,2)
       -- (4,4) -- (1,4) -- (1,0) -- (6,0);
);

% repeat to create a border
for i = 0 step 6 until 24 :
    draw gkey shifted (i,0) ;
endfor ;

% set final size of the meander
currentpicture := currentpicture xsized 6cm ;
% add .5 cm whitespace all over
setbounds currentpicture to boundingbox currentpicture enlarged 1/2cm ;

This “reversed” image draws the meander in white on a black background, the latter made with:
```
addbackground withcolor black ;
```
% tiling the quarter blocks into a wheel
picture wheel;
wheel := image (
  draw whmosaic;
  draw whmosaic rotatedaround ((1,1),90);
  draw whmosaic rotatedaround ((1,1),180);
  draw whmosaic rotatedaround ((1,1),270);
);

% drawing the wheel pattern
draw wheel;

% one way to add a frame around the wheel
setbounds currentpicture to boundingbox currentpicture enlarged 1/10;
draw boundingbox currentpicture
  withpen pencircle scaled 1/5;

% defining a single pelta
picture pelta;
pelta := image (  
  linejoin := mitered;  
  linecap := squared;  
  filldraw (0,0) .. (2,2) .. (4,0) -- (4,0)  
    .. (3,1) .. (2,0) -- (2,0) .. (1,1)  
    .. (0,0) -- cycle  
    withpen pencircle scaled 1/50;   ) ;

% defining a block of two
% square+pelta combinations
picture flooring;
flooring := image (  
  draw pelta shifted (-1,1);  
  draw pelta rotated 90 shifted (-1,-1);  
  draw pelta rotated 180 shifted (1,-1);  
  draw pelta rotated 270 shifted (1,1);  
  fill fullsquare scaled 2;  
  draw pelta shifted (3,1);  
  draw pelta rotated 90 shifted (5,-3);  
  draw pelta rotated 180 shifted (9,-1);  
  draw pelta rotated 270 shifted (7,3);  
  fill fullsquare scaled 2 shifted (6,0);   ) ;

draw flooring reflectedabout  
  ((ulcorner flooring),(urcorner flooring));

% rows 1 and 3
for i = 0 step 12 until 18 :  
  for j = 0 step 12 until 18 :  
    draw flooring shifted (i,j) ;  
  endfor ;
endfor ;

% rows 2 and 4
for i = 0 step 12 until 18 :  
  for j = 0 step 12 until 18 :  
    draw flooring reflectedabout  
      ((ulcorner flooring),(urcorner flooring))  
      shifted (i,j);  
  endfor ;
endfor ;

% adding the borderline around the flooring
% (as in the original floor)
setbounds currentpicture to boundingbox currentpicture enlarged 1/10;
draw boundingbox currentpicture
  withpen pencircle scaled 1/5;

% adding whitespace around the picture
setbounds currentpicture to boundingbox currentpicture enlarged 1/2;

% defining the size of final pdf graphic
% (whitespace and all)
currentpicture := currentpicture xsized 6.6cm;

The *pelta*—shield shape—is a frequently repeated pattern in early art, even on floor mosaics, despite its curvy nature being a bit of a challenge when laying the mosaic. I, too, had to fight with the pattern for a while before I realized that my curves behave better if I add a nil-length straight line between the curves.

The guest section of Hadrian’s villa in Tivoli, Italy, has a floor patterned with mirrored and rotated pelta and squares (Field, 1988, p. 62). The constituent parts are simple to code, but the combined floor pattern turned out to be harder to achieve than I first thought. Somehow the rotated and shifted and repeated curves seemed to mislead my eyes, and only after several tries did my graphic match the floor.

Mari Voipio
3 Celtic patterns

My first Celtic MetaPost pattern was Solomon's knot:

Intricate knotwork patterns seem to have been popular all over northern Europe during the Migration era, but we know them especially well from 9th century manuscripts like the Book of Kells. These patterns, as well as spirals and key patterns, are used both for page borders and to fill whole pages in these opulent manuscripts.

I started to draw Celtic patterns with MetaPost partly to create borders and headers for e.g. greeting cards and partly because I'm intrigued by all types of braiding structures. I'm slowly trying to create a set of "building blocks" that can be repeated to create frames and page backgrounds.

3.1 Celtic keys

I very much wanted to make a Celtic key pattern of the type that was used for fillings. It is a deceptively simple design with lines and a few filled shapes (Sloss, 1997, p. 36). First I planned to create the pattern as I was drawing with a pen, doing whole lines, but then I finally divided the pattern in five rectangular blocks — two corners, two sides, one middle — that can be used to build a larger key pattern picture. As a result of this approach, some lines consist of several smaller lines, but that doesn't show unless you start editing the graphic. Perhaps not the neatest possible solution, but it makes "growing" the pattern fairly straightforward.

% defining line width
numeric w ; w := 3/4 ;
linecap := squared ;
linejoin := mitered ;

% bottom left corner, size 4x4
picture blcorner ;
blcorner := image (filldraw (0,0) -- (0,4) -- (4,0) -- cycle withpen pencircle scaled w; draw (1,1) -- (4,4) withpen pencircle scaled w;);

% top right corner, size 4x4
picture trcorner ;
trcorner := image (draw blcorner rotated 180 ; ) ;

% horizontal side element, size 8x4
picture hside ;
hside := image (filldraw (0,0) -- (1,1) -- (2,0) -- cycle withpen pencircle scaled w; draw (0,2) -- (1,3) -- (5,-1) withpen pencircle scaled w; draw (2,4) -- (6,0) withpen pencircle scaled w; draw (5,1) -- (8,4) withpen pencircle scaled w; filldraw (6,0) -- (8,0) -- (6,2) -- (5,1) -- cycle withpen pencircle scaled w; draw (5,3) -- (6,4) withpen pencircle scaled w;);

% vertical side element, size 4x8
picture vside ;
vside := image (filldraw (0,0) -- (0,2) -- (1,1) -- cycle withpen pencircle scaled w; draw (-1,5) -- (3,1) withpen pencircle scaled w; draw (2,0) -- (4,2) withpen pencircle scaled w; draw (3,3) -- (2,4) -- (4,6) withpen pencircle scaled w; filldraw (0,6) -- (1,5) -- (2,6) -- (0,8) -- cycle withpen pencircle scaled w; draw (1,5) -- (4,8) withpen pencircle scaled w;)

% top left corner, size 4x4
picture tlcorner ;
tlcorner := image (filldraw (0,0) -- (1,1) -- (0,2) -- cycle withpen pencircle scaled w; filldraw (2,4) -- (3,3) -- (4,4) -- cycle withpen pencircle scaled w;)

Recreating historical patterns with MetaPost
Months after creating Solomon’s knot I decided to go back to the Celtic knotwork patterns, this time with a design that is known today as a Josephine knot (taken from Meehan, 2007, p. 199). You may see it as consisting of two loops, but careful analysis (and one false start) made me realize that it in fact boils down to four elements: the end, the middle, a shorter curve, a longer curve. The middle is the easiest, a straight line that isn’t rotated, just shifted; the other three elements sometimes need to be rotated 180 degrees to fit.

Just as with the mosaic floor, I defined each knot element separately, then shifted the original elements into correct positions in the knot. After that I created the rotated elements one at a time and shifted each into position. When the picture turned into a mess of white lines because I’d shifted something in a wrong direction, I found that colouring the newest (top) element with another colour helped me see what needed to go where in relation to the elements I’d placed earlier.

The Celtic scribes used a strict system of grid points, divisions and sub-divisions to work out their knots. I stuck to their idea of dividing areas by 4, 16 and 64, but did the final placements by trial and error rather than by doing the math. Clearly I’ll have to work out the rules before I can start expanding the knot e.g. by adding more curves and middles between the ends, but this simple knot was fairly easy to work out with estimates partly based on my earlier experience.

Mari Voipio
I’m indebted to Hans Hagen, both for the patience with my very basic questions and for the invaluable snippets of code that have pushed me into new paths of thinking.

4.1 Literature


4.2 Online resources

Greek keys: http://gydir.demon.co.uk/joy/greekkey/


The mosaic floors at *Hospitalia* at Hadrian’s Villa: http://commons.wikimedia.org/wiki/File:Pavement_Hospitalia_Villa_Hadriana_n2.jpg http://commons.wikimedia.org/wiki/Villa_Adriana#Hospitalia_and_Imperial_Triclinium


More on my Celtic knots: http://www.lucet.fi/craftex/celtic-metapost

diamond Mari Voipio
mari dot voipio (at) lucet dot fi
http://www.lucet.fi

Recreating historical patterns with MetaPost
The xpicture package

Robert Fuster

Abstract

The xpicture package extends the graphic abilities of the standard \LaTeX\ environment \texttt{picture} and the packages \texttt{pict2e} and \texttt{curve2e}, adding the ability to work with arbitrary reference systems, Cartesian or polar coordinates. Furthermore, in addition to drawing lines, vectors, polygons and polylines, this package allows you to draw conic sections and arcs, graphs of functions and parametrically defined curves. These curves are composed by using quadratic Bézier approximations automatically determined using the \texttt{calculator} and \texttt{calculus} packages.

1 Introduction

Since Leslie Lamport first included the possibility of composing pictures in \LaTeX\ the capabilities in this area have increased in several ways, as you can see by taking a look at \textit{The \LaTeX\ Graphics Companion} (Goossens, Mittelbach, Rahtz, Roegel, and Voß, 2008) or at related sections on CTAN.¹

Among these many choices, those exploiting the graphical capacity of PostScript (or PDF) have proved to be the most productive. In this area, we find different solutions.² This is what the package \texttt{pict2e} (Gäßlein, Nieprasch, and Tkadlec, 2011) does, solving the limitations in the standard environment \texttt{picture} by abandoning the technique of building pictures using special fonts, instead adopting a driver-oriented technique. The \texttt{curve2e} package (Beccari, 2012) redefines some \texttt{pict2e} commands and introduces more drawing facilities that allow drawing circular arcs and other curves.

The present \texttt{xpicture} package (Fuster, 2012b) retains the coordinate-oriented approach of \texttt{picture} while adding several features, the most important of which are the ability to work with various reference systems and to draw curves (including real functions) using their parametric equations. This package requires several packages: \texttt{curve2e}, \texttt{xcolor} (Kern, 2007) to handle colors, and \texttt{calculator} and \texttt{calculus} (Fuster, 2012a), to define functions and perform calculations.

We present the \texttt{xpicture} package in section 2, briefly describing its main features. In section 3 we discuss how some features of the package have been implemented, particularly the change of coordinate system and the drawing of curves. Finally, some conclusions and ideas about future work will be described in section 4.

2 The \texttt{xpicture} package

The \texttt{xpicture} package introduces several new graphical instructions, and some enriched versions of standard features provided by the \texttt{picture} environment. All these new instructions can be classified as follows:

(a) Declaration and use of different reference systems, using both Cartesian and polar coordinates.

(b) The \texttt{Picture} environment, an alternative to the \texttt{picture} environment, compatible with the new reference systems.

(c) Instructions to show Cartesian or polar frames and grids.

(d) Alternative instructions or extensions of the standard \texttt{picture} commands and those defined by the packages \texttt{pict2e} and \texttt{curve2e}:

- Enriched versions of the commands \texttt{\put} and \texttt{\multiput}, providing adequate control of the precise position in which objects are composed.
- Instructions for drawing straight segments, vectors (in any direction and using any reference system), polygonal lines, and regular or arbitrary polygons.

(e) Regular curves:

- Instructions for drawing conic sections (i.e., circles, ellipses, hyperbolas and parabolas) and arcs of these curves.
- Instructions for graphing functions and parametrically defined curves.

In the following subsections we will describe some of these features. Many of the examples we will show below are incomplete and only show the instructions that these examples try to describe. For example, although the drawings should be included in a \texttt{Picture} environment and they require the \texttt{\unitlength} length, in some examples this is not explicitly shown.

2.1 Reference systems and coordinates

By a reference system we mean an affine reference system, i.e., a point \(O(a,b)\) (the origin) and two linearly independent vectors, \(\mathbf{u} = (u_1, u_2)\) and \(\mathbf{v} = (v_1, v_2)\). If \((\bar{x}_1, \bar{x}_2)\) are the coordinates of a point \(P\) with respect to this reference system, then the standard coordinates of \(P\) are \((x_1, x_2)\), where

\[
\begin{bmatrix}
  x_1 \\
  x_2
\end{bmatrix} = \begin{bmatrix}
  u_1 & v_1 \\
  u_2 & v_2
\end{bmatrix}^{-1} \begin{bmatrix}
  \bar{x}_1 \\
  \bar{x}_2
\end{bmatrix}
\]
The \texttt{xpicture} package includes several new declarations to change the reference system. Our first example shows the geometric transformation produced by the declaration
\begin{verbatim}
\referencesystem(3,3)(1,0)(2,-2)
\end{verbatim}
Here, (3, 3) is the new origin of coordinates and the new coordinate vectors are (1, 0) and (2, -2).

\newcommand{\mypentagon}{...}
\begin{Picture}(-2,-2)(6,6)
\mypentagon
\referencesystem(3,3)(1,0)(2,-2)
\mypentagon
\end{Picture}

Using multiple reference systems is one of the strongest features of \texttt{xpicture}. Among other applications, the choice of the reference system allows you to display the graphic effect of several geometric transformations, using different scales in each of the coordinate axes, display inverse functions, etc.

Points can be referred to by their Cartesian or polar coordinates (always with respect to the active reference system). In addition, angles can be measured in both radians and degrees.

2.2 The new \texttt{Picture} environment

The \texttt{xpicture} package does not change the behavior of standard \LaTeX{} commands for drawing (more precisely, it does not change the behavior of these commands as packages \texttt{pict2e} and \texttt{curve2e} have redefined them). Instead, it introduces new commands and environments, with a syntax similar to the standard ones. In particular, the new environment \texttt{Picture} (or \texttt{xpicture}) is used as an alternative to the standard \texttt{picture} environment. By using
\begin{verbatim}
\begin{Picture}[(\textcolor{color})\{\langle x0,y0\rangle\}]{\langle x1,y1\rangle}
\end{verbatim}
we fix the drawing area $\langle x0,y0\rangle \times \langle x1,y1\rangle$, referred to as the active reference system. More precisely, this environment defines a \texttt{picture} box that circumscribes this drawing area. If the optional argument is used, the background is colored in the given \texttt{color}.

\newcommand{\mypentagon}{...}
\begin{Picture}(-2,-2)(6,6)
\mypentagon
\referencesystem(3,3)(1,0)(2,-2)
\mypentagon
\end{Picture}

2.3 Showing coordinate systems

Cartesian and polar axes and grids can be easily drawn, with widely customizable lines, cuts and labels, as shown in the following examples.

\begin{verbatim}
\begin{Picture}(-2,-2)(6,6)
\cartesiangrid(-2,-2)(2,2)
\end{Picture}
\begin{Picture}(-2,-2)(6,6)
\cartesianaxes(-2,-2)(2,2)
\end{Picture}
\begin{Picture}(-2,-2)(6,6)
\externalaxes
\renewcommand{\ticsize}{3pt}
\renewcommand{\secundaryticsize}{1.5pt}
\renewcommand{\axeslabelsize}{\scriptsize}
\renewcommand{\axeslabelmathversion}{bold}
\renewcommand{\xunitdivisions}{3}
\renewcommand{\yunitdivisions}{2}
\cartesianaxes(-2,-2)(2,2)
\end{Picture}
\begin{Picture}(-2,-2)(6,6)
\cartesianaxes(-2,-2)(2,2)
\end{Picture}
\end{verbatim}
2.4 Extensions of standard picture commands

Despite the special syntax of \makebox inside a picture environment, the precise positioning of objects that are not strictly graphics (for example, labels or formulas) is a bit complicated and often depends on the \unitlength value.

The \texttt{xpicture} package defines several new commands, extending \put and \multiput in several ways. First, coordinates refer to the active reference system. Second, the precise position of that object with respect to the reference point is fixed by a new argument supporting multiple values: a number (interpreted as an angle with respect to the reference point), some of the keys \texttt{c, t, b, br, rtr, ...} (to similar effect as those keys in other commands), or compass keys \texttt{N, S, SE, ENE, ...}. In addition, Cartesian and polar coordinates are allowed.

In the following example, the label \((x, \sin x)\) was placed at ENE of \((3\pi/4, \sin 3\pi/4)\) (these numbers were computed with the aid of the \texttt{calculator} package).
Commands to plot lines, vectors, polylines and polygons (including regular polygons) are provided.

\begin{Picture}(-4.5,-4.5)(4,4.25)
\externalaxes
\cartesianaxes(-4,-4)(4,4)
\thicklines
\xLINE(0,-4)(-4,0)
\xVECTOR(-2,-2)(-2,3)
\arrowsize{10}{4}
\xtrivVECTOR(-2,-2)(-1,-1)
\Put(1,1){\regularPolygon{2}{10}}
\Put(1,1){\regularPolygon{2}{5}}
\Put(1,1){\polarreference\degreesangles\Polygon(2,0)(2,144)(2,288)(2,432)(2,576)}
\end{Picture}

\setlength{\unitlength}{0.5cm}
\renewcommand{\axeslabels}{\scriptsize}
\begin{Picture}(-5.5,-4.5)(5.5,4.5)
\externalaxes
\cartesianaxes(-5,-4)(5,4)
\rotateaxes{\numberQUARTERPI}
\Ellipse{4}{3}
\Polyline(0,3)(0,0)(4,0)
\end{Picture}

Parabolas and hyperbolas, being unbounded curves, require two additional parameters, \langle xmax \rangle and \langle ymax \rangle, in order to delimit the portion of the curve to be drawn. Moreover, in the case of hyperbolas, you can draw either just one or both branches.

\Hyperbola{5}{2}{16}{8}
\xLINE(16,6.4)(-16,-6.4)
\xLINE(-16,6.4)(16,-6.4)

Commands for drawing arcs of all kinds of conic sections are also provided.

2.5 Regular curves

Using the \texttt{xpicture} package piecewise regular curves can be easily drawn. In a general way, you can draw any two-dimensional curve that can be described with parametric equations.

To simplify common needs, the package includes specific instructions to draw graphs of functions, polar curves and (arcs of) conic sections.

2.5.1 Conic sections and arcs

The \texttt{\Circle{\langle r \rangle}} and \texttt{\Ellipse{\langle a \rangle}\{\langle b \rangle\}} commands draw the circle of radius \langle r \rangle and the ellipse of semi-axes \langle a \rangle and \langle b \rangle, respectively.

The \texttt{xpicture} package
Then, the instruction
\PlotFunction[30]{\Ffunction}{-10}{10}
does all the work to draw the curve.³

\begin{Picture}(-11,-11)(11,11)
\makenotics\makenolabels\cartesianaxes(-10,-10)(10,10)
\printxticlabel{\numberPI}{\pi}
\printxticlabel{\numberTWOPI}{2\pi}
\printxticlabel{\numberTHREEPI}{3\pi}
\printxticlabel{\numberTHREEPI}{-3\pi}
\PRODUCTfunction{\IDENTITYfunction}
{\COSfunction}
{\Ffunction}
\PlotFunction[30]{\Ffunction}{-10}{10}
\end{Picture}

\begin{Picture}(-11,-11)(11,11)
\makenotics\makenolabels\cartesianaxes(-10,-10)(10,10)
\printxticlabel{\numberPI}{\pi}
\printxticlabel{\numberTWOPI}{2\pi}
\printxticlabel{\numberTHREEPI}{3\pi}
\printxticlabel{\numberTHREEPI}{-3\pi}
\PRODUCTfunction{\IDENTITYfunction}
{\COSfunction}
{\Ffunction}
\PlotFunction[30]{\Ffunction}{-10}{10}
\end{Picture}

2.5.3 Parametrically defined curves
You can declare the curve $x = f(t), \ y = g(t)$ with the \PARAMETRICfunction declaration. Assuming that functions $f$ and $g$ are stored in the commands \XFunction and \YFunction, we can declare the new parametric function \MyParametricFunction like this:

\PARAMETRICfunction{\XFunction}
{\YFunction}
{\MyParametricFunction}

To print it, use the \PlotParametricFunction command. An example:

³ The optional argument here means that the curve is approximated by 30 small quadratic pieces.

Robert Fuster
2.5.5 Drawing curves from a table of values

All the instructions to draw curves described here are based on the \texttt{qCurve} command, which draws quadratic Bézier curves. Specifically, \texttt{qCurve(x₀,y₀)(u₀,v₀)(x₁,y₁)(u₁,v₁)} draws a smooth curve between the points \((x₀,y₀)\) and \((x₁,y₁)\), with tangent vectors \((u₀,v₀)\) and \((u₁,v₁)\), respectively.

More generally, \texttt{PlotQuadraticCurve} draws a curve through several points.

With the \texttt{PlotQuadraticCurve} command you can approximate any smooth curve passing through a list of points when you know the tangent vectors. As a particular case of special interest (at least in a calculus course), the \texttt{PlotxyDyData} draws the graph of a function of a real variable from a table of values of the function and its derivative.

3 Implementation notes

The \texttt{xpicture} package loads the \texttt{curve2e} (and, then, \texttt{pict2e}), \texttt{xcolor}, \texttt{calculator} and \texttt{calculation} packages. The packages \texttt{curve2e} and \texttt{xcolor} are used as our interface with PostScript or PDF; so, \texttt{xpicture} is compatible with \texttt{dvips}, \texttt{DVIPDFx}, \texttt{pdflatex}, \texttt{Luatex}, \texttt{XeLaTeX} or any other \LaTeX{} successor supporting these packages.

The abilities of \texttt{calculator} are used to change the reference system (applying affine transformations in such a way that coordinates of points and vectors are internally converted to its standard coordinates), to convert from polar to rectangular coordinates, to

\begin{verbatim}
\PlotQuadraticCurve(1,0)(1,0)(0,1)(0,1)
(-1,0)(-1,0)(0,-1)(0,-1)
(1,0)(1,0)
\rotateaxes{45}
\PlotQuadraticCurve(1,0)(1,0)(0,1)(0,1)
(-1,0)(-1,0)(0,-1)(0,-1)
(1,0)(1,0)
\end{verbatim}

\texttt{PlotxyDyData(0,0,2)(1,1,0)(2,2,3)
(3,4,0)(5,1,-2)}

4 In earlier versions (only privately distributed at Universitat Politècnica de València), there was an option (not recommended) to omit loading \texttt{curve2e} and \texttt{pict2e}, so it was possible to compile the document as a pure \texttt{dvi}. But the quality of documents obtained was very poor and build time increased considerably. Therefore, and given that probably there is no reason to justify producing a \texttt{dvi} document instead of PostScript or PDF, that option has been abandoned.

The \texttt{xpicture} package
compute the precise position required by the \texttt{\Put} and \texttt{\multiPut}-like commands, and more.

Straight lines and vectors, once their standard coordinates have been calculated, are processed by \texttt{curve2e} commands.

Curves are locally approximated by quadratic Bézier splines, using the \texttt{\bez} command. This means that every curve is made up of several small quadratic approximations. A quadratic Bézier is determined by three control points: the endpoints of the curve, \( P_0 \) and \( P_1 \), and the point \( P_r \) where tangent lines at the endpoints intersect.

\begin{center}
\begin{tikzpicture}
\draw (-1,0) -- (1,0) node [right] {$P_r$};
\draw (0,-1) -- (0,1) node [left] {$P_0$};
\draw (1,1) -- (-1,-1) node [below left] {$P_1$};
\end{tikzpicture}
\end{center}

The \texttt{\bez}(\(x_0, y_0\))(\(x_r, y_r\))(\(x_1, y_1\)) command draws the Bézier curve whose control points are \( P_0(x_0, y_0) \), \( P_r(x_r, y_r) \) and \( P_1(x_1, y_1) \).

However, the \texttt{calculus} package can determine the points \( P_0(x_0, y_0) \) and \( P_1(x_1, y_1) \) that lie on the curve and the tangent vectors, \( \vec{u} \) and \( \vec{v} \), at these points.

\begin{center}
\begin{tikzpicture}
\draw (-1,0) -- (1,0) node [right] {$\vec{u}$};
\draw (0,-1) -- (0,1) node [left] {$\vec{v}$};
\draw (1,1) -- (-1,-1) node [below left] {$P_0$};
\draw (1,-1) -- (-1,1) node [below right] {$P_1$};
\end{tikzpicture}
\end{center}

Then, to determine \( P_r \), \texttt{xpicture} uses \texttt{calculus} to find the intersection of the two tangent lines, i.e., it solves the linear system
\[
\begin{align*}
u_1 y - u_2 x &= u_1 y_0 - u_2 x_0 \\
v_1 y - v_2 x &= v_1 y_0 - v_2 x_0
\end{align*}
\]

This is, essentially, what the command \texttt{\qCurve} does. All other commands to draw curves use \texttt{\qCurve}. For example, the command
\[
\texttt{\PPlotParametricFunction[n]{\F}{a}{b}}
\]
divides \([a, b]\) in \( n \) pieces, \([t_{i-1}, t_i]\) (\( 1 \leq i \leq n \)), computes \( F(t_{i-1}) \), \( F'(t_{i-1}) \), \( F(t_i) \) and \( F'(t_i) \), and calls \texttt{\qCurve}(\( F(t_{i-1}) \))(\( F'(t_{i-1}) \))(\( F(t_i) \))(\( F'(t_i) \)). If the tangent lines are parallel and not coincident, then it subdivides the interval into two half pieces.

4 Conclusions and future work

The aim of this package is not to compete with advanced general purpose drawing packages, but, first, to complete the abilities of the \texttt{picture} environment, by adding the capacity to use arbitrary reference systems and a greater control of the position of objects, and, second, to provide a solid tool for drawing scientific graphics.

Other well-known packages with the capability to draw graphs of curves from their equations (such as \texttt{pst-plot}, based on PSTricks (Van Zandt and Voß, 2013), \texttt{pgfplots}, based on PGF/TikZ (Feuersänger, 2012), and \texttt{LaTeX} (Reimers, 2011)) provide similar results. The differences between them lie in the way they make calculations to produce the graphs and in the syntax they use.

In this respect, the most interesting properties of \texttt{xpicture} are the following: the package is an extension of the standard \texttt{picture} environment and the syntax that is used is typical of \LaTeX{}; all calculations are performed directly by \TeX{}, by using the packages \texttt{calculator} and \texttt{calculus}; by using package \texttt{calculus} you can define virtually any elementary function or parametric curve; you can determine values of functions, lengths and other numerical parameters to fine-tune your picture, also by using \texttt{calculator} and \texttt{calculus}.

Finally, the most important feature of \texttt{xpicture} is, probably, its capability to change the reference system. As far as I know, no other package includes a general mechanism to manage arbitrary reference systems. The ability to change reference systems is a powerful tool for applying geometric transformations of objects without devoting a lot of effort to calculate coordinates. On the other hand, using this package you can draw virtually any curve that you can define by means of analytical equations. These are the most outstanding features of this package. Without reaching the high sophistication of packages based on PSTricks or PGF/TikZ, this package allows you to draw high quality graphics (especially scientific graphics) by using the standard \LaTeX{} syntax, hoping that users will feel comfortable without having to spend too much time learning to use it.

4.1 Enhancements

This package can be improved in several ways, some of them without expending too much effort. In the next version we will include extensions (compatible with the active reference system) of almost all commands of the standard \texttt{picture} environment and its extensions \texttt{pic2e} and \texttt{curve2e}. Namely, extensions of commands such as \texttt{\oval}, \texttt{\Curve} or \texttt{\lineto}, using the active reference system, will be defined.

Robert Fuster
Three dimensional straight lines and curves can be easily drawn by using a linear transformation between three- and two-dimensional spaces. Developing this idea, we intend to build a three-dimensional version of the \texttt{xpicture} package.

The other area that we want to develop is the representation of graphs (in the sense of graph theory). The idea is to define objects of types \texttt{node}, \texttt{edge}, \texttt{arc}, and \texttt{graph} and mechanisms to graphically represent them, and to manipulate them in order to show their different attributes. To draw a graph using \texttt{xpicture} is not very complicated, but we would like to provide a tool to easily choose the best display of the graph. This is a medium term project.

As longer term projects, it may be interesting to implement nonlinear coordinate transformations, which would allow, for example, to use logarithmic scales; the ability to draw smooth curves passing through several points (not including vectors of direction) would be another interesting utility (this could be done, for example, if we could build interpolating polynomials).

Acknowledgements

Claudio Beccari was kind enough to take a look at this package and suggested several improvements. Thanks to him, and to Gabriel Valiente Feruglio, who both took the time to make useful comments about \texttt{xpicture}. And many thanks to Karl Berry and Barbara Beeton for their patient kindness in reviewing my manuscript.

References


Reimers, Detlef. “L\textsc{a}P\textsc{d}F. Drawing in \TeX{} with PDF commands”. Available from CTAN, /macros/latex/contrib/lapdf, 2011.


○ Robert Fuster
Universitat Politècnica de València
Departament de Matemàtica Aplicada
Cami de Vera, 14
València E46022
Spain
rfuster (at) mat dot upv dot es
\url{http://www.upv.es/~rfuster/}

The \texttt{xpicture} package
Side-by-side figures in \LaTeX

Thomas Thurnherr

Abstract

Figures may be placed side-by-side for various reasons, such as comparing results generated under different conditions, because they are part of a bigger picture and therefore belong together, or simply to save vertical space in a document. \LaTeX{} knows several ways to align multiple figures neatly. These can generally be divided into standard environments and the more sophisticated packages. This article serves to introduce the different methods and highlight their differences.

1 General remarks on placing figures side-by-side

There are several factors controlling how figures are placed side-by-side. One such is the spacing between figures. By default, the methods described below leave little or no space between two sub-figures. Therefore, horizontal space needs to be added manually (if required) using, e.g., the standard lengths $\texttt{quad}$ and $\texttt{qquad}$ or the \texttt{hspace} command.

Another factor is how many figures are placed next to each other, or (equivalently) when to break a line. \LaTeX{} handles line-breaks automatically, implying that in order to place content side-by-side, one has to control the size of the figures. This is best achieved using $\texttt{\linewidth}$ (or a fraction thereof), a dynamic length parameter which adapts to the available width for content. The examples given below illustrate its usage. To force a line-break, it is sufficient to end the paragraph by adding a blank line and \LaTeX{} will start a new line.

This article introduces three packages: \texttt{subfig}, \texttt{subfigure} and \texttt{subcaption}. These packages offer many more options than the bare basics described here. They all come with extensive documentation available on your system, as part of the \TeX{} distribution, or online at CTAN (http://ctan.org).

The examples below all show how to arrange figures side-by-side. However, all methods work similarly with tables.

2 The \texttt{minipage} environment

The \texttt{minipage} environment is the most basic, and often sufficient, method to place figures side-by-side. Since \texttt{minipage} is not a floating environment, all figures have to go inside the \texttt{figure} floating environment. \LaTeX{} will determine the optimal position for the \texttt{figure} environment, which can be influenced through the optional parameter.

The example below illustrates how to align two figures side-by-side using the \texttt{minipage} environment:

\begin{verbatim}
\begin{figure}[ht]
\centering
\begin{minipage}[b]{0.45\linewidth}
\includegraphics...
\caption{Happy Smiley}
\label{fig:minipage1}
\end{minipage}
\quad
\begin{minipage}[b]{0.45\linewidth}
\includegraphics...
\caption{Sad Smiley}
\label{fig:minipage2}
\end{minipage}
\end{figure}
\end{verbatim}

Figure 1: Happy Smiley  Figure 2: Sad Smiley

The \texttt{minipage} environment works with figures, tables, lists, and paragraphed text as well as a mix of these content types. This fact implies, however, that the \texttt{minipage} environment does not primarily serve to align figures or tables, which is why specific packages like \texttt{subfigure} have been developed, providing additional figure- and table-specific functionality.

3 The \texttt{subfigure} package

The \texttt{subfigure} package is the oldest of a series of packages implementing commands for placing figures and tables side-by-side. It provides support for captioning and labeling of the sub-figures and sub-tables, which is missing in the \texttt{minipage} environment. After loading the package in the preamble, sub-figures and sub-tables are created using:

\begin{verbatim}
\usepackage{subfigure}
\begin{subfigure}[(lof entry)]{(sub-caption)}\%
  \begin{figure}
  \end{figure}
\end{subfigure}
\begin{subtable}[(lot entry)]{(sub-caption)}\%
  \begin{table}
  \end{table}
\end{subtable}
\end{verbatim}

To show the \texttt{subfigure} commands in context, here is a complete example aligning four figures side-by-side to illustrate a line break:

Thomas Thurnherr
3.2 Adding sub-captions to the lists of figures and tables with subfigure

By default, sub-captions are not added to the list of figures (lof) and list of tables (lot). However, the package provides a simple solution to add them to the respective list by setting the value of the counter \texttt{lofdepth, lotdepth} respectively, to 2 (default: 1).

\begin{figure}[ht]
\centering
\subfloat[Neutral Smiley]{
\includegraphics...
\label{fig:subfig1}}
\quad
\subfloat[Blush Smiley]{
\includegraphics...
\label{fig:subfig2}}
\subfloat[Sleepy Smiley]{
\includegraphics...
\label{fig:subfig3}}
\quad
\subfloat[Angry Smiley]{
\includegraphics...
\label{fig:subfig4}}
\caption{Main figure caption}
\label{fig:figure}
\end{figure}

3.3 hyperref and subfigure

The \texttt{subfigure} package supports using \texttt{hyperref} to link references and list entries with figures and sub-figures. However, when \texttt{\subref} is used, the link jumps to the main caption or sub-caption rather than the figure, which is not desirable. The packages need to be loaded in the correct order, with \texttt{hyperref} being last.

3.4 Deprecation of subfigure

The \texttt{subfigure} package was marked obsolete or deprecated as it was replaced by \texttt{subfig}. This means that the package is neither further developed nor maintained. However, conflicts and other potential issues are well documented and as long as this is kept in mind, nothing speaks against its usage.

4 The subfig package

The more recent \texttt{subfig} package was derived from \texttt{subfigure}. Therefore, the syntax is very similar, with one exception: It does not distinguish between figures and tables; both are produced by using the \texttt{\subfloat} command inside the desired environment.

\begin{figure}[ht]
\centering
\subfloat[Neutral Smiley]{
\includegraphics...
\label{fig:subfig1}}
\quad
\subfloat[Blush Smiley]{
\includegraphics...
\label{fig:subfig2}}
\caption{Main figure caption}
\label{fig:figure}
\end{figure}
4.1 Labeling and referencing with subfig

Similar to subfigure, the subfig package also implements the standard \ref and the \subref commands, producing the figure plus sub-figure labels or the sub-figure label only.

4.2 hyperref and subfig

Similar to subfigure, the subfig package supports the hyperref package. And again, hyperref needs to be loaded after subfig, and references to a sub-figure using \subref jump to the caption rather than the sub-figure.

4.3 Adding sub-captions to the lists of figures and tables with subfig

The subfig provides a slightly more convenient way to automatically add sub-captions to the lof and lot. It is sufficient to load the package with the lofdepth and lotdepth options:

\usepackage[lofdepth, lotdepth]{subfig}

5 The subcaption package

The subcaption package is the most recent of the three packages discussed here. The syntax is somewhat different from the other two packages: handling the size of figures is defined by the figure-enclosing environment, rather similar to minipage.

\usepackage[subcaption]{subfig}

\begin{subfig}[⟨position⟩]{⟨width⟩}
⟨figure⟩
\end{subfig}

\begin{subtable}[⟨position⟩]{⟨width⟩}
⟨table⟩
\end{subtable}

And again, the same example, this time using the commands provided by the subcaption package:

\begin{figure}[ht]
\begin{subfigure}[b]{.45\linewidth}
\centering
\includegraphics...
\caption{Neutral Smiley}
\label{fig:subcaption1}
\end{subfigure}
\quad
\begin{subfigure}[b]{.45\linewidth}
\centering
\includegraphics...
\caption{Blush Smiley}
\label{fig:subcaption2}
\end{subfigure}
\caption{Main figure caption}
\label{fig:figure}
\end{figure}

5.1 Labeling and referencing with subcaption

The package handles referencing the same way the two previous packages did, by providing the \ref and the \subref commands. The former produces a combination of the main plus the sub-label, whereas the latter produces the sub-label only.

5.2 Adding the sub-captions to the lists of figures and tables with subcaption

To add sub-captions to the lists of figures and tables, it suffices to load the package with the option list=true:

\usepackage[list=true]{subcaption}

5.3 hyperref and subcaption

The subcaption package is fully compatible with hyperref. The hyperref package needs to be loaded second and correctly links references and list entries with figures, tables, sub-figures, and sub-tables.

6 The columns environment in beamer

The presentation document-class beamer implements its own environment, called columns, for side-by-side content, in addition to minipage. The results with columns and minipage are almost exactly the same. Therefore, it is more of a personal preference which one to use. An example for minipage was given at the beginning of this article. To end, here is an example of how to place two figures side-by-side using the columns environment in beamer:

\begin{frame}{Frame title}
\begin{columns}
\begin{column}{0.45\textwidth}
\includegraphics...
\end{column}
\begin{column}{0.45\textwidth}
\includegraphics...
\end{column}
\end{columns}
\end{frame}

Thomas Thurnherr
texblog (at) gmail dot com
http://texblog.org
Glisternings
Peter Wilson

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.

We’ll meet again,
Don’t know where, don’t know when,
But I know we’ll meet again
Some sunny day.

We’ll Meet Again,
ROSS PARKER & HUGHIE CHARLES

1 Repetition
In September 2009 JT posed the following to the comp.text.tex newsgroup (ctt).

I have numbered propositions of the form:

(P1) Some proposition.
(P2) Another proposition.
(P3) Yet another proposition.

which are in a custom list environment and I can refer to the labels (P1, P2, etc) later in the document. However I sometimes want to also repeat the corresponding proposition like this:

Recall P2 from Chapter 1:

(P2) Another proposition.

Is there any way to output the entire list item without having to retype it?

Lars Madsen [5] responded with the following example code.

\documentclass[a4paper]{memoir}
\makeatletter
\newcommand\Reuse[1]{\@nameuse{forlater@#1}}
\newcommand\ForLater[2]{%\item[\#1]\def\@currentlabel{#2} %
  \global\long\@namedef{forlater@#1}{#2} %
  \Reuse{#1} %
\makeatother
\begin{document}
\begin{itemize}
\ForLater{P1}{Some long text}
\ForLater{P2}{More longer text. \par In paragraphs.}
\end{itemize}
\end{document}

Recall \ref{P2}:
\begin{itemize}
\item[\ref{P2}] \Reuse{P2}
\end{itemize}
\end{document}

As a demonstration that Lars’ \ForLater and \Reuse macros work, I used them in the description above of JT’s request.

In an earlier column I had tackled the question of repeating work in a somewhat different, and a not quite so elegant, manner [8].

\emph{verbatim et litteratim} — word for word and letter for letter.

Chambers Dictionary

2 Verbatims

2.1 \verb with an argument

Luca Merciadri asked on ctt if there was a way of defining a \verb macro that took the verbatim material as an argument enclosed in braces.

Ulrich Diez [2] responded with three solutions, the last two of which avoided any assignments. The first, shown below, looked much simpler to me.

\edef\verba#1#{\noexpand\verb#1\string}\
\let
\expandafter
\fi
\expandafter
\csname @gobble\endcsname
\string}

You can use \verba like this:

‘You can use either \verb{the \verb macro} or \verb*{the \verb* macro}, whichever suits.’

which will produce:

‘You can use either the \verb macro or the \verb* macro, whichever suits.’

However, just like \verb, \verba with its argument, cannot be used in an argument to another macro, not even in the argument to \verbatim.

2.2 Automatic line breaking

Hans Balsam asked on ctt:

I’m looking for a way to combine the features of the \verbatim environment and $$\LaTeX$$’s automatic line breaking.

‘Zappathustra’ (Paul Isambert) responded [3]:

\makeatletter
\def\zobeysp{ }
\makeatother
This redefines 使用空间字符，使其在 verbatim text 中变为一个正常空间，而不是一个不可打断的空间。然后你可以使用通常的‘verbatim’环境。

This proposal works, albeit with at least one surprise — a space following a comma gets swallowed so a double space should be used instead of a single space. The other potential surprises are that hyphenation is disabled and multiline verbatim text is set ragged right.

If any man will draw up his case, and put his name at the foot of the first page, I will give him an immediate reply. Where he compels me to turn over the sheet, he must wait my leisure.

3 Small pages

Harald Hanche-Olsen asked [slightly edited] on ctt:

I’d like to make some PDF files especially for reading on screen, more specifically on the iphone. For much of this, a fixed page length seems like a straitjacket. I want to divide the material into pages so that one topic will fit on one page. Some pages will be very short while others will be very long. I don’t want oceans of white space at the bottom of the pages.

I imagine doing this with BTEX ... [but] the output routine gives me goose bumps ... I don’t plan on using marginal notes and if I must do without floats and footnotes, that is fine too. I could of course do it in plain TEX, but would like to have the added power of BTEX available.

Will Robertson responded [7] with a potential solution based on the preview package [4]. His code follows, and I have taken the liberty of extending it very slightly to enable it to work with a variety of classes, and also extending the example document.

\documentclass[article]{memoir}
\documentclass{article}
\usepackage[charter]{article}
\usepackage{lipsum}
\makeatletter
\@ifclassloaded{memoir}{\let\section\chapter
\let\raggedy\raggedyright
\begin{preview}
\begin{minipage}{5cm}
\begin{minipage}{4.5cm}
\footnotesize\raggedy
\parindent=2em
\end{minipage}
\end{preview}}{\usepackage{ragged2e}
\let\raggedy\RaggedRight
\makeatother
\begin{minipage}{5cm}
\begin{minipage}{4.5cm}
\footnotesize\raggedy
\parindent=2em
\end{minipage}
\end{preview}}
\begin{document}
\begin{page}\tableofcontents\end{page}
\begin{page}
\section{Foo}
\lipsum[1]
\end{page}
\begin{page}
\section{Bar}\lipsum[2-5]\end{page}
\begin{page}
\section{Fuz}
Some text here. I wonder if one can have a marginal note. %\marginpar{At the side} It doesn’t work!
\section{Fuzzy}
What if we have two ‘sections’ on the same page?
\begin{page}
\begin{page}
\section{Fie}
Some text here. I wonder if one can have a footnote. %\footnote{At the end like this} It works!\par
\lipsum[2-5]
\end{page}
\end{page}
\end{document}

It is not possible to demonstrate Will’s page environment here, but it does seem to meet Harald’s request as far as I understand it. Floats do not work, and page numbers are not printed, but \tableofcontents and the hyperref package [6] work (at least as used in the test code above), if needed.

Peter Wilson

\begin{minipage}{5cm}
\begin{minipage}{4.5cm}
\footnotesize\raggedy
\parindent=2em
\end{minipage}
\end{preview}}
\begin{document}
\begin{page}\tableofcontents\end{page}
\begin{page}
\section{Foo}
\lipsum[1]
\end{page}
\begin{page}
\section{Bar}\lipsum[2-5]\end{page}
\begin{page}
\section{Fuz}
Some text here. I wonder if one can have a marginal note. %\marginpar{At the side} It doesn’t work!
\section{Fuzzy}
What if we have two ‘sections’ on the same page?
\begin{page}
\begin{page}
\section{Fie}
Some text here. I wonder if one can have a footnote. %\footnote{At the end like this} It works!\par
\lipsum[2-5]
\end{page}
\end{page}
\end{document}

He fixed thee ‘mid this dance
Of plastic circumstance.

Rabbi Ben Ezra, Robert Browning

4 Prefixing section heads

‘Ghoetker’ wrote to ctt along the following lines: I have changed the formatting of subsections in my document to start with the term ‘Activity’ (trust me, it made sense). So, the subsection heading is ‘Activity A.1. Test’. When I cross-reference, however, this isn’t what I want — I just want the ‘A.1’ part ...
As is so often the case, Donald Arseneau came up with an elegant solution [1] shown below. But first, to set the context:

The \LaTeX{} kernel \texttt{\@seccntformat} macro typesets the number of a (sub-)section head, and takes one argument which is the name of the section head. Its default definition is:

```latex
\newcommand*{\@seccntformat}[1]{% 
  \csname the#1\endcsname\quad}
```

and in, for example, a \texttt{\subsection} it would be called as:

... \@seccntformat{\subsection}...

resulting in the code:

... \thesubsubsection\quad...

Donald proposed:

\begin{verbatim}
\makeatletter
\renewcommand*{\@seccntformat}[1]{% 
  \@ifundefined{#1prefix}{}% 
  \csname #1prefix\endcsname\ }% 
  \csname the#1\endcsname. \quad}
\makeatother
\renewcommand*{\thesubsection}{% 
  \Alph{section}.$\arabic{subsection}$}
\newcommand*{\subsectionprefix}{Activity}
\end{verbatim}

As well as putting 'Activity' before subsection head numbers it also has the effect of putting a '. ' at the end of every sectional number. Using the above will result in \texttt{\section} heads like '2. Title', \texttt{\subsection} heads like 'Activity A.2. Title' and \texttt{\subsubsection} heads like 'A.2.3. Title'.

If the '.' after every sectional number is not required this can be dealt with by extending Donald's code to cater for putting something specific after the heading number, which can then be different for each section level:

\begin{verbatim}
\makeatletter
\renewcommand*{\@seccntformat}[1]{% 
  \@ifundefined{#1prefix}{}% 
  \csname #1prefix\endcsname\ }% 
  \csname the#1\endcsname\quad}
\makeatother
\renewcommand*{\thesubsection}{% 
  \Alph{section}.$\arabic{subsection}$}
\newcommand*{\subsectionprefix}{Activity}
\newcommand*{\subsectionpostfix}{.}
\end{verbatim}

Using the above will result in \texttt{\section} heads like '2 Title', \texttt{\subsection} heads like 'Activity A.2. Title' and \texttt{\subsubsection} heads like 'A.2.3 Title'.

\begin{thebibliography}{8}


\end{thebibliography}
The esami package for examinations*

Grazia Messineo and Salvatore Vassallo

Abstract

The package esami is a small collection of macros to prepare written examinations and tests for students at universities or secondary schools. It generates output in which questions and answers are scrambled. Exercises depend on random parameters, the values of which are assigned during the compilation and on which you can do many arithmetic operations.

1 Introduction

Among the main topics on tex.stackexchange.com are questions about writing exams with many different versions, scrambling questions or lists, hiding answers in tests, etc. We wrote this package in order to solve some of these problems for written maths examinations at the Faculty of Economy of Catholic University in Milan.

We began to develop the package in 2008 and we tried to extend some useful properties of the \LaTeX\ packages exerquiz by D. P. Story [4] and probsoln by N. Talbot [5]. In particular we liked the idea of creating a database of exercises from which we could select one or more. (The reader should know that we began work with the 2008 versions of these packages and never updated them.) Moreover we needed to be able to use random parameters in exercises and to generate many (from 12 to 100 or more) different, but similar, versions of the same assignment. Over the years, the development of our package has had several points in common with the development of Dr. Story’s package exerquiz and others; we found similar solutions, but with different code.

The package was developed having mathematics in mind, but can be used in different fields.

2 Our goal

Our goal was the creation of a simple database of exercises: many files, including many variants on several “basic” ones; the exercises can depend on random parameters and they can be multiple choice questions (MCQ) or problems, i.e. exercises with solutions, simple or divided in multiple parts. When the examination is generated, the process is:

- selection of the files with the exercises,
- random choice of the version of the exercise in the files,
- shuffle of the exercises chosen,
- shuffle of the answers in the MCQ,
- assignment of values to random parameters.

Moreover, we need a file with the solutions of the exercises and, for any version, a string (or more) with the correct choice for the MCQ. Some of these facilities were already contained in the package exerquiz (and some have been introduced by Dr. Story in later versions of the package). The choice of the variant of an exercise was solved by changing a command of the package probsoln. We wanted the package to be straightforward to use for users with very little knowledge of \LaTeX.\n
3 The exercises typology

Our aim was to use two types of exercises: “tests”, that is, sets of multiple choice questions, and “problems”, that is, exercises with an articulate development, such as the study of a function or the discussion and solution of a linear system. But we also wanted to be able to use the same software in different situations, so we decided to implement other typologies of “exercises”:

(a) questions with a short answer where the student has to write the answer in a provided space at the end of the exercise;
(b) “fill-in” questions; in which some blank space has to be filled (for example in theorems);
(c) questions with a “long” answer;
(d) tables to be completed;
(e) “matching”: exercises where the student has to connect the elements of two lists (like nations and capitals).

Naturally, all these exercises can contain random parameters, can be shuffled and can have a different appearance.

4 The randomization problem

Instead of using only the package random for the random choices, we preferred to use a “mixed” method. The shuffling of the answers in the MCQ and choosing of values of random parameters are totally random. On the other hand, the choice of a variant of an exercise is random if the number of variants is greater than a predetermined value (currently set at 8); otherwise, a permutation among 24 possible permutations (6 if there are 3 variants) is chosen (deterministically, based on the version number of the exam). A similar procedure is also used for choosing the order of the exercises in the task.

The seed of the randomization process is based on a combination of the exam date and the version

---

* Work supported by Catholic University Research Project “Progetto M.In.E.R.Va: Creazione di esercizi di tipo interattivo con percorsi differenziati per il recupero delle carenze e la valorizzazione delle eccellenze in matematica”.

Grazia Messineo and Salvatore Vassallo
number, such that different versions on the same date are essentially different and the “same” exams given on different dates are different. Moreover, when a value is given to a random parameter the seed is changed using the order number of the exercise, so that parameters defined in the same interval, but in different exercises, are not all equal.

5 The exam

In order to generate an assignment we need some files besides the .sty file:

(a) the files with the exercises chosen from the database;

(b) a “master” file in which the user has to write the date of the exam, the number of versions to generate, and the name of the files of the exercises; in this file it is also possible to change the appearance, for example the geometry of the page, or the size of the font (for this, some little \LaTeX knowledge is needed);

(c) another “master” file for solutions similar to the previous one, that generates the solutions for any version of the exam and the string of the correct answers to the MCQ;

(d) a configuration file with commands for the footers, the headers, the geometry of the page, the instructions for the students, etc.

One more optional file can be used to check the database: for any file of exercises it prints all the variants both in numeric and in parametric format. This uses an option that modifies the way in which the mathematical expressions are elaborated by \LaTeX.

So the task of the teachers is the creation of the database of exercises written using the instructions of the package.

We had many reasons for this type of structure:

(a) the process of production and checking of the exercises is completely independent from the generation of an assignment, even if the format of the files is the same: this allows the creation of a database of exercises that can be increased as the teachers have time;

(b) the appearance of the assignment can be modified without any change to the .sty file;

(c) users with little or no knowledge of \LaTeX can generate the assignment if the database is sufficient;

(d) with some small modifications to the master file it’s possible to obtain exams in one or more parts, with different kinds and numbers of exercises, etc.

6 The database of exercises

Every exercise with all its variants is written in a separate file; each variant is enclosed in the command \newproblem, a highly modified version of the (2008 version) command with the same name in the package probsoln. This command has just one argument: the text of the exercise itself.

6.1 Multiple choice questions

If the exercise is an MCQ the syntax is almost the same as that of exerquiz:

\item \PTs{(points)}
... Exercise text ...
\begin{answers}{(number-of-columns)}
\bChoices[random]
\Ans0 incorrect answer \eAns
\Ans0 incorrect answer \eAns
\Ans1 correct answer \eAns
\eFreeze
\Ans0 none of the preceding \eAns
\eChoices
\end{answers}

where:

- The ‘\item \PTs{(points)}’ introduces a question with a score of \PTs points\(^1\) (it can also be a decimal number and the separator can be the comma, unlike in \LaTeX);

- \begin{answers}{(number-of-columns)}
  \bChoices[\random]
  ...
  \eChoices
  \end{answers}

typesets the answers in \(\text{(number-of-columns)}\);

if there is the option \random, the answers are randomly shuffled;

- \Ans0 indicates an incorrect answer;

- \Ans1 indicates a correct answer;

- \eFreeze: after this command the answers are not randomized, and they appear at the end of the list.

Unfortunately, the method we used to obtain the string with the correct answers to a set of MCQ has for now excluded the possibility of using questions with more than one correct answer or with answers having different scores.

6.2 Open exercises

If the exercise is an open exercise (i.e. an exercise with a complete solution), it is embedded in the

\(^1\) Since the package is written for Italian users, the default label is “punto” or “punti” (Italian words for “point” and “points”): this can be changed using the macro \PTsHook.

The \texttt{esami} package for examinations
environment \texttt{problem} or \texttt{problem*} (if it has one or more parts). The syntax is:
\begin{problem}[(\textit{score})]
... Text of the exercise ...
\end{problem}
\begin{solution}[(\textit{space-for-sol})]
... solution ...
\end{solution}
where \texttt{⟨space-for-sol⟩} is the height of the (optional) blank space left for the solution and \texttt{⟨score⟩} is the score of the exercise. If it is an exercise with multiple parts:
\begin{problem*}[(\textit{total-score})]
... text ...
\item \texttt{⟨partial-score⟩} 
... text ...
\begin{solution}[(\textit{space-for-sol})]
... text of solution ...
\item \texttt{⟨partial-score⟩} 
... text ...
\end{parts}
\end{problem*}
where \texttt{⟨partial-score⟩} is the score of each part.

The package \texttt{exerquiz} has the facility of automatically calculating the total score of an exercise. In our package, due to the shuffling of the exercises, this is not always possible.

6.3 Other types of exercises

The other types of exercises we defined are:

\texttt{fill-in} For creating exercises in which some text is left blank and must be filled in by the student, or exercises with an open short answer. The syntax is:
\begin{fillin}[⟨type⟩]{⟨width-of-blank⟩}{⟨answer⟩}
\end{fillin}
The two mandatory parameters are the width of the blank space, expressed as a length, and the correct answer — text or number — that the student has to write: it will be printed in the solutions only. The optional parameter \texttt{⟨type⟩} defines the way the blank space is denoted: \texttt{u} (\texttt{underlined}), the default, produces an underlined space; \texttt{b} (\texttt{boxed}) produces a little box; \texttt{e} (\texttt{empty}) produces an empty space. In the blank space it’s not possible to use the commands for the simplifications (see Section 9).

Example 1
The capital of Italy is \texttt{\fillin[u]{5cm}{Rome}}, the capital of France is \texttt{\fillin[b]{4cm}{Paris}}

The capital of Italy is \underline{Rome}, the capital of France is \underline{Paris}

\texttt{matching} This is based on an idea from the package \texttt{examdesign} [1]. It is used to create exercises in which the student has to match items in two lists. The pairs are defined with ⟨\{item1\}⟨item2\}⟩, repeated for each pair of items to match. The two lists are shuffled and then printed with the command \texttt{\matching}.

Example 2
\begin{center}
\begin{tabella}{l}
\hline
Italy & Rome \cr
\hline
Germany & Berlin \cr
\hline
Greece & Athens \cr
\hline
\end{tabella}
\end{center}
The solution shows the correct matching.

\texttt{tabella} This is used to create exercises with many short open answers in a column. The syntax is (the \texttt{\cr} at the end of the line is necessary):
\begin{center}
\begin{tabella}[(\textit{num-visible-cols})]
{⟨visible-cols-align⟩}
{⟨hidden-col-align⟩}
... & ... \cr
\end{tabella}
\end{center}
The optional parameter (default 2) is the number of columns of the table visible in the text of the exercise. The last column is invisible in the text and visible in the solutions. The second parameter gives the alignment of the visible columns (the same for all the columns) and the third the alignment of the hidden column.

Example 3
\begin{center}
\renewcommand\arraystretch{3}
\begin{tabella}{1}{1}{1}
\hline
The domain of the function is:
& \mathcal{D}=(\mathbb{R};2] \cr
\hline
The range of \mathcal{f}(x) is:
& \mathcal{f}(\mathcal{D})=(\mathbb{R};0] \cr
\hline
\end{tabella}
\end{center}
we obtain (the second column is visible only in the solutions):

\begin{tabular}{|l|l|}
\hline
The domain of the function is: & \mathcal{D}=(\mathbb{R};2] \\
\hline
The range of \mathcal{f}(x) is: & \mathcal{f}(\mathcal{D})=(\mathbb{R};0] \\
\hline
\end{tabular}
The following environments don’t define exercises, but help to format or check the exercises.

**problema and problema** These environments are like **problem and problem**, but if the package option **solutionsonly** is specified, only the solution of the exercise is printed and not the text.

**risposta** This environment generates a ruled or boxed space in which the student has to write the answer to an exercise (“risposta” is the Italian word for “answer”). The syntax of the command is:

```
\begin{risposta}{\{type\}\{vertical-space\}}
\end{risposta}
```

The `{type}` parameter defines if the blank space has to be boxed (option `b`, the default) or ruled (option `l`). The parameter `{vertical-space}` defines the height of the space for the answer: it is a length if it is boxed or the number of rules if it is ruled.

**workarea** This environment defines a blank space on the paper sheet where the student can write. In this space it’s possible to put some text, a graphic, coordinate axis, etc. The syntax of the command is:

```
\begin{workarea}{\{height\}}
\end{workarea}
```

```
\begin{workarea}{\{width\}\{height\}}
\end{workarea}
```

The height of the `solution` and `workarea` environments should be equal; if the `workarea` height is larger, the text of the `workarea` will be misaligned in the space of the solution, overlapping with the exercise. The width of the `workarea` is optional and by default is equal to the textwidth.

### 7 The master files

#### 7.1 The file master and master-sol

The only difference between these two files is that the second one shows the solutions. They contain all the instructions to generate the exam. In both files it’s necessary to write:

- the (same) date in the (same) format, namely `{day}/{month}/{year}` (the day and month can be in any format, the year should be written with four digits: 3/12/2012, 03/7/2013),
- the name of the exercises (command `\esercizio`),
- the number of versions (command `\numcompiti`).

The exam can be in multiple parts and in any part it’s possible to use one or more of the environments defined above and one or more commands for the choice of the exercises. In the file there is also the definition of the random seed (command `\seme`). It is also possible to use the classical sectioning commands.

In these files the MCQ are embedded in the environment `test` with the optional parameter `{score}`. In this environment there are one or more sets of MCQ, each introduced by `\begin{questions}`.

The other kinds of exercises can be contained or not in the `\begin{questions}` environment, except that `problem` and similar cannot be there. However, although fill-in exercises with more than one blank to fill and matching exercises can be used in a `test` environment, the string of correct answers at the end of the file is no longer useful because the numbering of questions is wrong. If you are not interested in the final string of correct answers, you can use them without any problem. (See also the `fillb` package option described later.)

#### 7.2 The file `totale-versioni`

The file `totale-versioni` (i.e. all the versions) is used to generate all the versions of an exercise that are in a file of the database and it’s desired to check them. In this case the master must have the option `prova` (see Section 8); the compilation gives a numeric version and, with the option `param` the parametric version (with the random parameters not evaluated) of the exercises.

The `totale-version` file itself has just one command, `\def\esercizio{\{file\}}`, where `{file}` is the name of the file of exercises. When compiling the parametric version the name of the parameters and their range of variation will be printed. The file works similarly to the command `\selectallproblems` of the package `probsoln`.

### 8 Package options

The package `esami` has many options:

- `allowrandomize` and `norandomize`: with the first the answers in MCQ are shuffled (default), with the second they are printed in the order they are written;
- `shuffle, shufflerandom` and `noshuffle`: the first (default) shuffles the exercises (randomly if there are more than eight, in a deterministic way if there are 8 or less), with the second the exercises are always shuffled randomly (by uncommenting some lines in the file `esami.sty` it’s possible to make the choice be random for more than $n < 8$ exercises and deterministic otherwise), with the third the exercises are not shuffled at all;

The `esami` package for examinations
• xxxx: reads the file ‘esami-xxxx.cfg’ that contains some commands and configurations, such as the name of the course, instructions for the students, etc. The names of some configuration files are given in the file esami.sty, but it’s possible to read another configuration file without modifying anything: it’s sufficient to put a unknown option like zzz and create the file esami-zzz.cfg;
• pointsonright: a boolean option that generates a little box on the right of the page with the score of the exercise
• nosolutions: with this option the exam is generated without solutions (default);
• solutions: generates the file of solutions;
• solutionsonly: generates a file with solutions only if the environment problema is used;
• prova: as mentioned above, when compiling the file totale-versioni with this option, a PDF file is generated with all the variants of an exercise; the correct answers of all MCQ and the solutions of the exercises are automatically shown;
• param: with this option, used only in conjunction with the option prova, the versions of the exercise are printed in parametric form; it also shows the range of variation of the parameters;
• correzione: can be used only with the option prova, to print only the text of all the exercises, without solutions;
• fillb: this option is necessary to have the correct answers in the string of solutions if there are exercises of fillin type;
• twocolumns: with this option, the MCQ are printed in two columns;
• sansserif: a sans serif font is used.
• autopston and autopstoff: both these options load the package auto-pst-pdf, in the second case with the option off; in this way it’s possible to compile the file directly with PDFLaTeX even if the exercises contain graphics in pstricks—the graphics package we use. With the first option, the images are generated and included in the document, while the second doesn’t generate the images but includes them if they exist.

9 Package commands

9.1 Commands working with parameters

As we said above, one of the goals of the package is to use random parameters in exercises. We defined only integer parameters but it is possible to define also rational or (pseudo)real parameters, as D.P. Story does in the package rangen [3]. Since we use the package fp [2] to do calculations, almost all the commands operating on parameters are prefixed by FP. The command to define a parameter is FPsetpar⟨⟨seed⟩⟩⟨⟨param-name⟩⟩⟨⟨inf⟩⟩⟨⟨sup⟩⟩⟨⟨excl-values⟩⟩.

• the name of the random parameter will be the control sequence ⟨⟨param-name⟩⟩;
• the parameter’s range will be between ⟨⟨inf⟩⟩ and ⟨⟨sup⟩⟩ (inclusive);
• the optional ⟨⟨seed⟩⟩ is used to have a different seed for the generation of the random number, with a default value given by \seme (the Italian word for seed) defined in the preamble;
• one or more values can be excluded from the choice with ⟨⟨excl-values⟩⟩. If there is more than one excluded value, the whole list is enclosed in braces.

The lower and the upper bounds ⟨⟨inf⟩⟩ and ⟨⟨sup⟩⟩, with ⟨⟨inf⟩⟩ < ⟨⟨sup⟩⟩ and the excluded values can be random parameters defined earlier. In order to satisfy the conditions the generation of the random number may be repeated many times; the maximum number of repetitions is given by the command \maxLoopLimit, by default 10 (this can be redefined in the preamble of the document).

Example 4
\FPsetpar{a}{2}{10}[3]
\FPsetpar{b}{4}{12}[\a,6]
generates two random numbers \a (with range between 2 and 10, but not 3) and \b (with range between 4 and 12, excluding both the value assigned to \a and 6).

We defined some commands in addition to those in the fp package to do operations on parameters.

The command \FPsv{⟨⟨decimal⟩⟩}{⟨⟨operation⟩⟩} is used to evaluate ⟨⟨operation⟩⟩ (on numbers or parameters) obtaining either the numeric value with ⟨⟨decimal⟩⟩ decimal places (by default 0 decimal places) or, with the package option param, the typesetting of the operation.

Example 5 \FPsv{2*k+1}, with (say) \k = 2, gives either 5 or, with the option param, 2 * \k + 1; \FPsv{2*(2*k+1)/2} gives 2.50 or (2 * \k + 1)/2.

The syntax of the arithmetic operations is the same as in the package fp. When used with param, it’s easier to read if the operations are given in parentheses.

The command \FPval{⟨⟨name⟩⟩}{⟨⟨operation⟩⟩} assigns to \langle⟨name⟩⟩ the rounded result of ⟨⟨operation⟩⟩. (This is a modified form of the command \FPval from fp.)
Example 6

\FPsetpar\k\{1\}\{3\}
\FPval\a\{2\}\{k+1\}
\FPsetpar\b\{2\}\{20\}\a

generates a random parameter \b which assumes a value between 2 and 20, but different from \a, where \a is given by 2*\k+1. In the parametric version it will appear like this:
The parameter \b varies from 2 to 20. \b \not= (2 * \k + 1).

We also defined some commands to simplify fractions, that can also be used for correct formatting of the text.

The command \sempli\{\num\}\{\den\} simplifies a fraction where \num and \den can contain parameters or operations on them.

Example 7 If \k = 1, \sempli\{2*\k\}\{3*\k+1\} gives \frac{1}{2} or, with \param specified, \frac{2*\k}{3*\k+1}.

The command \semplix\{\num\}\{\den\} simplifies a fraction where \num and \den can contain parameters, but where the result 1 does not appear and the result −1 is shown as just a minus sign “−” (for example to be used before an \x). This command can also be used to format coefficients of a variable, setting the denominator equal to 1.

Example 8 If \k = 2, \FPsv\{k-1\}\x gives \x while \semplix\{k-1\}\{1\}\x gives just \x.

The command \esempli\{\num\}\{\den\} simplifies a fraction such that the result 1 does not appear, and the result −1 has to appear explicitly (as in exponents). The command can be used with a denominator of 1 to correctly format the exponents.

Example 9 If \k = 2, \x^\FPsv\{k-1\} gives \x while \x^\esempli\{k-1\}\{1\} gives just \x.

The command \esempliz\{\num\}\{\den\} simplifies fractions that can assume the value 0; with the other commands, the result 0 gives an error and stops the compilation.

The command \simpssqrt\{\ind\}\{\rad\} allows extracting factors from radicals; however, it is not possible to do other operations with these factors. The first mandatory parameter \ind is the index of the radical and can be parametric; the second one, \rad is the radicand and can be also a parameter or an operation.

Example 10 If \a = 2 and \b = 1,
\simpssqrt\{2\}\{\a^2+\b\} gives \sqrt{2}.

9.2 Commands for exercises and lists

The commands to manage both exercises and lists are in the same category since they work in the same way: given a list of tokens, they shuffle the objects and pick some elements.

The main command to manage exercises is: \esercizi\{file1\}, \{file2\}, ..., \{fileN\}.

This chooses a random exercise for each given file, shuffles them and sends the result to the output.

The command \estrai\{\param\}\{\list\}\{\name\}\{\name\} gives just \x when \param being a comma separated list of \n objects, picks \n − \m elements from \list; the selected elements will be called \\name\i, \\name\ii, and so on; they can be used, for example, within the command \esercizi.

Example 11

\estrai\{sets,log,exp\}\argi

This chooses two elements of the given list, setting \argi and \argii to the values: by writing \esercizi\{\argi, \argii\} we obtain two random exercises, one from each of the two randomly-chosen topics (sets, logarithms, and exponentials).

The command \estrai\esami\{\param\}\{\list\}\{\name\}\{\name\} similarly picks \n random objects from \list, but preserving the order. As before, the elements will be called \\name\i, \\name\ii, etc.

Example 12

\estrai\{a,b,c,d\}\alpha

This chooses two elements from the given set of four letters, while preserving alphabetical order. The chosen elements are stored in \alpha and \alphai.

Finally, the command \estrai\esami\{\num\}\{\list\} also works similarly to the command \estrai, but only on an exercise’s list; the chosen elements go to the output instead of being stored.

With these commands we can have many different possibilities of random choice.

It’s possible to use the commands \esercizi and/or \esami many times. This is useful if one would like to have exercises from two or more different subsets (for example, 5 exercises about limits chosen from 7 available, and 3 about derivatives chosen from 5) or, more simply, if one likes to have some exercises in two columns and others in one column.

10 Open issues

In the package there remain open issues. The code can be improved and made more efficient, in particular in the management of the lists — for example using etoolbox — and the solution we found to obtain the string of the solutions of the MCQ, using
\label and \ref and the aux file, doesn’t allow for questions with more than one correct answer.

Other improvements can be made from an aesthetic point of view: in particular, we decided to put each MCQ in a minipage to avoid misunderstandings by the students if a question was split across pages. As a result, the output is sometimes very ugly.

11 Examples of the working files

To conclude, here is a set of small complete files. First, an exercise file test1.tex, with one MCQ:

\begin{verbatim}
\newproblem{ \FPsetpar{a}{2}{5}
\item \PTs{1} exercise 1a
\begin{answers}{1}\bChoices[random]
\Ans1 answer 1 correct\eAns
\Ans0 answer 2 wrong\eAns
\Ans0 answer 3 wrong\eAns
\eChoices\end{answers}}
\end{verbatim}

Next, the file totale-\text{versioni} used to print all versions of an exercise file (in this case, with MCQ):

\begin{verbatim}
\documentclass[english]{article}
\usepackage[text,shuffle,nosolutions]{esami}
% make parametric version;
% for the numeric version, omit 'param'
\date{30/4/2008} % for the seed
\begin{document}
\FPeval\seme{round((\thenomefile+\thevers):0)} % the random seed can be anything;
% \thenomefile ‘is’ the date
\randomi=\seme
% immediate\write\sols{\string\begin{minipage}{.3\textwidth}Solution of Version \thevers
% \textwidth}Solution of Version \thevers
% \% for solutions
\begin{test} %\% MCQ for a total of 6 points
\begin{questions}
esercizi{test1}
\end{questions}
\end{test}
\end{document}
\end{verbatim}

Finally, an example of master file for generation of the exam (or solutions). The commented-out commands are for solutions.

\begin{verbatim}
\documentclass[english]{article}
\usepackage[shuffle,nosolutions]{esami}
% usepackage[shuffle,solutions]{esami}
\begin{document}
\selectallproblems{esercizio}
\begin{test}[6] %\% MCQ for a total of 6 points
\begin{questions}
esercizi{test1}
\end{questions}
\end{test}
\end{document}
\end{verbatim}

References


\diamond Grazia Messineo
Università Cattolica Milano
Largo Gemelli, 1
Milan, I-20123; and
ITC “G. Falcone”
Viale Italia, 22
Corsico, I-20094 Italy
grazia dot messineo (at) unicatt dot it

\diamond Salvatore Vassallo
Università Cattolica Milano
Largo Gemelli, 1
Milan, I-20123
salvatore dot vassallo (at) unicatt dot it
E-TeX: Guidelines for Future TeX Extensions — revisited
Frank Mittelbach

Contents

1 Introduction 47
2 A short history of “Extended”-TeX engines 48
  2.1 pTeX 48
  2.2 ML-TeX 48
  2.3 \NTeX 49
  2.4 \eTeX 49
  2.5 Omega/Aleph 49
  2.6 pdiTEx 49
  2.7 \XeTeX 49
  2.8 LuaTeX 50
  2.9 iTex 50
3 Review of the issues raised in 1990 50
  3.1 Line breaking 50
  3.1.1 Line-breaking parameters 51
  3.2 Spacing 51
  3.3 Page breaking 52
  3.4 Page layout 54
  3.5 Penalties — measurement for decisions 55
  3.6 Hyphenation 55
  3.7 Box rotation 56
  3.8 Fonts 56
  3.9 Tables 57
  3.10 Math 57
  3.11 TeX's language 58
4 Overcoming the mouth/stomach separation 59
  4.1 A standard TeX solution 60
  4.2 A LuaTeX solution 61
5 Conclusions 61

List of Figures

1 TeX-like engines evolution 48
2 Areas of concern in original article 50
3 Interword spacing 52
4 TeX's box/glue/penalty model 53
5 Baseline to baseline spacing 55
6 The expl3 logo 58

Abstract

Shortly after Don Knuth announced TeX 3.0 I gave a paper analyzing TeX's abilities as a typesetting engine. The abstract back then said:

Now it is time, after ten years' experience, to step back and consider whether or not TeX 3.0 is an adequate answer to the typesetting requirements of the nineties.

Output produced by TeX has higher standards than output generated automatically by most other typesetting systems. Therefore, in this paper we will focus on the quality standards set by typographers for hand-typeset documents and ask to what extent they are achieved by TeX. Limitations of TeX's algorithms are analyzed; and missing features as well as new concepts are outlined.

Now — two decades later — it is time to take another look and see what has been achieved since then, and perhaps more importantly, what can be achieved now with computer power having multiplied by a huge factor and, last but not least, by the arrival of a number of successors to TeX that have lifted some of the limitations identified back then.

1 Introduction

When I was asked by the organizers of the TUG 2012 conference to give a talk, I asked myself:

What am I currently working on that could be of interest?

The answer I gave myself was: I'm working on ideas to resolve or at least lessen the issues around complex page layout; in particular mechanisms to re-break textual material in different ways so that you can, for example, evaluate different float placements in conjunction with different caption formats, or to float galley text in different ways around floats protruding into the galley.

All that goes way back in time: the issues were formulated more than 20 years ago in a paper I gave in 1990 in Texas: “E-TeX: Guidelines for future TeX extensions” [26]. Back then there were no answers to the issues raised. However, that was a long time ago; computers got faster and people invented various TeX extensions since then — and once in a while there are even new ideas.

So when I reread my paper from that time I thought that it would be a good idea to analyze the issues listed from the 1990 paper again and see what has been achieved since then.

This paper starts with a short overview of the history of TeX's successor engines and the capabilities they added or improved. We will then re-analyze issues discussed two decades ago and evaluate their status.

We conclude with a summary of the findings and a suggestion for a way forward.
2 A short history of “Extended”-\TeX engines

Professor Donald Knuth developed the first version of the \TeX program in 1978–79 [16, 17]. (The first specifications in writing date back to 1977; see [21].) Over the course of the next two years he improved and changed the program further and it then became known as \TeX 82. This was the first widespread version of \TeX, with documented source code [19] and a published manual [18], and people all over the world started using it.

Back then \TeX used 7-bit fonts and typesetting in languages that required diacritics (as most European languages do) was difficult because, for example, hyphenation didn’t work properly in that case. Also, mixing of several languages in one document was impossible, at least if one wanted them to be hyphenated automatically.

Therefore, in 1989 a delegation of \TeX users from Europe came to the Stanford meeting and presented Don with a proposal to extend \TeX in several ways [31]. After several meetings and public discussions, Don recognized that he did not originally foresee a need for 8-bit input, and he agreed to extend \TeX slightly to account for the needs of the extended user base [20]. However, he only accepted those proposals that could be achieved with minimal adjustments to the existing program. If you compare The \TeXbook before and after, you will have difficulty spotting the differences, because apart from the introduction of language support, nothing much changed. For example, my request for \holdinginserts was added because that was trivial to implement, but the suggestion for providing \reconsiderparagraph (to allow undoing the paragraph breaking to re-typeset it under different conditions) was rejected, as it would have meant more drastic updates to the paragraph-breaking algorithm.

This new version of \TeX was called \TeX 3.0, and shortly afterwards Don publicly announced that there would be no further version of \TeX (except for bug fixes) and that his involvement in any future development of typesetting engines has ended with that version [20]. That announcement prompted me to analyze \TeX’s abilities compared to high quality hand-typeset documents resulting in the paper given at the conference in Texas.

While \TeX was thus officially frozen with version 3.0, other people started to build \TeX engine variants to resolve one or another issue. The most influential ones are briefly outlined below; a nice overview of the more complete picture is given in [40], from which Figure 1 was taken with kind permission.

In the following we only discuss the major developments that contributed in one way or the other to an enriched feature set of the engine or have been influential in other ways.

2.1 p\TeX

Typesetting in Japanese requires support of a huge character set and the ability to handle different typesetting directions. Thus early on developers in Japan created an extension to \TeX that supports both Kanji (two-byte fonts) and proper vertical typesetting, in addition to \TeX’s horizontally oriented approach. Early versions of p\TeX predate \TeX 3.0 [11, 32].

2.2 ML-\TeX

One of the earliest attempts to modify \TeX itself to handle the problem of multilingual typesetting was Michael Ferguson’s work on ML-\TeX. Amongst other things it added a \charsubdef primitive that provided substitutions for accented characters. This way \TeX’s hyphenation algorithm would be able to correctly hyphenate words with diacritics, even if the fonts used did not contain the characters as individual glyphs.

With the availability of T1-encoded fonts (containing most of the accented characters used in

Frank Mittelbach
“Western” languages as individual glyphs) ML-TeX was no longer necessary for these languages. Nevertheless, it is still available in most engines, but needs to be explicitly enabled on the command line.

2.3 \( \text{N}\text{T}\text{S}/\text{e\text{-}TEx} \)

The \( \text{N}\text{T}\text{S} \) project (New Typesetting System) was inaugurated by DANTE (the German TeX Users Group) in 1992. Its objective was to re-implement TeX in a 100% compatible way in Java. While TeX was frozen, \( \text{N}\text{T}\text{S} \) was to remain flexible and extensible. The project completed successfully in 2000, passing the trip test, and thus proving that a reimplementation of TeX in a different language was possible. As it turned out though, full compatibility with TeX resulted in code that was less modular than initially hoped for, so that adding any extensions or providing modifications of algorithms turned out to be far more difficult than initially anticipated.

For this and a number of other reasons, \( \text{N}\text{T}\text{S} \) itself wasn’t developed any further. \( \text{e\text{-}TEx} \) is a spin-off started around 2003 with the intention of developing a new Java-based system incorporating the experiences from \( \text{N}\text{T}\text{S}, \text{e\text{-}TEx}, \) \( \text{pdfTEx} \) and Omega. The project is represented on the web [1], but as of today it hasn’t left alpha stage.

2.4 \( \text{e\text{-}TEx} \)

\( \text{e\text{-}TEx} \) started out in 1992 as a project by Peter Breitenlohner reimplementing ideas by Knuth [15] for a bi-directional extension but avoiding the need for special DVI drivers. Ideas for additional extensions then were added, and in 1994 the first version of \( \text{e\text{-}TEx} \) was published.

Around that time members from the \( \text{N}\text{T}\text{S} \) team joined the effort and during 1994–98 \( \text{e\text{-}TEx} \) was run as an \( \text{N}\text{T}\text{S} \)-project in order to provide a small number of useful extensions to TeX to fill the gap while \( \text{N}\text{T}\text{S} \) was still under development. As it turned out, however, this set of extensions took on a life of its own and over time was incorporated into all major TeX-based engines. As a result, nowadays one can assume that all engines support the original TeX primitives plus the extensions offered by \( \text{e\text{-}TEx} \).

The new features offered by \( \text{e\text{-}TEx} \) are a number of additional programming primitives and better tracing facilities, support for mixed-direction typesetting, and an increase in the number of most register types. In the area of micro-typography enhancements, it offers a generalization of \texttt{orphanpenalty} and \texttt{widowpenalty} by supporting special penalty values for the first or last \( n \) lines. It also added a method to adjust the spacing in the last line of a para-graph to be close to that of the preceding line (instead of being set tight as standard TeX normally does).

2.5 Omega/Aleph

Omega, developed by John Plaice with Yannis Haralambous contributing ideas and developing fonts, was the first extension of the TeX program that supported Unicode instead of 8-bit input encodings. The driving force behind its development was to enhance TeX’s multilingual typesetting abilities and better support for complex scripts.

Aleph is a spin-off of Omega that was started to include \( \text{e\text{-}TEx} \) capabilities and stabilize the code base. Neither project is being developed any more, but most of Aleph’s and thus Omega’s functionality has been integrated into LuaTeX.

2.6 \( \text{pdfTEx} \)

The \( \text{pdfTEx} \) engine started as a Master’s thesis by H\(\text{\'a}n\) Th\(\text{\'e}\) Th\(\text{\'a}n\)h in the mid-nineties and initially offered PDF output, support for embedded Type 1 fonts, virtual fonts, hyper-links, and compression.

For his PhD thesis [37], H\(\text{\'a}n\) Th\(\text{\'e}\) Th\(\text{\'a}n\)h experimented with various micro-typography algorithms including the hz approach [42] and several of them were implemented in \( \text{pdfTEx} \) [38, 39].

Today, \( \text{pdfTEx} \) (with the \( \text{e\text{-}TEx} \) extensions included) is the dominant TeX-based engine in practical use, i.e., all major distributions use this program as the default TeX engine.

2.7 \( \text{XeTEx} \)

\( \text{XeTEx} \) is one of the more recent additions to the TeX engine successors [6]. It was created by Jonathan Kew and provides as one of its major distinguishing features extensive support for modern font technologies such as OpenType, Graphite and Apple Advanced Typography (AAT). It can make direct use of the advanced typographic features offered by these font technologies, such as alternative glyphs, swashes, optional ligatures, variant weights, etc. These fonts can be used without the need for configuring TeX font metrics for them.1

\( \text{XeTEx} \) natively supports Unicode both for input (UTF-8 encoding) as well as for accessing font glyphs. It can also typeset mathematics using Unicode fonts such as Cambria Math or Asana Math [3], provided they contain special mathematical features.

1. The downside of this is that it can’t be guaranteed that the formatting of a source document does not change over time (when libraries are updated on the host system) and there is no way to freeze all components of a document, as is possible with traditional TeX.
2.8 Lua\TeX

Lua\TeX\ made its first public appearance in 2005 at the TUG conference in China, as a version of pdf\TeX\ with an embedded Lua scripting engine. The first public beta was presented in 2007. It is being developed by a core team of Hans Hagen, Hartmut Henkel and Taco Hoekwater [4].

Important project objectives are merging of engines (combining ideas from Aleph and pdf\TeX), support for OpenType fonts, and access to all \TeX\ internals from Lua. Through various callbacks it is possible to hook into \TeX\’s typesetting algorithms and adjust or even replace them.

2.9 iT\TeX

Finally, as the ultimate successor engine we have or will have iT\TeX, a fictitious XML-based successor to \TeX\ announced by Donald Knuth at the TUG 2010 conference in San Francisco [14]. According to its author this program will resolve all issues related to high quality typesetting, including those that aren’t yet discovered—we can only hope that it doesn’t take Don too much time to finish it.

3 Review of the issues raised in 1990

With the knowledge of what today’s successors of the \TeX\ engine are capable of, we are now ready to re-analyze the issues discussed two decades ago and evaluate which of them are nowadays:

1. resolved (best case, denoted by \Large $\triangleright$ below), or
2. could now be resolved using the improved features of modern engines (hopeful case, denoted by \Large $\triangleright$ below), or
3. is still out there waiting for a resolution (bad case, denoted by \Large $\triangleright$ below).

We follow the order of the original paper (see Figure 2) to help people looking up additional details on the problems outlined. It is available in facsimile on the web [26].

3.1 Line breaking

\TeX\’s line-breaking algorithm is clearly a central part of the \TeX\ system. Instead of finding breaks line by line, the algorithm regards paragraphs as a unit and searches for an ‘optimal solution’ based on the current values of several parameters. Consequently, a comparison of results produced by \TeX\ and other systems will normally favor \TeX\’s methods.

Such an approach, however, has its drawbacks, especially in situations requiring more than block-style text with a fixed width.

\Large $\triangleright$ Issue: No post-processing of final lines based on their content

The final line breaks are determined at a time when information about the content of the current line has been lost (at least for the eyes of \TeX, i.e., its own macro language), so that \TeX\ provides no support for post-processing of the final lines based on their content.

For example, the last line of a paragraph is usually typeset using normal spacing between words, even if the previous line has been set loosely or tightly. (\TeX\ can now handle this to some extent, with its \texttt{\texttt{\texttt{lastlinefit}} primitive.)

Another example is that the tallest object in a line determines its height or depth so that lines might
get spread apart, even if they would fit perfectly.

In theory these issues could now be catered to with the LuaTex program, because it offers the ability to post-process the lines and modify the final appearance.

Issue: No way to influence the paragraph shape with regard to the current position on the page

TeX and all its successors break paragraph text into lines at a time where they do not know where this text will eventually appear on a page. Consequently, there is no possibility within the model of catering to special paragraph shape requirements based on that position.

The only way to work around this is a complex feedback loop, using placement information from a previous run to calculate the necessary \parshape. Because this requires multi-pass formatting (with many passes), it is impractical. A simpler, though still complicated, approach is to assume a strict linear formatting, in which case one can build the paragraph shapes one after the other.

A new approach, that we are currently exploring for LaTeX, involves storing paragraph data in a data structure that allows re-breaking the material for trial typesetting. This is outlined in Section 4.

3.1.1 Line-breaking parameters

While the algorithm provides a wide variety of parameters to influence layout, some important ones for high-quality typesetting are missing. To resolve some of these issues, we need only (slightly) modify or extend the current algorithm. For others, serious research is required just to understand how a solution might be approached.

None of the engines has modified the TeX algorithm, so all of the problems are still unsolved. LuaTeX offers a way to replace the whole algorithm, but for most of these problems, that would be overkill, because it would require reprogramming everything from scratch in Lua.

Issue: Zero-width indentation box

When TeX breaks text into individual lines it discards whitespace and kerns at both sides of each line except for the first. On the left side of the first line (in left-to-right formatting) the existence of the paragraph indentation box prevents this from happening. Normally this is not noticeable, but in the case of layouts without paragraph indentation it can lead to problems, e.g., when \mathsurround has a positive value.

In LuaTeX this could now be resolved by defining code that preprocesses the paragraph material and removes discardable items following the indentation box.

Issue: Managing consecutive hyphens in a general way

In TeX it is possible to discourage two consecutive hyphens, but there is no way to prohibit or strongly discourage three or more. Technically, this would mean a slight extension of the current algorithm by keeping track of the number of hyphens in a row. None of today’s engines supports that concept.

Issue: Only four types of line quality

To implement good-looking paragraphs, TeX classifies each line into one of four categories based on the line’s glue setting (tight, decent, loose, very loose). It then uses that classification to avoid abrupt changes in spacing (if possible). However, the small number of classes results in grouping of fairly incompatible settings in a single class (especially, loose and very loose are affected). Technically, it would be simple to extend the number of classes to support better granularity.

Issue: Rivers and identical words across lines

If interword spaces from different lines happen to fall close to each other, they form noticeable stripes (rivers) through the paragraph that can be quite disconcerting. TeX’s line-breaking algorithm is unable to detect such situations. Resolving this would require serious research into the question on how to detect rivers and how to classify the “badness” of different scenarios in order to programmatically handle it through an algorithm.

A somewhat related issue (but rather easier to resolve) is the placement of the same word at the same position in consecutive lines, especially at the beginnings of lines, which is likely to disrupt the reading flow.

3.2 Spacing

Micro-typography deals with methods for improving the readability and appearance of text; see for example [5]. While TeX already does a great job in this area, some of the finer controls and methods are not available in the original program.

However, most of them have been implemented in some of the successor engines and an interface for BTeX to these micro-typography features is provided through the package microtype [33].

Issue: No flexible interword spacing

In order to produce justified text, a line-breaking algorithm has to stretch or shrink the interword
space starting from some optimal value (e.g., given by the font designer) until the final word positions are determined. \TeX{} has a well-designed algorithm to take such stretchability into account. It can also alter spacing depending on the character in front of the space to change the behavior after punctuation, for example.

There is no provision, however, for influencing the interword spaces in relation to the current characters on both word boundaries. Ideally, shrinking or stretching should depend on the character shapes on both sides of the space as exemplified in Figure 3.

None of the \TeX{} successors provides any additional support for controlling the interword spacing above and beyond \TeX{}’s capabilities. But with \LTEx{}’s callback interfaces it is possible to analyze and modify textual material just before it is passed to the line-breaking algorithm. This allows for ways to resolve this issue either as a table-based solution (one size fits all), or on a more granular level where the chosen adjustments are tied to the current font.

\begin{itemize}
\item[\fbox{Issue: No flexible intercharacter spacing}]
Instead of, or in addition to, stretching or shrinking the interword spaces to produce justified text, there are also the methods of \textit{tracking} (increasing or decreasing inter-letter spaces) and \textit{expansion} (changing the width of glyphs). There are debates by designers whether such distortions are acceptable approaches, but there is not much doubt that, if used with care and not excessively, they can help to successfully resolve difficult typesetting scenarios.\textsuperscript{2}

\LTEx{} provides both methods, the latter by implementing a version of the \textit{hz} algorithm originally developed by Hermann Zapf and Peter Karow \cite{42}.
\end{itemize}

\begin{itemize}
\item[\fbox{Issue: No native support for hanging punctuation}]
Don Knuth \cite[pp. 394–395]{18} gave an example of how to achieve hanging punctuation but it required the use of specially adjusted fonts and it also interfered with the ligature mechanism. In other words, it is only a partial solution for restricted scenarios.

Fortunately, a fully general solution was implemented in \pdfTeX{} and later also incorporated into \LuaTeX{}, so nowadays this can be considered resolved. The remainder of the article is typeset using hanging punctuation to allow for a comparison.

\section{Page breaking}

In 1990 I wrote “The main contribution of \TeX{} 82 to computer-based typesetting was the step taken from a line-by-line paragraph-breaking algorithm to a global optimizing algorithm. The main goal for a future system should be to solve the similar, but more complex, problem of global page breaking”. Fortunately in the \TeX{} world no serious attempt was made since then to address the fundamental limitation in \TeX{}’s algorithm, let alone designing and implementing a globally optimizing page-breaking algorithm.\textsuperscript{3}

\begin{itemize}
\item[\fbox{Issue: \TeX{} generates pages based on precompiled paragraph data}]
This issue describes the fundamental problem in \TeX{}’s approach: the program builds optimized paragraph shapes without any knowledge about their final placement on a page. The result is a “galley” from which columns are cut to a specified vertical size. A consequence of this is that one can’t have the shape of a paragraph depend on its final position on the page when using \TeX{}’s page builder algorithm.

To some extent, it is possible to program around this limitation, e.g., by measuring the remaining space on a page and explicitly changing paragraph shapes after determining where the current textual material will finally appear. However, besides being complicated to implement, it requires accounting for all kinds of special situations that normally would be automatically managed by \TeX{}, and providing “programmed” solutions for them.

As a result, all attempts so far to provide such functionality had to impose strong limitations on the allowed input material, i.e., they worked only in restricted setups and even then, the results were often not satisfactory.
\end{itemize}

\textsuperscript{2} On page 54 one paragraph was typeset with a negative expansion of 3\% to avoid an overfull line. See if you can spot it without peeking at the end of the article where we reveal which it was.

\textsuperscript{3} In the wider document engineering research community some research was carried out in the last thirty years, e.g., \cite{7,9,22,41}, but so far none has led to a production system.
Issue: Paragraphs already broken into columns can’t be reformatted based on page/column break decisions.

The main operations possible in TeX’s box/glue/penalty-model are shown in Figure 4. All macro processing that acts on the level of tokens (characters/symbols, spaces, etc.) is only possible before TeX builds the so-called “unset horizontal lists” in which character tokens change their nature into glyphs from fonts. From that point on, the manipulation possibilities are reduced to the level of box manipulations that only allow relatively few actions, such as removal of the last item in a box. However, those operations have severe limitations, e.g., one can’t remove glyphs from a horizontal list nor is there any possibility to convert the data back to character tokens, etc. that could be directly reprocessed by the macro processor.

The moment TeX turns an “unset horizontal list” into an “unset vertical list”, i.e., when it applies line breaking, we move to the bottom half of the model and from there, there is no fully general way...
to get back to the upper half. At the line breaks, we potentially lose spaces that can’t be recovered. Thus, it is not possible to reconstruct the original “unset horizontal list” even if we would recursively take off items from the end of the “unset vertical list” in the attempt to reassemble it.

As a consequence it is not possible to safely reuse textual material once it has been manipulated by \TeX’s paragraph builder. Instead one needs to find a way to record the “unset horizontal list”. That this is easily possible in Lua\TeX{} (but also in standard \TeX{} with somewhat more effort) will be demonstrated in Section 4 on page 59, which is the reason why we give this issue a combined I□ I□ rating.

### 3.4 Page layout

For the tasks of page makeup, \TeX{} provides the concept of output routines together with insertions and marks. The concepts of insertions and marks are tailored to the needs of a relatively simple page layout model involving only one column output, footnotes, and at the most, simple figures once in a while. The mark mechanism provides information about certain objects and their relative order on the current page, or more specifically, information about the first and last of these objects on the current page and about the last of these objects on any of the preceding pages. However, being a global concept only one class of objects can take advantage of the whole mechanism.\footnote{The term ‘one column output’ means that all text is assembled using the same line width. Problems with variable line width are discussed in Section 3.3. Of course, this already covers a wide range of possible multi-column layouts, e.g., the footnote handling in this article. But a similar range of interesting layouts is not definable in \TeX{}’s box-glue-penalty model.}

\[\text{I□ I□} \text{ Issue: Only a single type of marks is fully supported}\]

In the original paper, I suggested extending this to support multiple independent mark registers; that idea was later implemented in the \TeX{} program. As it turned out, however, this did not really solve the issue. Whenever the page layout gets more complicated and output routines are used to inspect the current state without actually shipping out pages in a linear fashion, the information maintained in \texttt{\baselineskip} is always lost.

In the end, we abandoned the whole mechanism for \LaTeX{} and used only \TeX{}’s \texttt{\botmark} register to put marks on the page and kept track of all other information (class of mark and content of the mark) externally \footnote{The \LaTeX{} implementation provides an extended mark mechanism with two kinds of independent marks with the result that one always behaves like a \texttt{\firstmark} and the other like a \texttt{\botmark}. The information contained in the primitive \texttt{\topmark} is lost.}. This solves the issue, but at a fairly high programming cost with complex data management.

\[\text{I□ I□} \text{ Issue: Missing built-in support for complex float management}\]

Float placement across different types of publications is governed by rules of high complexity: placement options may depend on aesthetic requirements, captions and legends might require different formatting depending on placement, positioning of floats may influence options available for other floats, etc.

Unfortunately, out of the box, \TeX{} offers only a simplistic mechanism derived as an extension to the footnote concept. \LaTeX{} extended this to a slightly more flexible algorithm but only with respect to supporting different classes of floats (where the floats of one class have to stay in sequence) and by adding a few parameters to add limits for the number of floats in a float area or the maximum size of an area. More complex rules or arrangements with varying formatting depending on placements, support for floats across multiple columns (other than a simple two-column mode) are not supported.

To some extent, this is not that surprising, because codification of placement rules and effective algorithms for computing complex layouts is an area that is not well understood, and at the same time hasn’t attracted much attention by the research community. Only a handful of publications in three decades approach one or another aspect of this topic \footnote{A simplistic mechanism derived as an extension to the footnote concept. \LaTeX{} extended this to a slightly more flexible algorithm but only with respect to supporting different classes of floats (where the floats of one class have to stay in sequence) and by adding a few parameters to add limits for the number of floats in a float area or the maximum size of an area. More complex rules or arrangements with varying formatting depending on placements, support for floats across multiple columns (other than a simple two-column mode) are not supported.}

\[\text{I□ I□} \text{ Issue: No conceptual support for baseline to baseline spacing}\]

For designers, \TeX{}’s way of specifying interline glue is a rather foreign concept; they typically use baseline-to-baseline spacing instructions in their specifications. Unfortunately, those prescriptions are not directly possible in \TeX{} because of the way \TeX{} determines the “current” \texttt{\baselineskip} value; see Figure 5 on the next page. Only with rigorous control on the
To implement a baseline to baseline dimension, for example between a paragraph and a heading (denoted by the question mark), the value for \baselineskip has to be determined depending on the \baselineskip of the second paragraph. Unfortunately, the value of \baselineskip used will be the one current at the end of the second paragraph while the \parskip has to be computed at its beginning.

programming level (i.e., by preventing the users and package writer from accessing the full power of \TeX) can one provide an interface supporting baseline-to-baseline spacing specification.

Because the interline glue concept is deeply buried in \TeX algorithms, it is not surprising that none of today’s engines (including \LaTeX) addresses this area.

\textit{Issue: No built-in support for grid-based design} \TeX’s concepts that conflict with baseline to baseline spacing also make grid-based design difficult, as this strongly depends on text baselines appearing in predictable positions.

Nevertheless, over the last three decades there have been a number of attempts to provide support for grid based-design on top of standard \TeX. None of them has been particularly successful — due to the missing support from the engine, all of them worked only with subsets of the many \TeX packages. To make things work, it is necessary to adjust behavior of various commands, and thus any package not explicitly considered is a likely candidate for disaster.

The \texttt{xor} package for \LaTeX3 offers a structured interface on the code level for grid-design. Unfortunately, it isn’t production ready and awaits a major refactoring based on the new \texttt{expl3} language for \LaTeX3. But even then it would suffer from the limitations listed above as long as it is used together with \LaTeX2 packages that do not interface with its grid support.

From an engine perspective, \LaTeX{} is the only engine that offers some additional possibilities that may help with grid design, through additional hooks provided, and through access to the internal \TeX data structures. If and how this could be usefully deployed remains to be seen. To my knowledge, no Lua\TeX-specific code for grid design yet exists.

### 3.5 Penalties — measurement for decisions

Line and page breaks in \TeX are determined chiefly by weighing the “badness” of the resulting output and the penalty for breaking at the current point. This works very well in most situations but there is one severe problem with the concept of penalties.

\textit{Issue: Consecutive penalties behave as }\min(p_1, p_2)\textit{ }

The problem is that an implicit penalty (e.g., for discouraging but not prohibiting orphans or widows) will always allow a break at this particular point even if an explicit penalty by the user attempts to disallow a break there. Changing the algorithm to use \max(p_1, p_2) instead would resolve the problem. With this approach an explicit breakpoint could still be inserted by interrupting the sequence of consecutive penalties, e.g., through \texttt{\kern0pt}.

With \Lua\TeX this could probably be implemented, but would most likely require very complicated parsing and manipulation of internal \TeX data structures, unless \Lua\TeX itself gets extended, i.e., by making the code that handles consecutive penalties directly accessible.

### 3.6 Hyphenation

When typesetting text, especially in narrow columns, hyphenation is often inevitable in order to avoid unreadable, spaced out lines. \TeX’s pattern-based hyphenation algorithm [23] is quite good at identifying correct hyphenation points to avoid such situations.

However, hyphenation is a second-best solution and, if applied, there are a number of guidelines that an algorithm should follow to improve the overall result. Several of them cannot be specified with \TeX’s algorithm. In fact, all of the ones below are unresolved with today’s engines, if we disregard that in \Lua\TeX one could in principle replace the full paragraph-breaking algorithm with a new routine.

\textit{Issue: Prevent more than two consecutive hyphens} Hyphenation of two consecutive lines is controlled by the algorithm (\texttt{\doublehyphenpenalties}),

7. The badness is a function of the white space used in a line in comparison to the optimal amount, e.g., if the space between words needs to stretch or shrink, the badness of the solution increases.
but there is no possibility of avoiding paragraphs like the current one, in certain circumstances. As one can easily observe, the number of hyphens in this paragraph has been artificially increased by setting relevant line-breaking parameters to unusual values. In languages that have longer average word lengths than English, such situations present real-life problems.

\begin{itemize}
\item Issue: Assigning weights to hyphenation points
\end{itemize}

TeX’s hyphenation algorithm knows only two states: a place in a word can or cannot act as a hyphenation point. However, in real life certain hyphenation points are preferable over others. For example, “Nonnenkloster” (abbey of nuns) should preferably not be hyphenated as “Nomenenko-ster” (as that would leave the word “nun’s toilet” on the first line).

\begin{itemize}
\item Issue: More generally, support other approaches to hyphenation
\end{itemize}

Liang’s pattern-based approach works very well for languages for which the hyphenation rules can be expressed as patterns of adjacent characters next to hyphenation points. Such patterns may not be easy to detect but once determined they will hyphenate reasonably well. For the approach to be usable, the necessary set of patterns should be be reasonably small, as each discrepancy needs one or more exception patterns with the result that the pattern set would either become very large or the hyphenation results would have many errors.

To improve the situation for the latter type of languages one would need to implement and potentially first develop other types of approaches. For now Liang’s algorithm is hardwired in all engines, though in theory LuaTeX offers possibilities of dropping in some replacement.

\subsection*{3.7 Box rotation}

TeX’s concept of document representation is strictly horizontal and left-to-right oriented. Any further manipulation is left to the capabilities of the output device using \special commands in the language of the device.

\begin{itemize}
\item Issue: No built-in support for rotation
\end{itemize}

Because of the lack of a common interface for such operations, any document making use of \special commands is processable only in a specific environment, so that exchange of such documents is only possible in a limited fashion.

With the event of \LaTeX\ 2ε the \LaTeX\ project team resolved this issue by providing an abstract interface layer that (in the form of the \graphics and \color packages) hides the device peculiarities internally so that documents using these interfaces became portable again.

\subsection*{3.8 Fonts}

\begin{itemize}
\item Issue: Available font information (non-issue)
\item Issue: Encoding standardization
\end{itemize}

In the Texas paper, I suggested that additional font characteristics should be made available as font parameters to enable smarter layout algorithms. Looking at this from today’s perspective I think it was largely a misguided idea, at least until recently. Many of the fonts that have been made accessible to the \TeX\ engines in the last decades (mainly PostScript Type 1 fonts) do indeed have various additional attributes defined by their designers but alas with largely non-comparable values between different fonts making any interpretation difficult if not impossible.

With OpenType fonts, this may change again, and engines like Xe\TeX\ or Lua\TeX\ allow access to such additional attributes.

\begin{itemize}
\item Issue: Encoding standardization
\end{itemize}

By default, \TeX\ translates the input encoding (representation of characters in the document) one to one into the output encoding (glyph positions in the current font). E.g., if you put a “b” into your document then this is understood as “typeset the character in position 96 (ASCII code for ‘b’) in the current font”. This tight coupling between encoding of data in different places required that fonts used by \TeX\ always stored the glyphs in the same position (which they did only partially) or that the user understood the subtle differences and adjusted the document input accordingly. For example, in plain \TeX\ the command \texttt{$\&$} produces a $-sign — unless you are typesetting in Computer Modern Text Italic, in which case your output suddenly shows £-signs.

With more and more fonts (with different font encodings) becoming available and \TeX\ entering the 8-bit world (with numerous input encodings interpreting the document source differently), such issues got worse.

These problems were resolved for \LaTeX\ through three developments. Don Knuth developed the idea of virtual fonts \cite{13} and that concept was quickly adopted by nearly all major output-device drivers, so that it became usable in practical terms.\footnote{In 1990 I expressed my hope that these ideas would help to simplify matters \cite[p. 342]{26} and as it turned out, that was indeed the case.} With this concept available the \TeX\ community agreed on a special virtual font encoding \cite{2}. Finally we designed and implemented the \LaTeX\ Internal Char-
character Encoding for \texttt{E-\LaTeX} 2ε (LICR) \cite[chap. 7]{ LICR} that transparently maps between different input encodings and arbitrary font encodings. This is perhaps not a perfect solution, but for the 8-bit world it effectively resolved the issues.

Unicode support was also addressed (through the \texttt{inputenc} package) but here better support from the engines is required to come to a fully satisfactory solution. As mentioned above, explicit Unicode support was first added by Omega, and from there made its way into Aleph and Lua\LaTeX. Xe\LaTeX also natively supports Unicode.

\begin{itemize}
\item \textbf{Issue: Ligature and kerning table manipulation}
\end{itemize}

In \LaTeX, ligatures and kerns are properties of fonts, i.e., they apply to all text in a document. However, different languages use different rules about what to apply or not to apply in this case. Thus to model this in \LaTeX, one would need to define private fonts per language, each differing only in their ligature and kerning tables. While this would be a theoretical possibility, in practical terms it would be a logistical nightmare and so nobody has ever tried to implement such fine points of micro-typography.

With pdf\LaTeX this situation changed somewhat as pdf\LaTeX supports suppressing all ligatures or all ligatures that start with a certain character. This alone does not help much though, as it does not allow, for example, prohibiting the “ff” ligature (not used in the German language) while allowing for the other ligatures starting with “f”, nor does it support implementing new ligatures, such as “ft” or “ck”, through negative kerning.

Lua\LaTeX takes this a huge step forward and provides the necessary controls to improve the situation considerably. While I was writing this article (and asking around), the first experimental packages started to appear, e.g., \texttt{schnellig} \cite{ schnellig} by Mico Loretan, so it will be interesting to see what happens in the near future.

\subsection{3.9 Tables}

\begin{itemize}
\item \textbf{Issue: Combining horizontally- and vertically-spanned columns is impossible}
\end{itemize}

\LaTeX’s input format is inherently linear, and so it is not surprising that any \LaTeX interface to inherently two-dimensional table data is somewhat limited. Out of the box \LaTeX supports column formatting, e.g., it can calculate the necessary column width and apply a default formatting per column. It also allows for horizontally-spanned cells with their own formatting. However, there is no provision for providing cells whose content is able to reflow depending on the available space nor is there any mechanism to provide vertically-spanned cells. Both are essential formatting requirements.

\LaTeX offers a higher-level document syntax to the low-level capabilities that \LaTeX provides, and over time, many packages appeared that enhanced the solution in one way or the other. However, without any underlying direct support for the more complex concepts all these efforts show limitations and are often difficult to use. So far none of the existing engines addresses this area.

\subsection{3.10 Math}

Mathematical typesetting is one of \LaTeX’s major domains where—even today after thirty years—no other automatic typesetting system has been able fully to catch up. But even in this area several things could be improved.

\begin{itemize}
\item \textbf{Issue: Some mathematical constructs are not naturally available in \LaTeX, e.g., double accents, under-accents, equation number placement, \ldots}
\end{itemize}

For many of these problems workarounds have been implemented as (fairly complex) plain \LaTeX macros by Michael Spivak in the \texttt{AMS-\LaTeX} format \cite{ Spivak, AMS}. Most of this code plus further extensions were ported to \texttt{E-\LaTeX} and are nowadays available as the \texttt{E-\LaTeX} package \texttt{amsmath}.

For this reason this issue can be largely regarded as solved, even though native support for most of these constructs would improve the situation further.

\begin{itemize}
\item \textbf{Issue: Spacing rules and parameters are all hardwired in the engine or the math fonts}
\end{itemize}

\LaTeX’s spacing rules for math are quite good, but in cases where they needed adjustments, it was either impossible or quite difficult to do. This restriction has been finally lifted with Lua\LaTeX, because that engine offers access to all internal parameters of \LaTeX.

\begin{itemize}
\item \textbf{Issue: Sub-formulas are always typeset at their natural width}
\end{itemize}

No engine so far provided any alteration to the core algorithms of \LaTeX that format a formula. Thus sub-formulas (for example from $\left\lceil \text{left} \ldots \right\rceil$) are still boxed at natural width even if the top level math-list is subject to stretching or shrinking. This also means that there is no way to automatically break such constructs across lines.

\begin{itemize}
\item \textbf{Issue: Line breaking in formulas (not listed in original paper)}
\end{itemize}

Don declared line breaking in math too hard for \LaTeX to do automatically and as a result countless users struggled with manually formatting displayed
equations to fit a given measure. For a long time it looked as if that problem was indeed too difficult to tackle within the constraints of a formatting engine. However, in the late nineties Michael Downes proved everybody wrong by designing and implementing a first version of the \texttt{breqn} package for \LaTeX{}.

This package—further improved by Morten Høgholm after Michael’s untimely death—gets us already quite a way toward the goal of high-quality automatic line breaking in formulas [10]. However, with the increased processing power now available and with Lua\TeX{}’s access to \TeX{} internals, it should be possible to solve most or all of the remaining problems identified.

### 3.11 \TeX{}’s language

In 1990 I made the bold statement: “\TeX{}’s language is suitable for simple programming jobs. It is like the step taken from machine language to assembler”. Since then the new engines added one or the other primitive, but with the exception of Lua\TeX{}, which added an interface to Lua as an additional programming language, the situation hasn’t improved with respect to core engine support.

\begin{itemize}
  \item \textbf{Issue: Incompleteness with respect to standard programming constructs}

While this statement is still true with respect to most engines, the situation has nevertheless improved. With expl3 the \LaTeX{} Project team has provided a programming layer that offers a large set of data types and programming constructs. The development of expl3 started in fact long ago: initial implementations date back to 1992. However, back then, the processing power of machines was not good enough to allow executing code written in expl3 at a reasonable speed. For that reason, the ideas and concepts were put on the shelf, and the project team instead concentrated on providing, and later on maintaining, \LaTeX{}\ 2e.

Since then processor speed has increased tremendously, and as a result it became feasible to finally use the ideas that had been initially developed nearly two decades ago. The core of expl3 has been reimplemented (again) and its stable release is now gaining more and more friends—it even got its own logo designed as shown in Figure 6.

\begin{itemize}
  \item \textbf{Issue: A macro language . . . good or bad? / Difficulty of managing expansion-based code}

Since the first implementation of \TeX{} people have voiced their concern about the fact that the \TeX{} language is a macro language that works by expansion and not a “proper” programming language. However, despite this grumbling, nobody came up with a workable alternative model that successfully combines need for simple and easy input of document material (which makes up the bulk of a \TeX{} document) and the special needs of a programming environment that avoids the complexity of programming by macro expansion (which indeed becomes complex and difficult to understand if used for non-trivial tasks).

That the \TeX{} language can be used to produce truly obfuscated code was nicely demonstrated by David Carlisle’s seasonal puzzle [8] which is worth taking a look at, but even normal coding practice will easily lead to code that is difficult to understand (and to maintain) as demonstrated by many of today’s packages. Part of the reason for this is that all coding practice around \LaTeX{}2e (and other macro formats) is based on concepts and ideas originated in plain \TeX{}, with more and more complexity layered on top but without fundamentally questioning the core approach which was never intended for complex programming tasks.

So why \expl{}? Largely because with expl3 we now have a foundation layer available, that — while still based on macro expansion — provides a comfortable programming environment. From the engine side Lua\TeX{} nicely sidesteps the question by providing a separate programming language in addition.

\begin{itemize}
  \item \textbf{Issue: Inability to access and manipulate certain internal \TeX{} data structures}

Many of \TeX{}’s internal data structures are inaccessible to the programmer or only observable indirectly.\footnote{In the 1990 paper I gave the example of measuring the length of the last line in a paragraph by artificially following it by an invisible displayed formula as only within such a display is the desired information made available.}

Thus, whenever an adjustment to one of \TeX{}’s internal algorithms is needed it becomes necessary to bypass the algorithm completely and reimplement it on the macro level. This means accepting huge inefficiencies and in many cases makes such implementations unusable in real-life applications.

Frank Mittelbach

\begin{figure}[h]
\centering
\includegraphics[width=0.3\textwidth]{expl3}
\caption{The expl3 logo (courtesy of Paulo Cereda)}
\end{figure}
For more than two decades this was the situation with all engines. Finally Lua\TeX{} broke this restriction by offering access to all (or nearly all) internals including the ability to modify them (with the danger of breaking the system in unforeseen ways, but that comes with the territory).

\begin{itemize}
\item provide access and manipulation possibilities from the “mouth” to an intermediate data structure that holds character data plus attributes before they are turned into glyphs, or
\item provide additional manipulation possibilities of “stomach” material, or
\item offer conversion of typeset material back to token data similar to what \verb|\showbox| offers as symbolic information in the transcript file.
\end{itemize}

Sadly, none of the engines offers any direct support in this area. However, we will see that, with today’s increase in processing power, it becomes feasible to implement a strictly \TeX{}-based solution. This solution has some (acceptable) limitations for boundary cases, but a variant implementation using Lua\TeX{}’s callback interface can even get rid of those, as we’ll see in the next section. For this reason we give this issue a $\mathcal{F}$ rating today.

\section{Overcoming the mouth/stomach separation}

If we look back to Figure 4 on page 53 we can see that the best data structure available for use in trial typesetting is the “unset horizontal list”. The moment we apply line breaking we would lose information and if we store the information at an earlier stage (i.e., as token data) we would have to deal with the side effects of repetitive token processing.

Unfortunately, the “unset horizontal list” is not a data structure made available by \TeX{}. What is possible though (looking at the right side of the diagram), is to store it in a horizontal box. At a later stage this box can then be transformed back into an “unset horizontal list” and that could then be typeset into a “trial” paragraph shape.

However, simply storing the content of one or more paragraphs into an \verb|\hbox| for later processing is not a workable option either, because:

\begin{itemize}
\item \TeX{} applies some “optimizations” in restricted horizontal mode to save some processing time.\footnote{While such optimizations have been important at the time \TeX{} was originally developed, the speed gain they offer nowadays is negligible. What remains are inconsistencies in processing that should get removed in today’s engines such as pdf\TeX{} and Lua\TeX{}}
\item Under the assumption that text in an \verb|\hbox| cannot be broken into several lines, it drops all break penalties from in-line formulas (i.e., those normally added after binaries and relational symbols) and also doesn’t add any implicit \verb|\discretionary| hyphens in place of “\~”.
\end{itemize}

For the same reason it also ignores changes to \verb|\language|.

\item If we save each paragraph into one \verb|\hbox| then we effectively surround each paragraph by a \TeX{} group. Thus local changes, such as modifications to fonts or language would suddenly end at the paragraph boundary.

While these restrictions can be overcome, it means a far more elaborate approach needs to be taken.

\begin{itemize}
\item \TeX{} applies some “optimizations” in restricted horizontal mode to save some processing time.\footnote{While such optimizations have been important at the time \TeX{} was originally developed, the speed gain they offer nowadays is negligible. What remains are inconsistencies in processing that should get removed in today’s engines such as pdf\TeX{} and Lua\TeX{}}
\item Under the assumption that text in an \verb|\hbox| cannot be broken into several lines, it drops all break penalties from in-line formulas (i.e., those normally added after binaries and relational symbols) and also doesn’t add any implicit \verb|\discretionary| hyphens in place of “\~”.
\item If we save each paragraph into one \verb|\hbox| then we effectively surround each paragraph by a \TeX{} group. Thus local changes, such as modifications to fonts or language would suddenly end at the paragraph boundary.

While these restrictions can be overcome, it means a far more elaborate approach needs to be taken.

\end{itemize}
4.1 A standard \TeX solution

If storing the “unset horizontal list” directly in an $\texttt{\hbox}$ is not an option, what alternative is available according to our Figure 4? The only other path that results in an $\texttt{\hbox}$ is to transform the list into an “unset vertical list” and then remove the last box from that list (i.e., a circular path in clockwise direction around the center of the diagram). To make this work we have to overcome the limitations listed at various places along the path:

1. Transforming an “unset horizontal list” into an “unset vertical list” loses glues and penalties at line breaks.
2. Decomposition of the “unset vertical list” is not possible on the main galleys.
3. Decomposition of the “unset vertical list” stops at the first node that is not a box, glue, penalty, or kern item.

To alleviate issue 1 we will build our “unset vertical list” using the largest possible $\texttt{\hbox}$ available in \TeX. We remove the indentation box whenever we start a paragraph. In addition, we trap any forced break penalty, record it, and prematurely end the paragraph at this point to stay in control. As a result each $\texttt{\hbox}$ in the resulting “unset vertical list” will contain exactly the paragraph material from one forced break to the next (considering the paragraph boundaries as forced breaks).

What happens if the paragraph material exceeds the largest possible dimension available in \TeX? In that case (which means that the paragraph is noticeably longer than a page) we will end up with uncontrolled line breaks. In \TeX it is impossible to prevent this from happening, but at least it is possible to detect that it happened. One can then warn the user and request that the paragraph be artificially split somewhere in the middle using a special command.

Issue 2 is easily resolved: as we initially want to store the data, we can simply scan the material within a $\texttt{\hbox}$. This box, which is later thrown away, will form a boundary for local changes, but this is okay, as we can scan as many paragraphs as necessary in one go.\footnote{11} This would solve the problem if only portions of a document (e.g., float captions) are subject to trial typesetting. If the intention is to process the whole document in this manner then a slightly different approach is needed. In that case we would use the main vertical list for collection and devise a special output routine that is triggered at the end of each paragraph. The $\texttt{\hbox}$s holding the paragraph fragments would then be retrieved within the output routine. Once everything is collected as far as necessary, the output routine could then be changed to do trial typesetting.

To avoid issue 3 it is necessary to ensure that material from $\texttt{\insert}$ and $\texttt{\vadjust}$ is not migrated out of the horizontal material. If they were, they would appear after our “paragraph line”. On the one hand, this is the wrong place if we later rebreak the material into several lines, and on the other hand it would prevent us from disassembling the material and storing it away. Therefore the solution is to end the paragraph (generating one line for the current fragment that we can save away), then start an $\texttt{\hbox}$ into which we scan the $\texttt{\insert}$ or $\texttt{\vadjust}$ and then restart the scanning for the remainder of the “real” paragraph.

With these preparations, the algorithm then works as follows:

- When we start (or restart) scanning paragraph material we ensure that there is no indentation box at the beginning.
- When we reach the end of the paragraph (or a paragraph fragment where we have artificially forced a paragraph end) \TeX will typeset the material and because of the large line length it will (normally) result in a single-line paragraph. We then pick up this line via $\texttt{\lastbox}$ and repackage it by removing the glue (from $\texttt{\parfillskip}$) and penalty at its end. Then we save this box and an accessing function away in some data structure and restart scanning until we reach the end.
- If we see an $\texttt{\insert}$ or $\texttt{\vadjust}$ we interrupt and add the scanned material to the data structure. Then scan and store the vertical material as outlined above.
- If we see a forced penalty we interrupt and save the scanned material and then also record the value of the penalty in the data structure. Note that any non-forcing penalty could just be scanned as normal paragraph material because of the large $\texttt{\hsize}$.
- Once we are finished parsing we end up with a data structure that looks conceptually as follows:

\begin{verbatim}
\dobox box_1 \dopenalty \{10000\}
\dobox box_2 \doint {box_1}{box_2}
\dobox box_3 \dovadjust {box_1}{box_2}
\end{verbatim}

box$_1$ to box$_3$ are $\texttt{\hbox}$s holding paragraph text fragments; box$_1$ and box$_2$ are also $\texttt{\hbox}$s containing just an $\texttt{\insert}$ or $\texttt{\vadjust}$, respectively, i.e., they are generated basically by:

\begin{verbatim}
\setbox box_x = \hbox{\insert{...}}
\end{verbatim}

\footnote{11} The limit is available memory, which is huge these days.
With the right definitions for \dobox and friends (e.g., \unhcopy, etc.) this data structure can then be used to “pour” the saved paragraph(s) into various molds for trial typesetting.

This algorithm works with any \TeX engine and its only restriction is the maximum allowed size of a single paragraph. This is acceptable, as it normally would not happen unless somebody is typesetting a document in James Joyce style. If it does, it will be detected and the user can be asked to artificially split the paragraph at a suitable point.

4.2 A Lua\TeX solution

Using the Lua\TeX engine, it is possible to simplify the algorithm considerably. Lua\TeX offers the possibility of replacing the line breaking algorithm with arbitrary Lua code and we can use this fact to temporarily replace it with a very trivial function: one that simply packages the “unset horizontal list” into an \hbox and returns that. This would look as follows:

```lua
function hpack_paragraph (head)
    local h = node.hpack(head)
    return h
end

callback.register("linebreak_filter",
    hpack_paragraph)
```

The beauty of this is that it automatically resolves issues 1 and 3 listed above. Issue 1 is fully resolved, because node.hpack is able to build \hboxes of any size, even wider than \maxdimen (as long as you do not try to access the \wd of the resulting box). So even Joycean paragraphs are no longer any problem. Issue 3 is gone because we do not need to decompose material. With the simple code above any penalties, \inserts, or \vadjusts simply end up within the box; in contrast to the normal line breaking algorithm, the hpack_paragraph code does not touch them. For the same reason, we do not have to take off any \parfillskip from the end, as it isn’t added in the first place.

The only problem found so far is a bug in the current implementation of the linebreak_filter callback rendering \lastbox (which is needed by our algorithm) unusable the moment the callback is installed. This is due to some missing settings in the semantic nest of \TeX that are not fully carried out. For the same reason, we do not have to take off any \parfillskip from the end, as it isn’t added in the first place.

The task now is to put the new possibilities to use and work on solving the open questions with their help.

* * *

The author wants to thank Nelson Beebe, Karl Berry, and Barbara Beeton for their invaluable help in improving the paper through thorough copy-editing and numerous suggestions.

References


* * *

Finally, to answer the question posed in footnote 2: The paragraph typeset with negative expansion was the second one in Section 3.4. Without it, it would have looked like this:

The mark mechanism provides information about certain objects and their relative order on the current page, or more specifically, information about the first and last of these objects on the current page and about the last of these objects on any of the preceding pages. However, being a global concept only one class of objects can take advantage of the whole mechanism.

○ Frank Mittelbach
Mainz, Germany
frank.mittelbach (at) latex-project dot org
http://www.latex-project.org/latex3.html
Abstract
Here we introduce LuaJIT\TeX, an implementation of Lua\TeX that uses Lua\!JIT 2.0 instead of Lua 5.1.

1 Introduction
On Thursday, November 8, 2012 the long-awaited release of Lua\!JIT 2.0 finally happened, after 11 beta releases over three years. It happened a month after Euro\TeX 2012 & 6\textsuperscript{th} Con\TeXt meeting, where I and Hans Hagen discussed the possibility of building a set of bindings to shared libraries of general interest for Lua\TeX. A binding is an object module that acts as a bridge between a specific library and the Lua interpreter of Lua\TeX. Its role is to expose the library to the point of view of Lua(\TeX), thus easing the job of the programmer. The module is specific to the library, and its source code must be created in some way. A manual binding is feasible only for a small library; with a large library, it’s better to make use of dedicated tools. Each tool has its pros and cons, but we have found that SWIG\[11\] can satisfy our needs, having used it before (see [18] and [19]). Its syntax is quite similar to C and it can parse the header files of a library and automatically produce the source code of the binding. Usually the compilation of the module is also straightforward.

In a \TeX project we would like to satisfy the requirements of several platforms, each one with its own toolchain. A candidate library is not always available for a target platform, or complete support for a toolchain may be lacking (we need at least a compiler, assembler and linker). A library can use some “dirty tricks” that are hard to translate into a binding module and extensive testing can become prohibitive. Last but not least, during the meeting we also considered the consequences on the upcoming transition from Lua 5.1 to Lua 5.2 in Lua\TeX.

When we explore some ideas, sometimes we fall into what looks like an unsolvable problem, and it’s a good strategy to temporarily change focus and start a completely different activity and return after a while to the original problem with a fresh point of view. As it happens, I had such a problem with the binding of a function with a variable argument list (in an apparently absurd attempt to bind the libc library of Microsoft Windows 7), and the new release of LuaJIT offered a good reason to momentarily drop this task and start to see if it was possible to replace the Lua interpreter with a LuaJIT one. But before I go on, it’s necessary to understand what exactly LuaJIT is and why such a substitution may be interesting. I will first discuss the \TeX and Lua interpreters.

2 \TeX, Lua, Lua\!JIT
2.1 \TeX
Lua\TeX is the union of two interpreters, one of the Lua language and one for the \TeX language. The main actor is the \TeX interpreter [12]: the input processor scans each input line of the source producing a pair (character-code, category-code). Lua\TeX currently has $2^{21}$ character-codes and 16 category-codes. Starting from a pair, a character token, a control sequence token, or a parameter token is formed; there are currently around 350 subtypes of tokens. With an abuse of terminology, we can call this subtype an operation code (opcode), and hence an opcode fits into two bytes. A token is then executed by mean of a jump table using a function pointer: \((\text{jump\_table}[\text{opcode}])()\) calls the function that implements opcode. The statement while (1) means that this task continues until the variable main\_control\_state has the value goto\_return, that signals to exit from the main loop and end the program:

\begin{verbatim}
void main\_control(void) {
    main\_control\_state = goto\_next;
    init\_main\_control();
    if (equiv(every\_job\_loc) != null)
        begin\_token\_list(equiv(every\_job\_loc),
                           every\_job\_text);
    while (1) {
        if (main\_control\_state == goto\_skip\_token)
            main\_control\_state = goto\_next;
        else
            get\_x\_token();
        if (interrupt != 0 && OK\_to\_interrupt) {
            back\_input();
            check\_interrupt();
            continue;
        }
        if (int\_par(tracing\_commands\_code)> 0)
            show\_cur\_cmd\_chr();
        (jump\_table[(abs(mode)+cur\_cmd)])();
        if (main\_control\_state == goto\_return) {
            return;
        }
    }
    return;
}
\end{verbatim}

This kind of interpreter is called a Syntax-Directed Interpreter because it mimics what we do when we trace the code manually. It is well suited for a DSL (Domain Specific Language, see [16]) as is

Luigi Scarso
\LaTeX{} in this case, but usually a DSL is not so complex as \LaTeX{} (which is also Turing-complete). The main part of this kind of interpreter is usually a big switch-case statement, where each opcode has its own case. The C standards do not specify how to implement a switch statement, but usually a compiler can use jump_table only if each label is equal to the preceding label plus one (or if the compiler is able to bring values of the labels to an equivalent case, see [1], section 7.12 Branches and switch statements); if the values are far from each other, a compiler must implement it as a kind of binary search among if-then-else like statements, and this has a bad impact on the branch predictor of the CPU. For example, let’s consider this switch fragment of C code:

```c
switch (OPCODE) {
    case 0: func_000(); break;
    case 1: func_001(); break;
    case 2: func_002(); break;
    default: break;
}
```

A compiler, maybe with some kind of optimization enabled, can generate machine code corresponding to the following pseudo-code (not C):

```
static address jump_table[] =
    {case_0, case_1, case_2, end};
if (index > 2) goto end;
goto jump_table[index];
case 0: func_000(); goto end;
case 1: func_001(); goto end;
case 2: func_002();
end:
```

which is more efficient than multiple if-then-else. Note that this jump_table is not the same as we’ve seen for \LaTeX{}: there we use a label to run a function in the compiled code that refers to a jump_table being a function pointer table defined earlier in the source code, and once compiled it adds overhead due to the call of the selected function. On the other hand, the function pointer method definitely avoids the if-then-else like statements (because it’s a choice made by the programmer, not the compiler) and makes the code more compact and more manageable. More on this later.

It is known that in order to speedup the loading of a large set of macros, \LaTeX{} (or, more exactly, ini\LaTeX, a special version of \LaTeX{}) can also dump the macros into a kind of memory-compiled format (simply called format), which can be loaded at runtime. A format depends on the current release of the interpreter and the machine on which the interpreter runs: a format cannot generally be exchanged between different releases and formats cannot always be exchanged between different machines even with the same release — but this currently fails only if a format uses floating point values, because floating point numbers related to glue are stored in the format and hence will generally not be readable across platforms. (See [2]: \ttex, for example, doesn’t use glue values in the format and hence the result .fmt is portable, thankfully.)

This is not seen as a penalty: because \LaTeX{} is used as document compiler speed is a concern, and a format can give a speedup of several orders of magnitude, so it’s typically built when \LaTeX{} is installed. \LaTeX{} users are generally more interested in durability/portability of their document source code, not the format. A remarkable exception is Con\LaTeXt Mark IV, but its users find it natural to rebuild the format on every update of the code — which happens quite often, because it’s still evolving. Dumping a format is also an uncommon characteristic for a DSL language.

### 2.2 Lua

The Lua interpreter is designed in a different way: it first translates the source code into another form and then executes this form. The translation is called compiling into bytecode because it’s similar to the task of a compiler, which translates source code (like a C program) into machine code. While a compiler usually translates a source program into an intermediate representation which is optimized and then translated again into a machine code, a Lua interpreter directly translates the source into bytecode — but even in this case some optimization is possible [15].

Like \LaTeX{}, Lua can dump a module into a kind of “format” (called the bytecompiled version of the module) and this “format” can be exchanged between different machines with the same architecture and the same interpreter (i.e. the same major and minor version number). The reason for this is that the Lua interpreter has a kind of “software CPU” called Virtual Machine (VM), which is implemented in ANSI C and it is the same for all the same release of the interpreter. The name “bytecode” is not casual: each opcode of the instructions of the VM fits into one byte (while the size of an instruction is 32 bit) and a VM is nothing else than a bytecode interpreter (after all we can also see a physical CPU as a machine-code interpreter).

A bytecode interpreter is usually the best choice if we want to implement a general programming language and we also want a fast and portable interpreter (see [16], chapter 10). The reason is that the design of a general programming language is more
complex than a simple DSL, but the theory of compilers is a powerful tool that can help enormously — we have only to avoid producing machine code. In fact any specific CPU is its own machine, and to try to adapt the compiler to each CPU is in conflict with the portability across different architectures. A better solution is to design a byte-compiler for a VM — this task is common for all platforms, gaining in portability — and implement the VM with a high level and widely available language like C. A VM is usually simpler than a physical CPU, so the byte-compiler can be optimized for performance — the translation must be fast; this means that the code can be complex and hence its design and implementation can require more effort compared to a DSL. This is why the bytecode is also important: we can use a cache to avoid re-parsing the source language. Of course a VM must be also fast, otherwise the interpreter of the general language is slow.

There are two ways to implement a VM: simulating a stack (stack-based VM) and simulating a register machine (register-based VM). A register-based VM is similar to a real piece of hardware, because it uses simulated general-purpose registers, but has no practical limits on their number as a real CPU does. A stack-based VM doesn’t have to figure out which register to use for which value, because instructions have implicit operands. Stack-based VMs are thus easy to implement, but register-based implementations better optimize the use of the registers of the physical CPU, which is the fastest memory available (300 times faster than DRAM), but is also very limited in size (typically not more than 1000 bytes, vs. a typical 4 GBytes of DRAM). Lua is the first widely used language to have a register-based VM ([15], section Introduction).

Let’s see for example how \texttt{a=1;b=2;c=a+b} is translated in bytecode by \texttt{luac}, the Lua bytecode compiler (text after \texttt{;} is a comment):

\begin{verbatim}
SETTABUP 0 -1 -2 ; _ENV "a" 1
SETTABUP 0 -3 -4 ; _ENV "b" 2
GETTABUP 0 0 -1 ; _ENV "a"
GETTABUP 1 0 -3 ; _ENV "b"
ADD 0 0 1
SETTABUP 0 -5 0 ; _ENV "c"
RETURN 0 1
\end{verbatim}

Expanding the meaning of the opcodes we have:

\begin{verbatim}
SETTABUP 0 -1 -2 ; UpValue[0][RK(-1)] := RK(-2)
SETTABUP 0 -3 -4 ; UpValue[0][RK(-3)] := RK(-4)
GETTABUP 0 0 -1 ; R(0) := UpValue[0][RK(-1)]
GETTABUP 1 0 -3 ; R(1) := UpValue[0][RK(-3)]
ADD 0 0 1 ; R(0) := RK(0) + RK(1)
SETTABUP 0 -5 0 ; UpValue[0][RK(-5)] := RK(0)
RETURN 0 1 ; return R(0),R(-1)
\end{verbatim}

UpValue[0] is the current environment, while R(.) is a register and RK(.) is a register or a constant: the access can be relative. So,

\begin{verbatim}
UpValue[0][RK(-1)] := RK(-2)
\end{verbatim}

means “in the current environment, set RK[-1] (i.e. "a") to RK[-2] (i.e. 1).” Each of these instructions is executed by the bytecode interpreter, which is, perhaps a bit surprisingly, a big switch-case loop (here we show a fragment):

\begin{verbatim}
while (1) {
  ...
  switch (op) {
    case OPR_AND: {
      luaK_goiftrue(fs, v);
      break;
    }
    case OPR_OR: {
      luaK_goiffalse(fs, v);
      break;
    }
    case OPR_CONCAT: {
      luaK_exp2nextreg(fs, v);
      break;
    }
    case OPR_ADD:
    case OPR_SUB:
    case OPR_MUL:
    case OPR_DIV:
    case OPR_MOD:
    case OPR_POW: {
      if (!isnumeral(v)) luaK_exp2RK(fs, v);
      break;
    }
    ...
  }
  /* end switch */
  ...
  /* end while */
\end{verbatim}

Currently most bytecode interpreters use the threading model technique, where each instruction is the address of the case target code. This is similar to, but not the same as, what we have seen for the \TeX interpreter.

To explain exactly what this means, let’s first remember that C has a goto statement that transfers the program flow to a point marked by a label, i.e.

\begin{verbatim}
goto somelabel;
\end{verbatim}

\begin{verbatim}
  ...
somelabel:
  /* some code */
\end{verbatim}

The address marked by \texttt{somelabel} is fixed at compile time, but some compilers (notably GCC) allow us to store the address of \texttt{somelabel} into an array to be used with a \texttt{goto}, using the label as a value so that the \texttt{goto} is computed at runtime:

Luigi Scarso
static void *array[] = { &somelabel };
...

/* equivalent to goto somelabel */
goto *array[0];
...
somelabel:

/* code */

Such labels as values are valid only within a function: computed goto cannot be used to jump to code in a different function. (Computed goto for GCC is described in [3].) Hence in this case labels are not truly first-class values, i.e. values that can be dynamically created, destroyed or passed as an argument. In contrast, in Lua all types (nil, boolean, number, string, table, function, userdata, thread) are first-class values and version 5.2 of Lua adds the goto statement too.

In this way it could be possible to replace the switch-case statement storing the bytcode instruction of the program with an array instruction and counting the next instruction with a program counter pc, as in the following pseudo-code:

```c
static void* dispatch_table[] = {
    ...
    &OPR_AND,
    &OPR_OR,
    &OPR_CONCAT,
    &OPR_ADD,
    ...
};

#define DISPATCH() \
goto *dispatch_table[instruction[pc++]]

int pc = 0;
while (1) {
    ...
    OPR_AND:
    luaK_goiftrue(fs, v);
    DISPATCH();
    }
    OPR_OR: {
    luaK_goiffalse(fs, v);
    DISPATCH();
    }
    OPR_CONCAT: {
    luaK_exp2nextreg(fs, v);
    DISPATCH();
    }
    OPR_ADD: {
    if (!isnumeric(v)) luaK_exp2RK(fs, v);
    DISPATCH();
    }
    ...
}/* end while */
```

There is also another important benefit: computed goto helps the branch predictor of the CPU. To understand the problem, let’s consider a real CPU as an interpreter of machine code. At this level, it’s still possible to divide the execution of a single instruction into atomic stages: let’s call them fetch (read an instruction from memory) decode, execute and write-back (write the result into memory) and let’s suppose that all the stages take the same time (which is not true in modern CPUs). A clock is mandatory to synchronize the stages, and a simple method (called Single-Cycle) is fetch - decode - execute - write the first instruction (4 cycles), fetch - decode - execute - write the second (again 4 cycles) and so on. If we have 4 instructions then after 16 cycles the overall execution is done, if we suppose that none of the instructions use jumps to other instructions.

But, if each stage is independent the CPU can do a better job: after the fetch of the first instruction, it can start its decode stage and simultaneously the fetch stage of the second instruction, as explained in this picture from [4]:

The CPU can thus complete 4 instructions after 8 cycles, doubling the throughput, even if each instruction still takes 4 cycles. This method is called pipelining. Modern CPUs can have more than 4 stages: more stages means more cycles, but also simpler circuitry and hence the chance to use a faster clock; but especially more stages mean high throughput. In fact in this case the distance $T_{pipeline}$ (in CPU cycles) between $I_1$ and $I_2$ is 1 cycle, while with the Single-Cycle we have $T_{Single-Cycle} = 4$ cycles and in general for an N-stage pipeline we have at best $T_{pipeline} = T_{Single-Cycle}/N$.

Back to the picture: a problem arises if execution of $I_1$ has as a consequence a jump to $I_4$, skipping $I_2$ and $I_3$ — and this is known only at the execution

LuajITTeX
stage. Given that the CPU knows that \( I_2 \) is a type of jump, a simple solution is to avoid using any stage of the pipeline until \( I_2 \) has ended its execute stage; in this case \( I_2 \) will end at the 5th cycle and \( I_4 \) will end at the 9th cycle—and we have the same performance of a Single-Cycle, because in this case with a Single-Cycle the CPU fetches \( I_4 \) at the 5th cycle. This is called a (control or branch) hazard.

To reduce the performance impact of this, modern CPUs have specialized hardware that predicts, with a conditional jump, the next instruction to fetch. This branch prediction can be a fixed rule (“never take the second choice”) or a dynamic branch prediction, which is usually based on a branch history table: a small amount of memory indexed by the lower portion of the address of the branch instruction, which contains some bits that say whether that branch was recently taken or not. (For the sake of simplicity, we are not distinguishing between branch predictors (has to decide if a branch condition will fail or not) and branch path predictors (which address to jump to), because often both are on the same circuitry.)

If the branch predictor makes the right choice, the throughput increase is saved; otherwise it has to clean the pipeline and fetch the correct instruction—and this is bad for performance. Nowadays, with an adequate algorithm, it’s possible to have from 99% to 82% of correct predictions. For a comprehensive treatment of these subjects see [14] and [17].

The key point in the above is that the prediction is based on the current branch instruction. We have seen that, in the best situation, a switch-case is implemented with the switch condition used as offset in a look-up table: in the \( \TeX \) pseudo-code above the crucial line is ‘\texttt{goto jump\_table[index];}’. The branch predictor sees this line as a branch instruction and, starting from index, it has to choose between all the following cases (the branches): it has one base address and \( n \) equally-spread branches to choose from, and modern CPUs cannot manage large \( n \) efficiently. The line ‘\texttt{if (index > 2) goto end;}’ which is mandatory for the switch also adds overhead.

With a computed goto the key line is
\[
\texttt{goto \ast\texttt{dispatch\_table[instruction[pc+]]}}
\]
Each individual case becomes the address used by the branch predictor; the branch predictor has \( n \) different base addresses and statistically the next instruction is not equally spread between all \( n \) choices and hence the address to jump is better predicted.

The difference between these two can be significant: following [5], a branch predictor mispredicts 81%–98% with switch and 57%–63% with the threaded model.

So, why doesn’t the Lua implementation use the computed goto? The reason is that this extension is not ANSI C, the language chosen to implement Lua. This is clearly explained in [15]: the threading model would compromise the portability of Lua. For example, the Microsoft C compiler doesn’t support labels as values, and the Intel ICC compiler supports them under Linux but not under Microsoft Windows.

In the end, this is a good choice at least for \( \TeX \), given that also \( \TeX \) aims to be portable: and if it were a true bottleneck, it should be possible to re-factor the C source of Lua with macros and conditional compilation to choose at compile time the type of the interpreter. And, finally, maybe there will be ANSI C compatible interpreters with lower error rates based on completely different models (after all, a failure rate of 57% is surely high enough to justify further research).

So far we have seen that \( \TeX \) and Lua use different kinds of interpreter (direct-syntax vs. bytecode), both optimal for their purposes; both have bytecode output, with almost the same issues on portability, though not so relevant for the \( \TeX \) users (and \( \TeX \) is slightly better), both use the best choice of main loop of the interpreter, compatible with the portability goal and manageability of the code.

Let’s look now at LuaJIT.

**2.3 LuaJIT**

LuaJIT, by Mike Pall [6], drops the requirement of portability on as many platforms as possible, and changes important parts of the Lua interpreter, keeping compatibility with Lua 5.1 plus other constructs like goto. First, LuaJIT still has a bytecode interpreter, but it is written in assembly language. It’s clear that this immediately leads to the conclusion that there can be (and in fact are) some platforms that are not supported, but we postpone this topic for later.

The way that LuaJIT builds the VM is a bit complex: first, a buildvm program is built for the given platform, then buildvm uses a dasc file (a mix of C and assembly) that describes the physical CPU and emits a lj_vm.s assembly file (the VM) that is finally compiled. For an x86_64 CPU, the vm_x86.dasc file looks like this:

```c
/* Generate the code for a single instruction. */
static void build_insn(BuildCtx *ctx,
                      BCOp op, int defop)
{
    int vk = 0;
```
It seems that the main loop is still a `switch` statement but LuaJIT under the hood uses a threading model — the same computed goto we saw above. This is possible because assembly language does not have the limitations of the C language, but of course the price to pay is maintaining several different assembly language sources of the same program.

It’s important to stress a couple of things: the optimal use of registers (LuaJIT keeps all important variables of the state in registers, and this kind of optimization is hard to achieve with a C compiler, at least for an x86 CPU) and the size in bytes. An assembly program once compiled is usually smaller than the C counterpart, which means it has a better chance of fitting into cache memory (which is at least two times faster than DRAM). For example, in a compiled version of LuaJIT for an x86_64 CPU the VM is 28560 bytes and it’s common to find laptop computers with an L1 cache of 128 KiB with an access time 100 times faster than DRAM. Note that the bytecode is still portable between different LuaJIT VMs (sharing the same version), but it’s not compatible with the Lua VM. The interpreter alone is claimed to be from 2 to 4 times faster than the Lua interpreter [7].

The second important feature is that the VM supports translation of the bytecode into machine code at run time. This is called Just In Time compilation (hence the name LuaJIT), and it uses a `trace` compiler: a compiler that keeps track of frequently used “flat” sequences of bytecodes and only translates the “hot” ones the first time, reusing the machine-code subsequently. The VM and the trace compiler cooperate very closely, but we can describe the operations in four phases (see [7], section ‘How a trace compiler works’):

1) interpretation: the VM interprets the bytecodes and collects statistics, so that if some code path (i.e. a sequence of Lua statements) reaches a given threshold its trace (the relative linear sequence of bytecodes) is considered “hot” and the VM goes on to the next phase;

2) interpretation and recording: while continuing the interpretation of bytecode, the VM records the associate actions and translates them into an intermediate representation called static-single assignment (SSA), in which each variable is assigned exactly once;

3) trace compilation: if the recording is ok (for example all bytecode of the trace can be translated into a SSA), the SSA is optimized and translated into machine code;

4) trace execution: the compiled code is executed and reused if possible.

It’s important to note that even during the trace compilation phase some runtime conditions (e.g. a bound check that fails) can halt execution of the compiled code and return to the standard way of bytecode interpretation, with a loss of performance. Of course we cannot forget the fact that we can have the benefits of a compiled language (high speed of execution) with the benefits of an interpreted one (high speed of development) — the key point of the JIT method.

The last important fact is the support of the Foreign Function Interface (FFI) via the Lua module `ffi`. Briefly, this module allows two things:
1) pure Lua code can call external C functions (i.e., functions in external libraries such as *.dll and *.so); there is a special namespace ffi.C that permits using, at least on POSIX systems and Microsoft Windows, the symbols from the current system C library.

2) it’s possible to use C data structures from pure Lua code: they are compiled to machine code at runtime by the JIT compiler. An example shown at http://lua.org/ext_ffi.html is eloquent: replacing a Lua table with a C struct on x86_64 has a speedup of 110x (i.e., 110 times faster than Lua) and the memory consumption is 64 times less. Apart from the C preprocessor, LuaJIT with the ffi module is hence similar (but less powerful) to a C interpreter like cling [8] (an interpreter for C++).

3 Building LuaJITTEX and first results

3.1 Building LuaJITTEX

Building LuaJITTEX was a bit of a complicated task, because LuaJIT has its own system to detect the host CPU and build the VM in assembly language, and this system doesn’t fit well with the way LuaTEX builds its binaries. After a few tries, we eventually decide to modify the layout of the source code of the original LuaTEX, moving the LuaJIT source to the same level of other support libraries like png, cairo, zlib, xpdf and using the original build system of LuaJIT. This seemed reasonable, given that LuaTEX is moving in the same direction (i.e., also move the stock Lua 5.2 to the level of the support libraries), so integration in the future can be easier than now. Some C files (less than ten) also needed to be adapted, but overall, after the change of the layout the integration was quite easy.

We knew that an important point was building LuaJITTEX for several platform with different compilers and checking the performance with a significant Lua code base. Initially the first version was only for Linux 32-bit, then the support for 64-bit was added; after that, we checked the mingw 32-bit version, using the same compiler of LuaTEX, but cross-compiling under Linux. After the mingw version we started to work on the source of luatex.exe from http://www.w32tex.org/ by Akira Kakuto, which is known to compile with the Microsoft compiler for x86. We were able to adapt that source code to LuaJIT as well and compile it with the MS compiler VC 2008 Express edition, again under Linux with Wine.

Compilations in hand, we started a period of testing using the Lua code base of ConTeXt Mark IV. That ended around mid-December 2012, and eventually the LuaJITTEX project was created at http://foundry.supelec.fr/gf/project/luajittex; the first release was on Christmas 2012. On 31 December the first version of luajittex.exe made by Prof. Kakuto was on the w32tex server. Later, in January 2013 we fixed the binary for Mac OS X 64-bit and added support for the compilation with the clang compiler. Of course as always testing is welcome: as stated in [9], the choice of compiler can influence the performance, and we need more feedback on this.

3.2 First impressions

Extensive tests were done on the Lua code base of ConTeXt Mark IV, and [13] (in this issue of TUGboat) reports numerical results. The first tests show that there was an improvement of speed of about 25%, and, if we decompose into the TEx time and the Lua time, we have measured effectively a 2x speedup of the Lua interpreter. Turning the JIT compilation itself on and off didn’t change the results significantly; in fact, with JIT on, LuaJITTEX is a bit slower than with JIT off. Very likely the reason is that few functions of the Lua standard libraries are JIT-compiled (see [10]) and when the JIT compiler sees a Not Yet Implemented (NYI) instruction, it has to jump from the trace compilation phase to the interpretation phase, and this has a cost. And of course, when nothing can be JIT-compiled the analysis is useless overhead.

Given that Mark IV uses the standard libraries and does lots of node-list manipulations it’s not a surprise that there is a performance penalty: there is not much to JIT. Thus, the speedup essentially comes from the new VM written in assembly language.

The full power of JIT can be seen with pure Lua or with the math functions; we have also made some quick tests on using the ffi module and registered 10x speedups. Of course the price to pay is the loss of garbage collection: using ffi we must pay attention to the memory management and how the garbage collector works. We have not checked the calling of external libraries.

The overall impression is that LuaJITTEX is faster than LuaTEX, but not so overwhelmingly fast: in both, the TEX interpreter is still the dominant part. The memory footprint was slightly less in LuaJITTEX.

4 Conclusion

LuaJITTEX looks like the best way to write a high-performance Lua interpreter — it’s hard to believe that another implementation could do better without multi-threading. We have consistently measured that the VM is 2 times faster than standard Lua (as in [7])
on an x86 CPU, which shows that LuaJIT is well-adapted to the LuaTeX code base. We also measured a 25% improvement on time, which is probably the best we can achieve modifying only the Lua side.

We think that the JIT compiler and FFI can achieve their full potential only if one starts by writing LuaJIT code from the very beginning; currently LuaJIT is not well-suited to a large standard Lua code base that uses the standard libraries.

Overall, we don’t see LuaJITTeX as a potential replacement of LuaTeX, but rather as an engine that can have higher performance in particular situations, for example, an automatic workflow of simple typesetting tasks, especially in the hands of a developer with a good knowledge of the C language and memory management. In this situation, writing a format that is a mix of TeX, Lua and C, together with the ability of LuaJIT to make simple the task of the binding, can make LuaJITTeX a very effective tool.

On the other hand, we cannot hide the potential compatibility issue as LuaTeX moves on to Lua 5.2 and the resulting differences with LuaJIT 2.0 (which currently uses Lua 5.1 plus some constructs from Lua 5.2). We will try to keep LuaJITTeX and LuaTeX in sync as much as possible, but the preference is for LuaTeX, which is the main reference. Users with no particularly demanding tasks are strongly encouraged to use LuaTeX.

Finally, it was very instructive to learn how to set up a toolchain for different compilers, especially for the compilation of luatex.exe. We see this as preparation for the SwigLib project, where one of the challenges will be checking the binaries of the libraries for different platforms.

References


(Links checked on 21 January 2013.)

Luigi Scarso
luigi dot scarso (at) gmail dot com

LuaJITTeX
ConT\TeX: Just-in-time Lua\TeX

Hans Hagen

1 Introduction

Reading occasional announcements about Lua\textsc{jit},\footnote{Lua\textsc{jit} is written by Mike Pall and more information about it and the technology it uses is at \url{http://luajit.org}, a site also worth visiting for its clean design.} one starts wondering if just-in-time (“jit”) compilation can speed up Lua\TeX. As a side track of the SwigLib project and after some discussion, Luigi Scarso decided to compile a version of Lua\TeX that had the jit compiler as the Lua engine. That’s when our journey into jit began.

We started with Linux 32-bit as this is what Luigi used at that time. Some quick first tests indicated that the Lua\textsc{jit} compiler made ConT\TeX\textsc{m}k\textsc{iv} run faster but not that much. Because Lua\textsc{jit} claims to be much faster than stock Lua, Luigi then played a bit with ffi, i.e. mixing C code and Lua, especially data structures. There is indeed quite some speed to gain here; unfortunately, we would have to mess up the ConT\TeX\ code base so much that one might wonder why Lua was used in the first place. I could confirm these observations in a Xubuntu virtual machine in VMware running under 32-bit Windows 8. So, we decided to conduct some more experiments.

A next step was to create a 64-bit binary because the servers at Pragma are KVM virtual machines running a 64-bit OpenSuse 12.1 and 12.2. It took a bit of effort to get a jit version compiled because Luigi didn’t want to mess up the regular codebase too much. This time we observed a speedup of about 40\% on some runs so we decided to move on to Windows to see if we could observe a similar effect there. And indeed, when we adapted Akira Kakuto’s Windows setup a bit we could compile a version for Windows using the native Microsoft compiler. On my laptop a similar speedup was observed, although by then we saw that in practice a 25\% speedup was about what we could expect. A bonus is that making formats and identifying fonts is also faster.

So, in that stage, we could safely conclude that Lua\TeX combined with Lua\textsc{jit} made sense if you want a somewhat faster version. But where does the speedup come from? The easiest way to see if jitting has effect is to turn it on and off.

\begin{verbatim}
jit.on()
\end{verbatim}

2 We also tweaked some of the fine-tuning parameters of Lua\textsc{jit} but didn’t notice any differences. In due time more tests will be done.

3 If we want to improve these mechanisms it makes much more sense to make more helpers. However, profiling has shown us that the most demanding code is already quite optimized.

\begin{verbatim}
jit.off()
\end{verbatim}

To our surprise ConT\TeX runs are not much

\begin{verbatim}
\setupbodyfont[dejavu] % benchmark-1.tex
\starttext
\dontcomplain
\startluacode
if jit then
    jit.on()
    jit.off()
end
\endluacode
\end{verbatim}

influenced by turning the jitter on or off.\footnote{We also tweaked some of the fine-tuning parameters of Lua\textsc{jit} but didn’t notice any differences. In due time more tests will be done.} This means that the improvement comes from other places:

- The virtual machine is a different one, and targets the platforms that it runs on. This means that regular bytecode also runs faster.
- The garbage collector is the one from Lua 5.2, so that can make a difference. It looks like memory consumption is somewhat lower.
- Some standard library functions are recognized and supported in a more efficient way. Think of \texttt{math.sin}.
- Some built-in functions like \texttt{type} are probably dealt with in a more efficient way.

The third item is an important one. We don’t use many standard functions. For instance, if we need to go from characters to bytes and vice versa, we have to do that for UTF so we use some dedicated functions or LPEG. If in ConT\TeX we parse strings, we often use LPEG instead of string functions anyway. And if we still do use string functions, for instance when dealing with simple strings, it only happens a few times.

The more demanding ConT\TeX code deals with node lists, which means frequent calls to core Lua\TeX\ functions. Alas, jitting doesn’t help much there unless we start messing with ffi which is not on the agenda.\footnote{If we want to improve these mechanisms it makes much more sense to make more helpers. However, profiling has shown us that the most demanding code is already quite optimized.}

2 Benchmarks

Let’s look at some of the benchmarks. The first one uses MetaPost and because we want to see if calculations are faster, we draw a path with a special pen so that some transformations have to be done in the code that generates the PDF output. We only show the Windows and 64-bit Linux tests here. The 32-bit tests are consistent with those on Windows so we didn’t add those timings here (also because in the meantime Luigi’s machine broke down and he moved on to 64 bits).

\begin{verbatim}
\setupbodyfont[dejavu] \% benchmark-1.tex
\starttext
\dontcomplain
\startluacode
\endluacode
\end{verbatim}
The following times are measured in seconds. They are averages of 5 runs. There is a significant speedup but jitting doesn’t do much.

<table>
<thead>
<tr>
<th></th>
<th>traditional</th>
<th>jit on</th>
<th>jit off</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 8</td>
<td>26.0</td>
<td>20.6</td>
<td>20.8</td>
</tr>
<tr>
<td>Linux 64</td>
<td>34.2</td>
<td>14.9</td>
<td>14.1</td>
</tr>
</tbody>
</table>

Our second example uses multiple fonts in a paragraph and adds color as well. Although well optimized, font-related code involves node list parsing and a bit of calculation. Color again deals with node lists and the backend code involves calculations but not that many. The traditional run on Linux is somewhat odd, but might have to do with the fact that the MetaPost library suffers from the 64 bits. It is at least an indication that optimizations make less sense if there is a different dominant weak spot. We have to look into this some time.

Again jitting has no real benefits here, but the overall gain in speed is quite nice. It could be that the garbage collector plays a role here.

<table>
<thead>
<tr>
<th></th>
<th>traditional</th>
<th>jit on</th>
<th>jit off</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 8</td>
<td>54.6</td>
<td>36.0</td>
<td>35.9</td>
</tr>
<tr>
<td>Linux 64</td>
<td>46.5</td>
<td>32.0</td>
<td>31.7</td>
</tr>
</tbody>
</table>

This benchmark writes quite a lot of data to the console, which can have impact on performance as \TeX flushes on a per-character basis. When one runs \TeX as a service this has less impact because in that case the output goes into the void. There is a lot of file reading going on here, but normally the operating system will cache data, so after a first run this effect disappears.\footnote{On Windows it makes sense to use \texttt{console2} because due to some clever buffering tricks it has a much better performance than the default console.}

The third benchmark is one that we often use for testing regression in speed of the Con\TeXt core code. It measures the overhead in the page builder without special tricks being used, like backgrounds. The document has some 1000 pages.
The fourth benchmark uses some structuring, which involved Lua tables and housekeeping, an itemize, which involves numbering and conversions, and a table mechanism that uses more Lua than TeX.

Here it looks like jit slows down the process, but of course we shouldn’t take the last digit too seriously.

Here we see jit having an effect! First of all the Lua JIT versions are now 4 times faster. Making the \texttt{sin} a local function does not make much of a difference because the math functions are optimized anyway. See how we’re still faster when jit is disabled:

<table>
<thead>
<tr>
<th>Platform</th>
<th>traditional</th>
<th>jit on</th>
<th>jit off</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 8</td>
<td>4.5</td>
<td>3.6</td>
<td>3.6</td>
</tr>
<tr>
<td>Linux 64</td>
<td>4.8</td>
<td>3.9</td>
<td>4.0</td>
</tr>
</tbody>
</table>

Again, this example does a bit of logging, but not that much reading from file as buffers are kept in memory.

We should start wondering when jit does kick in. This is what the fifth benchmark does.

Here it looks like jit slows down the process, but of course we shouldn’t take the last digit too seriously.
and compared to those speed-ups the 25\% is not that much. Also, as TeX deals with files, the advance of SSD disks and larger and faster memory helps too. Faster and larger CPU caches contributes too. On the other hand, multiple cores don’t help that much on a system that only runs TeX. Interesting is that multi-core architectures tend to run at slower speeds than single cores where more heat can be dissipated and in that respect servers mostly running TeX are better off with fewer cores that can run at higher frequencies. But anyhow, 25\% is still better than nothing and it makes my old laptop feel faster. It prolongs the lifetime of machines!

Now, say that we cannot speed up TeX itself that much, but that there is still something to gain at the Lua end — what can we reasonably expect? First of all we need to take into account that only part of the runtime is due to Lua. Say that this is 25\% for a document of average complexity.

\[
\text{runtime}_{\text{TeX}} + \text{runtime}_{\text{Lua}} = 100
\]

We can consider the time needed by TeX to be constant; so if that is 75\% of the total time (say 100 seconds) to begin with, we have:

\[
75 + \text{runtime}_{\text{Lua}} = 100
\]

It will be clear that if we bring down the runtime to 80\% (80 seconds) of the original we end up with:

\[
75 + \text{runtime}_{\text{Lua}} = 80
\]

And the 25 seconds spent in Lua went down to 5, meaning that Lua processing got 5 times faster! It is also clear that getting much more out of Lua becomes hard. Of course we can squeeze more out of it, but TeX still needs its time. It is hard to measure how much time is actually spent in Lua. We do keep track of some times but it is not that accurate. These experiments and the gain in speed indicate that we probably spend more time in Lua than we first guessed. If you look in the ConTeXt source it’s not that hard to imagine that indeed we might well spend 50\% or more of our time in Lua and/or in transferring control between TeX and Lua. So, in the end there still might be something to gain.

Let’s take benchmark 4 as an example. At some point we measured for a regular LuaTeX 0.74 run 27.0 seconds and for a LuaJITTeX run 23.3 seconds. If we assume that the LuaJIT virtual machine is twice as fast as the normal one, some juggling with numbers makes us conclude that TeX takes some 19.6 seconds of this. An interesting border case is \texttt{directlua}: we sometimes pass quite a lot of data and that gets tokenized first (a TeX activity) and the resulting token list is converted into a string (also a TeX activity) and then converted to bytecode (a Lua task) and when okay executed by Lua. The time involved in conversion to byte code is probably the same for stock Lua and LuaJIT.

In the LuaTeX case, 30\% of the runtime for benchmark 4 is on Lua’s tab, and in LuaJITTeX it’s 15\%. We can try to bring down the Lua part even more, but it makes more sense to gain something at the TeX end. There macro expansion can be improved (read: ConTeXt core code) but that is already rather optimized.

Just for the sake of completeness Luigi compiled a stock LuaTeX binary for 64-bit Linux with the \texttt{-o3} option (which forces more inlining of functions as well as a different switch mechanism). We did a few tests and this is the result:

\begin{itemize}
  \item \texttt{benchexam-1} 15.3 15.0
  \item \texttt{benchexam-2} 35.8 34.0
  \item \texttt{benchexam-3} 4.0 3.9
  \item \texttt{benchexam-4} 16.0 15.8
\end{itemize}

This time we used \texttt{--batch} and \texttt{--silent} to eliminate terminal output. So, if you really want to squeeze out the maximum performance you need to compile with \texttt{-o3}, use LuaJITTeX (with the faster virtual machine) but disable jit (disabled by default anyway).

We have no reason to abandon stock Lua. Also, because during these experiments we were still using Lua 5.1 we started wondering what the move to 5.2 would bring. Such a move forward also means that ConTeXt MkIV will not depend on specific LuaJIT features, although it is aware of it (this is needed because we store bytecodes). But we will definitely explore the possibilities and see where we can benefit. In that respect there will be a way to enable and disable jitting. So, users have the choice to use either stock LuaTeX or the jit-aware version but we default to the regular binary.

As we use stock Lua as benchmark, we will use the \texttt{bit32} library, while LuaJIT has its own bit library. Some functions can be aliased so that is no big deal. In ConTeXt we use wrappers anyway. More problematic is that we want to move on to Lua 5.2 and not all 5.2 features are supported (yet) in LuaJIT. So, if LuaJIT is mandatory in a workflow, then users had better make sure that the Lua code is compatible. We don’t expect too many problems in ConTeXt MkIV.

4 About speed

It is worth mentioning that the Lua version in LuaTeX has a patch for converting floats into strings. Instead of some \texttt{INF#} result we just return zero, simply because TeX is integer-based and intercepting
incredibly small numbers is too cumbersome. We had to apply the same patch in the jit version.

The benchmarks only indicate a trend. In a real document much more happens than in the above tests. So what are measurements worth? Say that we compile The \TeXbook\ This grandparent of all documents coded in \TeX is rather plainly coded (using of course plain \TeX) and compiles pretty fast. Processing does not suffer from complex expansions, there is no color, hardly any text manipulation, it’s all 8 bit, the pagebuilder is straightforward as is all spacing. Although on my old machine I can get Con\TeX\xt to run at over 200 pages per second, this quickly drops to 10\% of that speed when we add some color, backgrounds, headers and footers, font switches, etc.

So, running documents like The \TeXbook\ for comparing the speed of, say, pd\TeX, X\TeX, Lua-T\EX\ and now LuaJIT\TeX\ makes no sense. The first one is still eight bit, the rest are Unicode. Also, The \TeXbook\ uses traditional fonts with traditional features so effectively that it doesn’t rely on anything that the new engines provide, not even \e-\TeX\ extensions. On the other hand, a recent document uses advanced fonts, properties like color and/or transparencies, hyperlinks, backgrounds, complex cover pages or chapter openings, embeds graphics, etc. Such a document might not even process in pd\TeX\ or X\TeX, and if it does, it’s still comparing different technologies: eight bit input and fast fonts in pd\TeX, frozen Unicode and wide font support in X\TeX, instead of additional trickery and control, written in Lua. So, when we investigate speed, we need to take into account what (font and input) technologies are used as well as what complicating layout and rendering features play a role. In practice speed only matters in an edit-view cycle and services where users wait for some result.

It’s rather hard to find a recent document that can be used to compare these engines. The best we could come up with was the rendering of the user interface documentation. The last column is the time in seconds, the others are the command line invocation.

\begin{verbatim}
texexec --engine=pdftex --global x-set-12.mkii 5.9
texexec --engine=xetex --global x-set-12.mkii 6.2
texexec --engine=luatex --global x-set-12.mkiv 6.2
texexec --engine=lua4jitter --global x-set-12.mkiv 4.6
\end{verbatim}

Keep in mind that texexec is a Ruby script and uses kpsewhich while context uses Lua and its own (TDS-compatible) file manager. But still, it is interesting to see that there is not that much difference if we keep jit out of the picture. This is because in MkIV we have somewhat more clever XML processing, although earlier measurements have demonstrated that in this case not that much speedup can be assigned to that.

And so recent versions of MkIV already keep up rather well with the older eight bit world. We do way more in MkIV and the interfacing macros are nicer but potentially somewhat slower. Some mechanisms might be more efficient because of using Lua, but some actually have more overhead because we keep track of more data. Font feature processing is done in Lua, but somehow can keep up with the libraries used in X\TeX, or at least is not that significant a difference, although I can think of more demanding tasks. Of course in Lua\TeX\ we can go beyond what libraries provide.

No matter what one takes into account, performance is not that much worse in Lua\TeX, and if we enable jit and so remove some of the traditional Lua virtual machine overhead, we’re even better off. Of course we need to add a disclaimer here: don’t force us to prove that the relative speed ratios are the same for all cases. In fact, it being so hard to measure and compare, performance can be considered to be something taken for granted as there is not that much we can do about getting nicer numbers, apart from maybe parallelizing which brings other complexities into the picture. On our servers, a few other virtual machines running \TeX\ services kicking in at the same time, using CPU cycles, network bandwidth (as all data lives somewhere else) and asking for disk access have much more impact than the 25\% we gain. Of course if all processes run faster then we’ve gained something.

For what it’s worth: processing this text takes some 2.3 seconds on my laptop for regular Lua\TeX\ and 1.8 seconds with LuaJIT\TeX\, including the extra overhead of restarting. As this is a rather average example it fits earlier measurements.

Processing a font manual (work in progress) takes LuaJIT\TeX\ 15 seconds for 112 pages compared to 18.4 seconds for Lua\TeX. The not yet finished manual loads 20 different fonts (each with multiple instances), uses colors, has some MetaPost graphics and does some font juggling. The gain in speed sounds familiar.

5 The future

At the 2012 Lua conference Roberto Ierusalimschy mentioned that the virtual machine of LuaJIT is about twice as fast due to it being partly done in assembler while the regular machinery is written in standard C code and keeps portability in mind.

He also presented some plans for future versions of Lua. There will be some lightweight helpers for
UTF. Our experiences so far are that only a handful of functions are actually needed: byte to character conversions and vice versa, iterators for UTF characters and UTF values and maybe a simple substring function is probably enough. Currently LuaTEX has some extra string iterators and it will provide the converters as well.

There is a good chance that LPEG will become a standard library (which it already is in LuaTEX), which is also nice. It’s interesting that, especially on longer sequences, LPEG can beat the string matchers and replacers, although when in a substitution no match and therefore no replacements happen, the regular gsub wins. We’re talking small numbers here, in daily usage LPEG is about as efficient as you can wish. In ConTeXt we have a \texttt{lppeg.UR} and \texttt{lppeg.US} and it would be nice to have these as native UTF related methods, but I must admit that I seldom need them.

This and other extensions coming to the language also have some impact on a jit version: the current LuaJIT is already not entirely compatible with Lua 5.2 so you need to keep that into account if you want to use this version of LuaTEX. So, unless LuaJIT follows the mainstream development, as ConTeXt MkIV user you should not depend on it. But at the moment it’s nice to have this choice.

The yet experimental code will end up in the main LuaTEX repository in time before the \TeX Live 2013 code freeze. In order to make it easier to run both versions alongside, we have added the Lua 5.2 built-in library \texttt{bit32} to LuaJITTEX. We found out that it’s too much trouble to add that library to Lua 5.1 but LuaTEX has moved on to 5.2 anyway.

6 Running

So, as we will definitely stick to stock Lua, one might wonder if it makes sense to officially support jitting in ConTeXt. First of all, LuaTEX is not influenced that much by the low level changes in the API between 5.1 and 5.2. Also LuaJIT does support the most important new 5.2 features, so at the moment we’re mostly okay. We expect that eventually LuaJIT will catch up but if not, we are not in big trouble: the performance of stock Lua is quite okay and above all, it’s portable! For the moment you can consider LuaJITTEX to be an experiment and research tool, but we will do our best to keep it production ready.

So how do we choose between the two engines? After some experimenting with alternative startup scenarios and dedicated caches, the following solution was reached:

\texttt{context --engine=luajittex ...}

The usual preamble line also works:

\texttt{\% engine=luajittex}

As the main infrastructure uses the \texttt{luatex} and related binaries, this will result in a relaunch: the \texttt{context} script will be restarted using \texttt{luajittex}. This is a simple solution and the overhead is rather minimal, especially compared to the somewhat faster run. Alternatively you can copy \texttt{luajittex} over \texttt{luatex} but that is more drastic. Keep in mind that \texttt{luatex} is the benchmark for development of ConTeXt, so the jit aware version might fall behind sometimes.

Yet another approach is adapting the configuration file, or better, provide (or adapt) your own \texttt{texmfconf.lua} in for instance \texttt{texmf-local/web2c} path:

\begin{verbatim}
return {
  type = "configuration",
  version = "1.2.3",
  date = "2012-12-12",
  time = "12:12:12",
  comment = "Local overloads",
  author = "Hans Hagen, PRAGMA-ADE, Hasselt NL",
  content = {
    directives = {
      ["system.engine"] = "luajittex",
    },
  },
}
\end{verbatim}

This has the same effect as always providing \texttt{--engine=luajittex} but only makes sense in well controlled situations as you might easily forget that it’s the default. Of course one could have that file and just comment out the directive unless in test mode.

Because the bytecode of LuaJIT differs from the one used by Lua itself we have a dedicated format as well as dedicated bytecode compiled resources (for instance \texttt{tmb} instead of \texttt{tmc}). For most users this is not something they should bother about as it happens automatically.

Based on experiments, by default we have disabled jit so we only benefit from the faster virtual machine. Future versions of ConTeXt might provide some control over that but first we want to conduct more experiments.

7 Addendum

These developments and experiments took place in November and December 2012. At the time of this writing we also made the move to Lua 5.2 in stock

\texttt{ConTeXt}: Just-in-time LuaTEX
LuaTeX: the first version to provide this was 0.74. Here are some measurements on Taco Hoekwater’s 64-bit Linux machine:

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>LuaTeX 0.70</th>
<th>LuaTeX 0.74</th>
</tr>
</thead>
<tbody>
<tr>
<td>benchmark-1</td>
<td>23.67</td>
<td>19.57 faster</td>
</tr>
<tr>
<td>benchmark-2</td>
<td>65.41</td>
<td>62.88 faster</td>
</tr>
<tr>
<td>benchmark-3</td>
<td>4.88</td>
<td>4.67 faster</td>
</tr>
<tr>
<td>benchmark-4</td>
<td>23.09</td>
<td>22.71 faster</td>
</tr>
<tr>
<td>benchmark-5</td>
<td>2.56/2.06</td>
<td>2.66/2.29 slower</td>
</tr>
</tbody>
</table>

There is a good chance that this is due to improvements of the garbage collector, virtual machine and string handling. It also looks like memory consumption is a bit less. Some speed optimizations in reading files have been removed (at least for now) and some patches to the `format` function (in the `string` namespace) that dealt with (for `TeX`) unfortunate number conversions have not been ported. The code base is somewhat cleaner and we expect to be able to split up the binary in a core program plus some libraries that are loaded on demand.

In general, we don’t expect too many issues in the transition to Lua 5.2, and ConTeXt is already adapted to support LuaTeX with 5.2 as well as LuaJITTeX with an older version.

Running the same tests on a 32-bit Windows machine gives this:

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>LuaTeX 0.70</th>
<th>LuaTeX 0.74</th>
</tr>
</thead>
<tbody>
<tr>
<td>benchmark-1</td>
<td>26.4</td>
<td>25.5 faster</td>
</tr>
<tr>
<td>benchmark-2</td>
<td>64.2</td>
<td>63.6 faster</td>
</tr>
<tr>
<td>benchmark-3</td>
<td>7.1</td>
<td>6.9 faster</td>
</tr>
<tr>
<td>benchmark-4</td>
<td>28.3</td>
<td>27.0 faster</td>
</tr>
<tr>
<td>benchmark-5</td>
<td>1.95/1.50</td>
<td>1.84/1.48 faster</td>
</tr>
</tbody>
</table>

The gain is less impressive but the machine is rather old and we can benefit less from modern CPU properties (cache, memory bandwidth, etc.). I tend to conclude that there is no significant improvement here but it also doesn’t get worse. However we need to keep in mind that file I/O is less optimal in 0.74 so this might play a role. As usual, runtime is negatively influenced by the relatively slow speed of displaying messages on the console (even when we use `console2`).

A few days before the end of 2012, Akira Kakuto compiled native Windows binaries for both engines.

This time I decided to run a comparison inside the SciTE editor, that has very fast console output.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>LuaTeX 0.74</th>
<th>LuaJITTeX 0.72</th>
</tr>
</thead>
<tbody>
<tr>
<td>benchmark-1</td>
<td>25.4</td>
<td>25.4 similar</td>
</tr>
<tr>
<td>benchmark-2</td>
<td>54.7</td>
<td>36.3 faster</td>
</tr>
<tr>
<td>benchmark-3</td>
<td>4.3</td>
<td>3.6 faster</td>
</tr>
<tr>
<td>benchmark-4</td>
<td>20.0</td>
<td>16.3 faster</td>
</tr>
<tr>
<td>benchmark-5</td>
<td>1.93/1.48</td>
<td>0.74/0.61 faster</td>
</tr>
</tbody>
</table>

Only the MetaPost library and conversion benchmark didn’t show a speedup. The regular `TeX` tests 1–3 gain some 15–35%. Enabling jit (off by default) slowed down processing. For the sake of completeness I also timed LuaJITTeX on the console, so here you see the improvement of both engines.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>LuaTeX 0.70</th>
<th>LuaTeX 0.74</th>
<th>LuaJITTeX 0.72</th>
</tr>
</thead>
<tbody>
<tr>
<td>benchmark-1</td>
<td>26.4</td>
<td>25.5</td>
<td>25.9</td>
</tr>
<tr>
<td>benchmark-2</td>
<td>64.2</td>
<td>63.6</td>
<td>45.5</td>
</tr>
<tr>
<td>benchmark-3</td>
<td>7.1</td>
<td>6.9</td>
<td>6.0</td>
</tr>
<tr>
<td>benchmark-4</td>
<td>28.3</td>
<td>27.0</td>
<td>23.3</td>
</tr>
<tr>
<td>benchmark-5</td>
<td>1.95/1.50</td>
<td>1.84/1.48</td>
<td>0.73/0.60 faster</td>
</tr>
</tbody>
</table>

In this text, the term jit has come up a lot but you might rightfully wonder if the observations here relate to jit at all. For the moment I tend to conclude that the implementation of the virtual machine and garbage collection have more impact than the actual just-in-time compilation. More exploration of jit is needed to see if we can really benefit from that. Of course the fact that we use a bit less memory is also nice. In case you wonder why I bother about speed at all: we happen to run LuaTeX mostly as a (remote) service and generating a bunch of (related) documents takes a bit of time. Bringing the waiting down from 15 to 10 seconds might not sound impressive but it makes a difference when it is someone’s job to generate these sets.

In summary: just before we entered 2013, we saw two rather fundamental updates of LuaTeX show up: an improved traditional one with Lua 5.2 as well as the somewhat faster LuaJITTeX with a mixture between 5.1 and 5.2. And in 2013 we will of course try to make them both even more attractive.

⋄ Hans Hagen

http://pragma-ade.com

---

6 Of course this poses some constraints on stability as components get decoupled, but this is one of the issues that we hope to deal with properly in the library project.

7 Most of my personal `TeX` runs are from within SciTE, while most runs on the servers are in batch mode, so normally the overhead of the console is acceptable or even neglectable.

Hans Hagen
ConTeXt basics for users: Images
Aditya Mahajan

Abstract
As the cliché goes, a picture is worth a thousand words. This article provides an overview of inserting pictures or images in a ConTeXt document.

A note about MkII and MkIV
In contrast to the previous articles in this series, from now on, I will assume that ConTeXt Mk IV is being used: LuaTEX engine and PDF output. ConTeXt Mk IV behaves differently from MkII, and in most cases provides additional features that are absent from MkII.

1 Basic usage
The simplest way to insert an image is to use:
\externalfigure[logo.pdf]

This command places the PDF image logo.pdf in a \vbox; the width and height of the image are equal to the natural dimensions of the image.

To set the width of the image to a specific size, say 1cm, use:
\externalfigure[logo.pdf][width=1cm]

Similarly, to set the height of the image to a specific size, say 2cm, use:
\externalfigure[logo.pdf][height=2cm]

If only the width or height of the image is specified, the other dimension is scaled appropriately to keep the aspect ratio.

To include a specific page, say page 5, of a multi-page PDF file, use:
\externalfigure[logo.pdf][page=5]

These four variations cover 90% of the use cases.

1.1 Natively supported file formats
ConTeXt natively supports the image formats enumerated below. The image format is determined from the file extension (case insensitive).

- PDF: File extension .pdf
- MPS (MetaPost output): File extension .mps or ./digits
- JPEG: File extension .jpg or .jpeg
- PNG: File extension .png
- JPEG 2000: File extension .jp2
- JBIG or JBIG2: File extension .jbig, .jbig2, or .jb2

1.2 Including images after conversion
The image formats listed in Sec. 1.1 are the ones that may be embedded directly in a PDF. ConTeXt also supports a few other formats which are first converted to PDF using an external program. Of course, for such a conversion to work, the corresponding converter must be in the PATH.

<table>
<thead>
<tr>
<th>Format</th>
<th>Extension</th>
<th>Converter</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVG</td>
<td>.svg, .svgz</td>
<td>inkscape</td>
</tr>
<tr>
<td>EPS</td>
<td>.eps, .ai</td>
<td>gs (or gswin32c on Windows): Ghostscript</td>
</tr>
<tr>
<td>GIF</td>
<td>.gif</td>
<td>gm from GraphicsMagick</td>
</tr>
<tr>
<td>TIFF</td>
<td>.tiff</td>
<td>gm from GraphicsMagick</td>
</tr>
</tbody>
</table>

The conversion generates a PDF file with prefix m_k_i_v_ and a suffix .pdf added to the name of the original file. The result is cached, and the conversion is rerun if the timestamp of the original file is newer than that of the converted file.

1.3 Specifying image directories
By default, ConTeXt searches an image in the current directory, the parent directory, and the grand-parent directory.

To search for images in other directories, say a ./images subdirectory and /home/user/images, use:
\setupexternalfigures
[ directory={images, /home/user/images} ]

Note that one should always use forward slashes in path names, even on Windows.

The default search order is: the current directory, the parent directory, the grand-parent directory, and then the paths specified by the directory key. To restrict image search only to the paths specified by the directory key, use:
\setupexternalfigures
[ directory={global} ]

To restore the default search behavior, use:
\setupexternalfigures
[ directory={local, global} ]

The ConTeXt distribution includes three sample images: cow.pdf, mill.png, and hacker.jpg, that are useful when creating minimum working examples to illustrate a bug on the mailing list. These images are locating in the TEXMF directory. To add the TEXMF directory to the image search path, use:
\setupexternalfigures
[ directory={local, global, default} ]
The above alternative adds the entire \texttt{TEXMF} directory to the search path, \textit{including the \texttt{doc/} directory!} Therefore, one needs to be extremely careful when using this option. In fact, I would advise not using \texttt{location=default} except for illustrative minimal working examples.

1.4 Including remote images

Like all other Con\TeXt macros that read files, \texttt{\externalfigure} also supports reading remote files from HTTP(S) servers. An example:

\begin{verbatim}
\externalfigure [http://tug.org/images/logobw.jpg]
\end{verbatim}

When a document containing a remote file is compiled for the first time, the remote file is downloaded from the server and stored in the Lua\TeX cache directory. This cached file is used during subsequent runs.

Normally, the remote image is downloaded again if the image in the cache is older than 1 day. To change this threshold to, for example, 2 minutes (120 seconds), either add \texttt{\enabledirectives[schemes.threshold=120]} in the Con\TeXt file, or compile the Con\TeXt file using the command

\begin{verbatim}
context --directives=schemes.threshold=120 \filename
\end{verbatim}

The variable \texttt{schemes.threshold} is global, so changing its value affects all other macros like \texttt{\input}, \texttt{\usemodule}, \texttt{\component}, etc. that load remote files.

2 Image transformations

2.1 Scaling images

To scale an image use the \texttt{scale} key: \texttt{scale=1000} corresponds to the original dimensions of the image, \texttt{scale=500} scales the image to 50\% of the original size, \texttt{scale=1500} scales the images to 150\% of the original size, and so on. For example:

\begin{verbatim}
\externalfigure[logo.pdf][scale=500]
\end{verbatim}

If either \texttt{width} or \texttt{height} is specified, then the \texttt{scale} key has no effect.

In addition, the \texttt{xscale} and \texttt{yscale} keys scale the image in only one dimension. For example:

\begin{verbatim}
xscale=500 yscale=500
\end{verbatim}

2.2 Specifying maximum size of an image

Often, we want the included image to be no larger than a given size. E.g., this ensures that an included image is no more than \texttt{0.2\textwidth}:

\begin{verbatim}
\externalfigure[logo.pdf][maxwidth=0.2\textwidth]
\end{verbatim}

If \texttt{maxwidth} is specified and the width of the image is less than \texttt{maxwidth}, then the image is not scaled; if the width of the image is greater than \texttt{maxwidth}, then the width is restricted to \texttt{maxwidth} and the height is scaled appropriately to maintain the original aspect ratio.

Analogous to \texttt{maxwidth} is the option \texttt{maxheight}, which checks the height of the image.

In my own style files, I usually specify the following to ensure that figures do not overflow the text area:

\begin{verbatim}
\setupexternalfigures [maxwidth=\textwidth, maxheight=0.8\texttheight]
\end{verbatim}

2.3 Rotating images

To rotate the included image by 90, 180, or 270 degrees, use the \texttt{orientation} key. For example:

\begin{verbatim}
\externalfigure[logo.pdf][orientation=90]
\end{verbatim}

To rotate by an arbitrary angle, use the \texttt{\rotate} command. For example:

\begin{verbatim}
\rotate[rotation=45] \externalfigure[logo.pdf]
\end{verbatim}
2.4 Mirroring images

To mirror (flip) an image, use the generic \mirror command. For example, to mirror horizontally:
\mirror{\externalfigure[logo.pdf]}

To mirror vertically, first rotate the image by 180° and then mirror it:
\mirror{\externalfigure[logo.pdf][orientation=180]}

2.5 Clipping images

To clip an image, use the generic \clip command. For example, to clip the original image to a 1cm x 2cm rectangle at an offset of (3mm, 5mm) from the top left corner:
\clip[width=1cm, height=2cm, hoffset=3mm, voffset=5mm]{\externalfigure[logo.pdf]}

As another example, this cuts the image into a 3x3 pieces and then outputs the (2,2) piece:
\clip[nx=3, ny=3, x=2, y=2]{\externalfigure[logo.pdf]}

3 Troubleshooting

3.1 Visualizing the image bounding box

If, for instance, the image is taking more space than expected, it can be useful to visualize the bounding box of the image. To do this:
\externalfigure[logo.pdf][frame=on]

ConTeXt includes a Perl script pdftrimwhite that removes extra white space at the borders of a PDF file. To run this script:
mtxrun --script pdftrimwhite [flags] input output

The most important flag is --offset=⟨dimen⟩, which keeps some extra space around the trimmed image.

Similar functionality is provided by another Perl script, pdfcrop, that is included in most T\TeX distributions.

3.2 Tracking what is happening

To get diagnostic information about image inclusion, enable the tracker graphics.locating either by adding
\enabletrackers[graphics.locating]
in the ConTeXt file, or by compiling the ConTeXt file using the command
context --trackers=graphics.locating filename

The tracker writes diagnostics to the console. Suppose we use \externalfigure[somefile.pdf] and ConTeXt finds the file in the current search path; then the following information is printed on the console: (The tracking messages here are formatted to typeset nicely. The actual messages are slightly different.)

As another example, this cuts the image into a 3x3 pieces and then outputs the (2,2) piece:
\clip[nx=3, ny=3, x=2, y=2]{\externalfigure[logo.pdf]}

If the file somefile.pdf is not found in the current search path, then the following information is printed on the console (even if the graphics.locating tracker is not set):

If the file somefile.pdf is not found
Sometimes, one would rather use a placeholder image for an image that is yet to be made. In such cases, load the MP library dum via:
\useMPLibrary[dum]
Then, whenever an image file is not found in the current search path, a random MetaPost image is shown in the output.

3.3 Images at the beginning of a paragraph
Using \externalfigure[...] at the beginning of a paragraph results in a line break after the image. This is because \externalfigure is a \vbox and when \TeX encounters a \vbox at (what appears to be) the beginning of a paragraph, it remains in vertical mode. To prevent this, add \dontleavehmode before \externalfigure, like this:
\dontleavehmode
\externalfigure[...] ... first line ...

4 Settings for multiple images
4.1 Image settings
Suppose your document contains many side-by-side images, and you want all of these images to be of the same size. In addition, you want to control the size of all images by changing only one setup. To do this, you can use the \defineexternalfigure macro, which defines a named collection of image settings. For example, to define a collection where the image width is 3cm, use:
\defineexternalfigure[logo-settings]
[width=3cm]
And then to use these settings in an image, use:
\externalfigure[group.pdf][logo-settings]
or, if you want to add or override settings, use:
\externalfigure[group.pdf][logo-settings]
[height=2cm]

4.2 Labeled images
Suppose your document contains an image at multiple locations; all of these images are to be of the same size, which is not necessarily the same as the natural size of the image. Furthermore, as before, you want to set the size of all the images by changing only one setup. Here, the macro to use is \useexternalfigure, which defines a symbolic label for inserting an image plus settings. For example:
\useexternalfigure[mylogo]
[logo.pdf][width=2cm]
defines an image label mylogo that maps to the image file logo.pdf and sets its width to 2cm. This image label may be used as a normal image filename:
\externalfigure[mylogo]

5 Conclusion
In this article, I briefly explained how to include images in your document. Usually, one wants the included images to have a number and a caption — i.e., to display the image in a float. I will discuss floats in a future article in this series.

○ Aditya Mahajan
adityam (at) ieee dot org
New ČSplain of 2012
Petr Olsák

The ČSplain package has existed since 1994 and it is a gentle extension of plain TeX to make using Czech and Slovak languages feasible. This was the case until October 2012, when the author carried out significant revisions and additions to ČSplain. The basic change resulted from the decision to set the default input encoding of ČSplain to UTF-8. In addition, ČSplain got many other new features: the possibility of loading all available hyphenation patterns, the ability to cooperate with 16-bit TeX engines (LuaTeX, XeTeX), more effective work with fonts including math, easy switching of the internal encoding (including Unicode), and the user-friendly macros OPmac.

In the default configuration, ČSplain remains a gentle extension of plain TeX, backwards-compatible with previous versions. The new possibilities are easily accessed with \input and when they are used it is no longer correct to talk of a gentle extension. On the contrary, it is a strong competitor to all other macro systems based on TeX, even very large ones. ČSplain has advantages in its simplicity, effectiveness, and ease of usage.

The new ČSplain is available through CTAN and the usual TeX distributions, and its home on the web is http://petr.olsak.net/csplain-e.html.

Introduction

In October 2012, a discussion was held on the cstex@ mailing list about the configuration of the input encoding of ČSplain. It was shown that for many years ČSplain used the wrong default input encoding on MS Windows: ISO 8859-2, which is foreign on this operating system. I was surprised.

Our old decision was that the input encoding of ČSplain was to be set depending on the operating system in use. This is similar to the ASCII versus EBCDIC encodings on old systems, where TeX did reencoding of its input depending on its environment. It is essential that when the Czech and Slovak characters in the source file are shown correctly in the text editor then ČSplain prints them correctly too. On the other hand, when we see bad characters in the text editor, we cannot wonder that ČSplain produces broken output. Unfortunately, this idea was valid ten years ago, but not so much today. Nowadays there are text editors with special intelligence—they try to autodetect the encoding and they try to show anything properly. In such an environment, the above rule makes no sense. These modern editors handle the UTF-8 encoding, so we decided that this will be implicitly set as the input encoding of ČSplain on all systems.

The conversion between UTF-8 input codes and the internal encoding (i.e., font encoding and hyphenation pattern encoding) must be done straightforwardly at the input processor level. No active characters are allowed for this purpose. When we do
\begin{verbatim}
def\test#1#2%
  {\the first character is #1, second is #2}
\end{verbatim}

then we expect the output “the first character is č, second is ř”. Therefore, ČSplain needs to activate the encoding extension in 8-bit TeX engines (T2X, pdfTeX). The 16-bit TeX engines are more straightforwardly used for this case.

Format generation

The following lines show various methods to generate the format files csplain and pdfcsplain. The implicit output (DVI and PDF) is set by the name of the generated format (csplain sets DVI output, while pdfcsplain sets PDF output).

```
pdftex -ini -enc "\let\enc=u \input csplain.ini"
pdftex -jobname csplain -ini -etex -enc csplain-utf8.ini
pdftex -jobname pdfcsplain -ini -etex -enc csplain-utf8.ini
xetex -jobname pdfcsplain -etex -ini csplain.ini
latex -jobname pdfcsplain -ini csplain.ini
```

ČSplain — basic features

The basic behavior of ČSplain is similar to plain TeX. The only difference is that the default \hspace and \vspace are set to create one inch margins in A4 paper format, not letter format. One can consider that the second difference is the presence of macros unknown in plain TeX:

```
\chyp % Czech hyphenation patterns and
% \frenchspacing initialised.
\shyp % Slovak hyphenation patterns and
% \frenchspacing initialised.
\csaccents % redefines \', \v etc. expand to
% to expand to given internal slot.
```

You can return to the default behavior with:

```
\ehyp % US hyphenation patterns and
% \nonfrenchspacing.
\cmaccents % \', \v etc. expand to
% \accent primitive.
```

The implicit internal encoding and the implicit fonts are set to ČSencoding/ČSfonts in ČSplain. It
means that (for example) the font \texttt{csr10} is preloaded as \texttt{cmr10} instead of \texttt{cm10}. These \texttt{cs*} fonts keep the 7-bit half of the encoding table the same as their \texttt{cm*} counterparts, while Czech and Slovak letters are placed in the second part of encoding table, ordered by ISO-8859-2.

\texttt{CSplain} defines control sequences which correspond to the special glyphs used in \texttt{CSfonts}.

\begin{verbatim}
\clqq % left Czech double quote.
\crqq % right Czech double quote.
\flqq % left French double quote
% (used at right side in Czech).
\frqq % right French double quote
% (used at left side in Czech).
\promile % per mille character.
\uv % quotation macro: \uv{text} gives
% \texttt{\clqq text\crqq}.
\ogonek a % Polish a-ogonek
% (composed from components)
\end{verbatim}

**UTF-8 input encoding when \texttt{encTeX} is used**

You can recognize the UTF-8 encoded \texttt{CSplain} with \texttt{encTeX} by the message:

The format: csplain <Nov. 2012>.
The cs-fonts are preloaded and A4 size implicitly defined.
The utf8->iso8859-2 re-encoding of Czech+Slovak alphabets:

\begin{verbatim}
\oeOE \oe \oe \oe \oe \oe \oe
\end{verbatim}

These characters are mapped by encTeX to one byte (one slot) corresponding to the internal encoding. Moreover, the characters known from plain \TeX{} are mapped to the control sequences:

\begin{verbatim}
plain: \ss \S s, \l, \L, \ae a, \oe o, \AE A, \OE O
\end{verbatim}

\texttt{encTeX} is able to map the UTF-8 code to the internal 8-bit slot or to the control sequence. When such a mapped control sequence or internal 8-bit slot is processed by the \texttt{\write} primitive, it is converted back to the UTF-8 code. So, the 8-bit \TeX{} engine can handle an unlimited number of UTF-8 codes. But by default, only the characters mentioned above are properly processed by \texttt{CSplain}. If another UTF-8 code occurs in the input, \texttt{CSplain} reports the following warning (the $\~N$ character is used in this example):

```
WARNING: unknown UTF-8 code: \~N = "^c3^91"
```

(line: 42)

and users can add their own mapping and definition of such a character. For example:

\begin{verbatim}
\mubyte\Ntilde ^^c3^^91\endmubyte
% \UTF-8 code mapped to \Ntilde.
\def\Ntilde{\~N} % The \Ntilde is defined.
\end{verbatim}

Now \texttt{CSplain} processes the $\~N$ character properly even though it is not included in the Czech or Slovak alphabets.

The distribution \texttt{enctex.tar.gz} contains these two files:

\begin{verbatim}
utf8lat1.tex % Latin1 Supplement U+0080-U+00FF
utf8lata.tex % Latin Extended-A U+0100-U+017F
\end{verbatim}

These files do the mapping of the abovementioned UTF-8 codes by \texttt{encTeX} and provide the definitions for the mapped control sequences. You can \texttt{\input} them to your document and/or create analogous files for your purposes.

**Internal encoding**

The internal encoding means the encoding of the fonts and hyphenation patterns that are used. By default, \texttt{CSplain} sets the internal encoding to the \texttt{CS}-encoding (as mentioned above). But you can change this encoding via \texttt{\input} at the beginning of your document. There are two possibilities:

```
\input t1code % the T1 internal encoding is set
\input ucode % the Unicode internal encoding
% is set (in 16-bit \TeX{} engines)
```

These \texttt{\input} files do the following:

- Set the correct \texttt{\ucode}/\texttt{\ucode}.
- Reset the \texttt{\chyst} and \texttt{\chyst} macros, so they choose the hyphenation patterns in proper encoding.
- Remap the UTF-8 codes to the new slots, if \texttt{encTeX} is used.
- Redefine some character-like control sequences (\texttt{\ss}, etc.).
- Redefine \texttt{\csaccents}, so \texttt{\x}, \texttt{\N x}, etc. expand to the right slots.

As you can see, these files don’t reload the fonts with the proper encoding. This has to be done with the next \texttt{\input} in your document, for example \texttt{\input lmfonts or ctimes or cs-pagella}.

\texttt{CSplain} preloads the Czech and Slovak hyphenation patterns in \texttt{CS}-encoding, in T1 encoding and

Petr Olšák
(if a 16-bit TeX engine is detected) in Unicode. The only thing the user need be concerned with is initial-
ing the hyphenation patterns with \chyp or \shyp after the \input tcode or \input ucode is done. The section below, “More languages”, de-
scribes how \csplain is able to load hyphenation pat-
ters of another languages.

Font loading
The \csplain package provides the following ready-
to-use files which load the given font family (typi-
cally \rm, \it, \bf and \bi):

\begin{itemize}
  \item lmffonts \% Latin Modern fonts
  \item ctimes \% Times
  \item chelvet \% Helvetica
  \item cavantga \% AvantGarde
  \item cncent \% NewCentury
  \item cpalatin \% Palatino
  \item cs-terms \% TeX-Gyre Termes (Times)
  \item cs-heros \% TeX-Gyre Heros (Helvetica)
  \item cs-cursor \% TeX-Gyre Cursor (Courier)
  \item cs-adventor \% TeX-Gyre Adventor (AvantGarde)
  \item cs-bonum \% TeX-Gyre Bonum (Bookman)
  \item cs-pagella \% TeX-Gyre Pagella (Palatino)
  \item cs-schola \% TeX-Gyre Schola (NewCentury)
  \item cs-antt \% Antykwa Torunska
  \item cs-polta \% Antykwa Poltawskiego
  \item cs-bera \% Bera
  \item cs-arev \% ArevSans
  \item cs-charter \% Charter
\end{itemize}

All of these font files include the switch to load the correct font for the chosen internal encoding (\cs-
encoding or T1 or Unicode). These font files simply load the fonts for the needed variants with the \font
primitive, redefining the control sequences \tenrm, \tenit, \tenbf, \tenbi and \tenbb. Again, users can easily create their own additional font files by using these as a model.

The font loading files do not deal with the vari-
sions of the fonts, because they do not need to. That is the subject of the next section.

Font handling
\csplain introduces a simple font-resizing principle. The main credo is: “power is in simplicity”. That is the reason why I don’t use NFSS, for example.

The command \font\foo=something declares font selector \foo which selects the font something. The terminology font selector in this section is used only for selectors declared by the \font primitive. This means that \bf (for example) isn’t a font se-
ector. It is a macro.

\csplain defines the following macros for font size handling.

\begin{itemize}
  \item \resizefont\foo resizes the font represented by font selector \foo. More precisely, it de-
  clares (locally) \foo as the same font but with the size given in the macro \sizespec. The \sizespec macro can have the form at\langle\dimen\rangle or scale\langle\factor\rangle.
  \item \regfont\foo registers the font selector \foo as a resizable font. By default \csplain declares the following selectors with \regfont: \tenrm, \tenit, \tenbf, \tenbi and \tenbb. Users can declare more selectors.
  \item \resizeall resizes (locally) all registered font selectors to the size given by the \sizespec macro.
  \item \letfont \foo=\bar at\langle\dimen\rangle or \letfont \foo=\bar scaled\langle\factor\rangle declares a new font
  selector \foo as the same font as \bar with the given size. The \bar font selector is unchanged.
\end{itemize}

Here’s an example:

\begin{verbatim}
\font\zapfchan=pczm18z \regfont\zapfchan
\def\sizespec{at13.5pt} \resizeall \tenrm \baselineskip=15pt
\end{verbatim}

Here is the typesetting at size 13.5pt including \{\it italics\}, \{\bf bold\} and including the \{\zapfchan Zapf Chancery font\}.

\begin{verbatim}
\def\sizespec{at8pt} \resizeall \tenrm
\end{verbatim}

Now all the typesetting is at the 8pt size.

Another example uses the font loading files:

\begin{verbatim}
\input chelvet \% \tenrm, \tenit, etc. is now \% the Helvetica family.
\letfont\titlefont = \tenbf at14.4pt \% \titlefont is for titles:
\% Helvetica Bold at14.4pt.
\input ctimes \% \tenrm, etc. is Times Roman.
\def\sizespec{at11pt} \resizeall \tenrm \% Normal text will be typeset
\% by Times Roman at11pt.
\def\small{\def\sizespec{at9pt}\resizeall \tenrm} \% The \small macro switches the whole family
\% of Times Roman to the 9pt size, \% e.g., for footnotes.
\end{verbatim}

Note #1. The font selectors \tenrm, \tenit, etc. have the subword ten in its name but this is only for historical reasons. The current meaning of these selectors can be fonts at an arbitrary size.

Note #2. These macros do not solve the resizing of math fonts. This is the subject of the following section.

Note #3. The selection of the proper design size (cmr5 or cmr7 or \ldots or cmr17) is not solved by default. But the math font macros solve this and you can simply redefine \resizefont so that the proper design size is selected.
Math fonts

The \texttt{CSplain} package provides two macro files for math fonts: \texttt{ams-math.tex} and \texttt{tx-math.tex}. The first one loads \texttt{AMS} fonts and declares hundreds of math symbols and operators like \texttt{AMS\LaTeX}. The second macro file does the same but loads the \texttt{tx} fonts which are visually compatible with Times Roman and similar designs.

By default, neither of these macro files are read. But you can load \texttt{ams-math.tex} explicitly, or the proper macro file is loaded implicitly with \texttt{\input ctimes, lmfonts, etc.}

These files provide the macro:

\begin{verbatim}
\setmathsizes{[\text/the]/\scriptscriptscriptscript\text}\
\end{verbatim}

in which the user can set the sizes of basic text, script and superscript. The parameters have to be written without unit (the unit pt is used). For example \texttt{\setmathsizes[10/7/5]} is the default from \texttt{plain \LaTeX}.

The following math alphabets are available after \texttt{ams-math.tex} or \texttt{tx-math.tex} is loaded:

\begin{verbatim}
\mit % mathematical variables  \rm, \it % text fonts in math
\bf, \bi % bold sans fonts (might be % different than text fonts)
\cal % normal calligraphic  \script % script
\frak % fraktur  \bbchar % double stroked letters
\end{verbatim}

The \texttt{ams-math.tex} defines the \texttt{\regtfm} macro to declare the mapping from a desired size to the list of design sizes represented by names of the metric files. For more information about this, see the file \texttt{ams-math.tex}, where \texttt{\regtfm} is defined and used. Once this mapping is set, you can redefine the internal subpart of the \texttt{\resizefont} macro in the following way:

\begin{verbatim}
\def\resizefontskipat#1 #2\relax
  \{\whichtfm[#1] \sizespec\relax\}
\end{verbatim}

Now \texttt{\resizefont} chooses the right metrics if \texttt{\sizespec} and \texttt{\dgsize} are properly set. This complexity can be hidden from the user, if he or she uses the \texttt{\typosize} and \texttt{\typoscale} macros from OPmac.

The following example shows how to set the font for a title that includes math formulas:

\begin{verbatim}
\def\titlefont{\def\at14pt{\resizefont\tenbf
  \tenbf \setmathsizes[14/9.8/7]\boldmath}
  \def\vpar{\centerline{\titlefont #1}}
\title More about \int_x^{\infty} f(t)\{\text{rm d}\}t$
\end{verbatim}

The \texttt{\boldmath} command selects the alternative set of all math families more compatible with \texttt{bold} fonts usually used in titles.

Unicode fonts

Historically, \texttt{CSplain} worked with 8-bit \TeX\ engines where Unicode fonts are impossible. So, all the font handling mentioned so far is primarily intended for 8-bit fonts. The Unicode support for text fonts in \texttt{CSplain} is only experimental, and Unicode math isn’t solved in \texttt{CSplain} at all.

The 16-bit \TeX\ engines expect the UTF-8 input encoding and work in Unicode internally. So T1-encoded fonts cannot be used because Czech and Slovak alphabets are unfortunately not in the intersection of T1 and Unicode encodings. On the other hand, colleagues writing in German or French can use T1-encoded 8-bit fonts in 16-bit \TeX\ engines because their whole alphabet is in this intersection.

\texttt{Xe\TeX} has a font loader linked with system libraries and it extends the syntax of the \texttt{\font} primitive. For example:

\begin{verbatim}
\font\foo=\[\texttt{filename}\](\texttt{\fontfeatures}) \(\texttt{\sizespec}\)
\end{verbatim}

where \texttt{\texttt{filename}} is the file name without the .\texttt{otf} suffix and the \texttt{\sizespec} is \texttt{\at\dimen} or \texttt{\scaled \\factor}. The \texttt{\fontfeatures} are font modifiers separated by semicolon. You have to which know which features are implemented in the font and which in the font loader. For example, \texttt{Xe\TeX}'s font loader provides the feature \texttt{mapping=\texttt{tex-text}} which activates the usual \texttt{\TeX} ligatures like \texttt{{\texttt{--} \texttt{--}}} The normal ligatures (e.g., ‘fi’) are activated implicitly.

On the other hand, \texttt{Lua\TeX} implements its extension of the font loader by Lua code. I have extracted the core of this code (from \texttt{luatofload.sty}) for \texttt{CSplain}, in a file \texttt{luafonts.tcl}. Its stability can’t be guaranteed because the Lua functions from the \texttt{Lua\TeX} distribution are called, and they may change in the future. If \texttt{Lua\TeX} is being used, the \texttt{\\cs-heros.tex, cs-termes.tex, Cs-heros.tex, etc.} input \texttt{luafonts.tcl} before the first usage of the extended \texttt{\font} primitive.

The extension of the \texttt{\font} primitive seems to have the same syntax in \texttt{Xe\TeX} and \texttt{Lua\TeX}. But, unfortunately, the font features are different. By default, no ligatures are activated in Unicode fonts in \texttt{Lua\TeX}. Users must use \texttt{script=latn} to activate the \texttt{fi-ligatures} and \texttt{+tlig} to activate the \texttt{\TeX} special ligatures. Users can define the \texttt{\fontfeatures} macro for special needs of features. If this macro isn’t defined, \texttt{CSplain}'s font-loading macros make the following default:
\def\fontfeatures
{mapping= tex-text; script= latn; +tlig}
which works in both X\TeX and Lua\TeX.

More languages

The following hyphenation patterns are preloaded in \texttt{\textsc{C}splain} by default:

- \texttt{\textbackslash USenglish=0} ... default US hyphenation patterns from plain \TeX, ASCII encoding.
- \texttt{\textbackslash czILtwo=5} ... Czech patterns, ISO-8859-2.
- \texttt{\textbackslash skILtwo=6} ... Slovak patterns, ISO-8859-2.
- \texttt{\textbackslash czCork=15} ... Czech patterns, T1 encoding.
- \texttt{\textbackslash skCork=16} ... Slovak patterns, T1 encoding.
- \texttt{\textbackslash czUnicode=115} ... Czech patterns, Unicode (only for 16-bit \TeX engines).
- \texttt{\textbackslash skUnicode=116} ... Slovak patterns, Unicode (only for 16-bit \TeX engine).

Hyphenation patterns are selected with \texttt{\textbackslash uslang}, \texttt{\textbackslash czlang} and \texttt{\textbackslash sklang}, which are equivalent to the old selectors \texttt{\textbackslash ehyph}, \texttt{\textbackslash chyph} and \texttt{\textbackslash shyph}. The proper encoding is used if the command \texttt{\textbackslash input t1code} or \texttt{\textbackslash input ucode} precedes the patterns selector.

Since 2012, \texttt{\textsc{C}splain} is able to load hyphenation patterns of other languages (ca. 50 languages). If the patterns use a subset of T1 encoding, they can be loaded in T1 (alias Cork) and/or in Unicode. Otherwise, only the Unicode encoding for the patterns is allowed. Unicode patterns can be loaded only in 16-bit \TeX engines.

The loading of extra hyphenation patterns can be done on the command line when format is generated. Examples follow:

```
\texttt{pdftex -ini -enc \}
  "\texttt{\textbackslash Input csplain.ini}\texttt{"
\texttt{pdftex -ini -enc "\textbackslash Input csplain.ini\
\texttt{\textbackslash Input csplain.ini\textbackslash Input csplain.ini"
\texttt{luatex -jobname pdfcsplain -ini \}
  "\texttt{\textbackslash Input csplain.ini\textbackslash Input csplain.ini\textbackslash Input csplain.ini"
\texttt{luatex -jobname pdfcsplain -ini \}
  "\texttt{\textbackslash Input csplain.ini\textbackslash Input csplain.ini\textbackslash Input csplain.ini"

The first line adds Polish hyphenation patterns in the T1 encoding to \texttt{\textsc{C}splain}. The second line loads all available hyphenation patterns for 8-bit \TeX engines (i.e. Czech&Slovak in ISO-8859-2 and T1, and others, ca. 30 languages, in T1). The third line loads the Russian hyphenation patterns in Unicode. Finally, the last line loads all available hyphenation patterns (in T1 and in Unicode). The pattern selectors have the form \texttt{\langle\textbackslash twoletters\rangle lang}, for example \texttt{\textbackslash pllang}, \texttt{\textbackslash delang}, \texttt{\textbackslash itlang}, \texttt{\textbackslash rulang} etc. Please read the \texttt{hyphen.lan} file for more information.

The \texttt{\textsc{O}Pmac} macro package

The \texttt{\textsc{O}Pmac} (Olsak’s Plain macros) package is part of \texttt{\textsc{C}splain}. It provides more \LaTeX-like features in plain \TeX: font size changing, automatic creation of tables of contents and indexes, working with bibliography databases, tables, references including hyperlinks options, etc. For more information about this macro package, see the companion article in this same issue of \textit{TUGboat}.

\diamond \ Petr Olšák
\v{C}esk{\v{r}}e Technick{\v{u}} Universit{\v{a}}
\v{r}i Praha
\v{C}esk{\v{r}}e Republik{\v{a}}

New \texttt{\textsc{C}splain} of 2012
OPmac: Macros for plain \TeX

Petr Olšák

The OPmac package provides simple additional macros on top of plain \TeX. It enables users to take advantage of basic \LaTeX functionality: font size selection, automatic creation of tables of contents and indices, working with bibliography databases, tables, references optionally including hyperlinks, margin settings, etc. In this paper, the significant properties of OPmac are described. The complete source of the macros, as well as user and technical documentation, is available through CTAN and the usual \TeX distributions, and its home on the web is http://petr.olsak.net/opmac-e.html.

Introduction

I have decided to publish my macros together with the new version of CS\plain. I have been using these macros for a long time for many purposes in my own work. Now, I have made them cleaner, added user and technical documentation, and released them.

The main reason is to give a set of macros which solves common authorial tasks for plain \TeX users. A side benefit is that the macros demonstrate that it is possible to do \TeX code simply and effectively. Most \LaTeX macro packages don't have this feature. All macros are in the single (documented) file opmac.tex with only 1500 lines. On the other hand the \LaTeX code which solves comparable tasks is placed in a kernel and dozens of \LaTeX packages with many tens of thousands of lines in total.

Here are the main principles which I followed when creating this macro package:

- Simplicity is power.
- Macros are not universal, but are readable and understandable.
- Users can easily redefine these macros as they wish.

Each part of the macro code is written to maximize readability for humans who want to read it, understand it and change it.

The OPmac package offers a markup language for authors of texts (like \LaTeX), i.e. a fixed set of tags to define the structure of a document. This markup is different from \LaTeX markup. It offers the possibility of writing the source text of a document somewhat more clearly and attractively. The OPmac package, however, does not deal with the many possible typographic designs of a document. A simple, sober document is created if no additional macros are used. We assume that authors will be able to modify the look of the document to suit their requirements. You can see a complex example of using OPmac with added macros for typesetting design at http://petr.olsak.net/ctustyle.html: CTUstyle is the recommended design style for bachelor, master or doctoral theses at Czech Technical University in Prague.

The following text is a short digest of the documentation. It illustrates the capability of the OPmac package.

Using OPmac

OPmac is not compiled as a format. To use it in plain \TeX, you can simply \texttt{\input opmac} at the beginning of your document. Here's a trivial document as a first example:

\begin{verbatim}
\input opmac
\typosize[11/13] % set basic font size
% and baselineskip
\margins/1 a4 (1,1,1,1)in % set 1in margins
\bye
\end{verbatim}

Font sizes

The commands for font size setting described here are all local. In other words, if you use them in a \TeX group, the font sizes are selected locally within the group, not globally.

The command

\begin{verbatim}
\typosize[(fontsize)/(baselineskip)]
\end{verbatim}

sets the font size of text and math fonts and the baselineskip. If one of the two parameters is empty, the corresponding feature stays unchanged. The metric unit is pt by default; this unit isn't written in the parameter values. You can change the unit by the command \texttt{\ptunit=(something-else)}, for instance \texttt{\ptunit=1mm}. Examples:

\begin{verbatim}
\typosize[10/12] % default in plain \TeX
\typosize[11/12.5] % font size 11pt, % baselineskip 12.5pt
\typosize[8/] % font size 8pt, % baselineskip left unchanged
\end{verbatim}

The command

\begin{verbatim}
\typoscale[(font-factor)/(baselineskip-factor)]
\end{verbatim}

sets the text and math fonts size and baselineskip to a multiple of the current font size and baselineskip. The factor is written like \TeX's \texttt{scaled} values, meaning that 1000 leaves the value as-is. An empty parameter is equivalent to 1000. Examples:
The sizes declared by these macros (for example in titles) are relative to the basic size selected for the font (this may be an arbitrary size, not only 10pt). The size of the current font can be changed with the command \texttt{\textbackslash thefontsize[(font-size)]} or rescaled with \texttt{\textbackslash thefontsize[(factor)]}. These macros do not change the math font sizes or the baselineskip.

The commands \texttt{\textbackslash resizefont}, \texttt{\textbackslash regfont} and \texttt{\textbackslash resizeall} are available for generally resizing fonts. They’re described in the companion article on CS plain (pp. 83–87), but can be used with OPmac alone; CS plain need not be the format. The best design size of the font for desired size is used. For example, with Computer Modern, \texttt{\textbackslash typosize[18/]} selects the font \texttt{cmr17} at 18pt.

### Parts of the document

A document can be titled and divided into chapters, sections and subsections. The parameters have to be ended with an empty line (no braces are used):

\texttt{\tit Document title \{empty line\}
\chap Chapter title \{empty line\}
\sec Section title \{empty line\}
\secc Subsection title \{empty line\}}

Chapters are numbered with one number, sections by two numbers (\texttt{\{chapter\}.\{section\}}) and subsections by three numbers (similarly). If there are no chapters then sections have only one number and subsections two.

The design of the chapter etc. titles are implemented in the macros \texttt{\textbackslash printchap}, \texttt{\textbackslash printsec} and \texttt{\textbackslash printsecc}. Users can simply change these macros to get their desired output.

The first paragraph after the title of chapter, section and subsection is not indented by default; giving \texttt{\textbackslash let\{firstnoindent\}=	extbackslash relax} makes all paragraphs indented.

If a title is long enough, it breaks across multiple lines. It is better to explicitly give the breakpoints because \TeX{} cannot interpret the meaning of the title. Users can insert the \texttt{\textbackslash nl} (meaning newline) macro to specify the breakpoints.

### Other numbered objects

Apart from chapters, sections and subsections, there are other automatically-numbered objects: equations and captions for tables and figures.

If \texttt{\textbackslash eqmark} is given as the last element in a math display then this equation is numbered. The format is one number in brackets. This number is reset in each section.

In displays using \texttt{\textbackslash eqalignno}, \texttt{\textbackslash eqmark} can be given in the last column before \texttt{\textbackslash cr}. For example:

\texttt{\textbackslash eqalignno{
  a^2+b^2 & = c^2 \textbackslash cr
  c & = \sqrt{a^2+b^2} & \textbackslash eqmark \textbackslash cr}}

The next numbered object is captions; these are tagged with \texttt{\textbackslash caption/t} for tables and \texttt{\textbackslash caption/f} for figures. Example:

\texttt{\textbackslash hil\{table\}\{rl\}{
  \hfil\table{rl}{
    \textbackslash noalign{\smallskip}
    \hfil age & value \cr
    0--1 & unmeasurable \cr
    1--6 & observable \cr
    6--12 & significant \cr
    12--20 & extreme \cr
    20--40 & normal \cr
    40--60 & various \cr
    60--\infty & moderate\cr}
  \caption/t The relationship of computer-dependency to age.\par
}}

This example produces:

\begin{tabular}{rl}
  age & value \\
  0--1 & unmeasurable \\
  1--6 & observable \\
  6--12 & significant \\
  12--20 & extreme \\
  20--40 & normal \\
  40--60 & various \\
  60--\infty & moderate
\end{tabular}

\textbf{Table 2.3} The relationship of computer-dependency to age.

The word “Table” followed by a number is added by the macro \texttt{\textbackslash caption/t}. The macro \texttt{\textbackslash caption/f} creates the word figure. The caption text is centered. If it occupies multiple lines then the last line is centered.

The added word (table, figure) depends on the value of the \texttt{\language} register. OPmac implements the mapping from \texttt{\language} numbers to languages and the mapping from languages to the generated words.

To make the table or figure a floating object, you can use the plain \TeX{} macros \texttt{\midinsert}, \texttt{\topinsert} and \texttt{\endinsert}.

A \texttt{\textbackslash label{[\{label\}]} command preceding the automatically-numbered object allows symbolic referencing to the object. The reference commands

OPmac: Macros for plain \TeX
are \ref{\label} (for the value of the number) and
\pgref{\label} (for the page number). Example:
\label[beatle] \sec About The Beatles
...
\label[comp-dependence]
\hfil\table{rl}{...} % the table
\caption{The relationship of computer-dependency to age.}
...
\label[pythagoras]
$$ a^2 + b^2 = c^2 \eqmark $$

Now we can point to the section~\ref[beatle] on
the page~\pgref[beatle] or write about the
equation~\ref[pythagoras]. Finally there
is an interesting Table~\ref[comp-dependence].

Lists

A list of items is surrounded by \begitems and
\enditems commands. The asterisk (*) is active
within this environment and it starts one item. The
item style can be chosen by \style parameter writ-
ten after \begitems:

\style o % small bullet
\style 0 % big bullet (default)
\style - % hyphen char
\style n % numbered 1., 2., 3., ...
\style N % numbered 1), 2), 3), ...
\style i % roman numerals (i), (ii), (iii), ...
\style I % Roman numerals I, II, III, ...
\style a % lettered a), b), c), ...
\style A % Lettered A), B), C), ...
\style x % small rectangle
\style X % big rectangle

Another style can be defined with the command
\sdef{item: ⟨style⟩}{{⟨text⟩}}. The default style can
be redefined with \def\normalitem{{⟨text⟩}}. List
environments can be nested. Each new level of item
is indented by next multiple of \iindent which is set to \parindent by default.

Table of contents

The \maketoc command prints a table of contents
of all \chap, \sec and \secc titles used in the
document. The text is read from an external file, so you
have to run \TeX\ more than once (typically three times if the table of contents is at the beginning of
the document).

A section name for the table of contents itself
is not printed. The usage of \chap or \sec isn’t
recommended here because the table of contents is
typically not referenced to itself. You can print the
unnamed (and unreferenc-able) title with the
code:

\def\thesecnum{}
\printsec{\unskip Table of Contents}
\maketoc

The titles of chapters etc. are written to an ex-
ternal file and then read from this file in a sub-
sequent run of \TeX. This technique can create prob-
lems when a somewhat complicated macro is used
in a title. OPmac solves this problem in a differ-
ent way than \La\TeX: users declare the problematic
macro as “robust” via an \addprotect\macro decla-
ration. The \macro itself cannot be redefined. The
common macros used in OPmac which are likely to
occur in titles are already declared in this way.

Making an index

An index can be included in a document with the
\makeindex macro. No external program is needed:
the alphabetical sorting is done inside \TeX\ at the
macro level.

The \ii command (insert to index) declares the
following word, terminated by a space, as the index
item. This declaration is represented as an invisible
atom on the page connected to the next visible word.
The page number of the page where this atom occurs
is listed in the index entry. So you can type:

The \ii resistor resistor is a passive
electrical component ...

You can avoid doubling the word by using \iid
instead \ii:

The \iid resistor is a passive
electrical component ...

Now we’ll deal with the \iid resistor .

As shown, a period or comma has to be sepa-
rated from the word by a space when \iid is used.
This space (before the punctuation) is removed by
the macro in the current text.

If you need to have an actual space in an index
entry, use “~”. For example:

\ii linear~dependency Linear dependency of ...

Multiple-word entries are often organized in the
index in the format (for example):

linear dependency 11, 40–50
— independence 12, 42–53
— space 57, 76
— subspace 58

To do this you have to declare the parts of the
words with the / separator. Example:

{\bf Definition.}
\ii linear/space, vector/space
{\it Linear space} (or {\it vector space}) is ...
The number of parts in one index entry is unlimited. You can save typing via commas in the \ii parameter: the previous example is equivalent to \ii linear\slash space \ii vector\slash space.

Another need is to propagate to the index the “reversed” terms; e.g. given linear\slash space, you also want to index space\slash linear. You can do this conveniently with the shorthand ,@ at the end of the \ii parameter. For example:

\ii linear\slash space, vector\slash space, @

is equivalent to:

\ii linear\slash space, vector\slash space
\ii space\slash linear, space\slash vector

The \makeindex macro creates the list of alphabetically sorted index entries with no section title and without using multiple columns. OPmac provides another macro for multi-column typesetting:

\begmulti (number of columns)
\text
\endmulti

The columns will be balanced. The index title can be printed with \sec. So an index in an OPmac document might look like this:

\sec Index\par
\begmulti 3 \makeindex \endmulti

Only “pure words” can be propagated to the index with the \ii command; there cannot be any macros, \TeX primitives, math selectors etc. OPmac provides another way for create such complex index entries: use a “plain text equivalent” as the \ii parameter, and map this equivalent to the desired \TeX word which is printed in the index entry with the \iis command. Here’s an example:

The \ii chiquadrat $\chi$-quadrat method is ...
If the \ii relax \relax command is used then \TeX is relaxing.
...
\ii chiquadrat \{\$\chi\$-quadrat\}
\ii relax \{\tt \char`\`\relax\}
...

The \iis (equivalent) \{\text\} creates one entry in the “dictionary of the exceptions”. The sorting is done by (equivalent), while \text is printed in the index entry list.

Czech/Slovak standard alphabetical sorting is used if the \language register is set to the Czech or Slovak hyphenation patterns when \makeindex is in progress. (The main difference from English sorting is that “ch” is treated as one character between “h” and “i”.)

Colors

The color selection macros work only if a pdf\TeX-like engine is used. OPmac provides a small number of color selectors: \Blue, \Red, \Brown, \Green, \Yellow, \White, \Grey, \LightGrey and \Black. Users can define more such selectors by setting the CMYK components. For example:

\def\Orange{\setcmykcolor{0 0 0.5 0}}

The selectors change the color of the text and of lines with a thickness larger than 1bp. If \linecolor immediately precedes the color selector then the lines with a thickness less than or equal to 1bp are colored. This is a second independent color setting.

The color selectors work globally starting on the current page. If the colored text continues to the next page, the color is correctly set on the following page(s) after a second run of \TeX, because this event is implemented via external file. Users can also write \localcolor inside a group. This command saves the current color and restores it after the group is completed. By default, it is assumed that the group corresponds to the boundary of a box which cannot break across pages. If this is not true, \longlocalcolor can be used instead of \localcolor. A basic example:

\Red the text is red
\bbox{\localcolor \Blue here is blue}
{\localcolor \Green and green}
restored blue \Brown and brown
now the text is red again.

A more usable example follows. Let’s define a macro which creates colored text on a colored background, to be used like this:

\coloron\{background\}\{foreground\}\{\text\}

Such a macro can be defined and used like this:

\def\coloron#1#2#3{\setbox0=\hbox{#3}\leavevmode
{\localcolor\rlap{#1\strut\vrule width\wd0}#2\box0}}

\coloron\Yellow\Brown{Brown text on a yellow background}

PDF hyperlinks and outlines

If the command
\hyperlinks{\{color-int\}\\{color-ext\}}

is used at the beginning of the file, then the following are hyperlinked when PDF output is used:
numbers generated by \ref or \pgref,
numbers of chapters, sections and subsections in the table of contents,
numbers or marks generated by \cite command (bibliography references),
texts printed by \url command.

The last object is an external link and it is colored by \hyperlinks (color-ext). Others links are internal and they are colored by \hyperlinks (color-int). Example:
\hyperlinks \Blue \Green % internal links blue, links: frames which are visible in the PDF viewer but invisible when the document is printed. To do this, define the macros \pgborder, \tocborder, \citeborder, \refborder and \urlborder to be the RGB color value (a triple) to use. Examples:
\def\tocborder{0 0 1} % links in toc:
% red frame
\def\pgborder{0 1 0} % links to pages:
% green frame
\def\citeborder{0 0 1} % links to references:
% blue frame

By default these macros are not defined, so no frames are created.

There are “low level” commands to create the links. You can specify the destination of an internal link with \dest[(type):(label)]{(height)}. Active text linked to the \dest can be created with \link[(type):(label)]{(color)}{(text)}. The \type parameter is one of toc, pg, cite, ref or one user-defined for your purposes. The \height parameter gives the vertical distance between the actual destination point and the current baseline.

The \url macro prints its parameter in the \ttt font and inserts potential breakpoints (after slash or dot, for example). If the \hyperlinks declaration is used then the parameter is treated as an external url link. An example:
\url{http://www.olsak.net}

The PDF format also provides for “outlines” which are notes placed in a special frame of a PDF viewer. These notes are usually managed as a structured and hyperlinked table of contents of the document. The command \outlines{(level)} creates such an outline from the table of contents data in the document. The \level parameter gives the default level of opened outlines. Deeper levels can be opened by (typically) clicking on the triangle symbol after that.

The command \insertoutline{(text)} inserts next entry into “outlines” at the main level 0. This entry can be placed before table of contents (created by \outlines) or after it.

Verbatim

Display verbatim text in OPmac is surrounded by the \begtt and \endtt pair. Inline verbatim is tagged (before and after) by a character declared with \activettchar{char}. For example \activettchar{1} makes the 1 character do inline verbatim markup, as in the \texttt{TUGboat} style.

If the numerical register \ttline is set to a non-negative value then display verbatim numbers the lines. The first line is numbered \ttline+1 and when the verbatim display ends, the \ttline value is equal to the number of last line printed. The next \begtt...\endtt environment will continue the line numbering. OPmac sets \ttline=-1 by default.

The indentation of lines in display verbatim is controlled by the \tindent register. This register is set to \parindent at the time opmac.tex is read. Users should change its value as desired, e.g. if \parindent is changed after opmac.tex is read.

The \begtt starts a group in which the catcodes are changed. Then the \tthook macro is run. This macro is empty by default; users can control fine behavior with it. For example, more catcodes can be reset here. To define an active character in \tthook, you can use \adef as in this example:
\def\tthook{\adef{1}{\char'\ }}
\begtt
Each occurrence of the exclamation mark will be changed to the question mark and vice versa. Really? You can try it!
\endtt

The \adef command sets its parameter as active after the body of \tthook is read. So you need not worry about active definitions beforehand.

Here are some tips for global \tthook definitions:
\% setting font size for verbatim:
\def\tthook{\parindent[9/11]}
% each listing is numbered from 1:
\def\tthook{\ttline=0}
% visible spaces:
\def\tthook{\adef{ }{\char'\ }}

You can print a verbatim listing of an external file with the \verbinput command. Examples:
\% whole file program.c is printed:
\verbinput (-) program.c
% only lines 12-42:
\verbinput (12-42) program.c
% from beginning to line 60:
\verbinput (-60) program.c
% from line 61 to the end:
\verbinput (61-) program.c
% starting at line 70, only 10 lines printed:

Petr Olšák
The \texttt{ttline} influences the line numbering in the same way as the \texttt{begtt...endtt} environment. If \texttt{ttline=1} then real line numbers are printed; this is the default. If \texttt{ttline < -1} then no line numbers are printed.

The \texttt{verbatim} output can be controlled by \texttt{thook} and \texttt{tindent}, also just as with \texttt{begtt...endtt}.

### Tables

The macro \texttt{\table{⟨declaration⟩}{⟨data⟩}} provides \texttt{⟨declaration⟩} similar to L\TeX\: you can use the letters \texttt{l}, \texttt{r}, and \texttt{c}, with each letter declaring one column aligned to left, right, center respectively. These letters can be combined with the \texttt{“|”} character to create a vertical line.

The command \texttt{\cr} ends a row as usual. OPmac defines the following similar commands:

- \texttt{\crll} ends the row, with a horizontal line after.
- \texttt{\crlli} is like \texttt{\crll}, but the horizontal line doesn’t intersect any vertical double lines.
- \texttt{\crllii} is like \texttt{\crlli}, but horizontal line is doubled.

Basic example:

\begin{verbatim}
\table{|c||l||r|}{
\multispan3\vrule\hss\bf Title\hss
\vrule\tabstrut\crl
\noalign{\kern\hhkern}\crli
first & second & third \crlli
seven & eight & nine \crli}
\end{verbatim}

creates the following result:

<table>
<thead>
<tr>
<th></th>
<th>first</th>
<th>second</th>
<th>third</th>
</tr>
</thead>
<tbody>
<tr>
<td>July</td>
<td>seven</td>
<td>eight</td>
<td>nine</td>
</tr>
</tbody>
</table>

The rule width of tables (and the implicit width of all \texttt{\vrule}s and \texttt{\hrule}s) can be set by the command \texttt{\rulewidth=⟨dimen⟩}. The default value set by \TeX\ is 0.4pt.

### Images

The command \texttt{\inspic⟨filename⟩⟨extension⟩⟨space⟩} inserts the image in the file \texttt{⟨filename⟩⟨extension⟩}. Before the first \texttt{\inspic} command, you have to set the picture width with \texttt{\picw=⟨dimen⟩}. Images can be in PNG, JPG, JBIG2 or PDF format. The \texttt{\inspic} command works with pdf\TeX\ only.

### PDF transformations

All typesetting elements are transformed in pdf\TeX\ by a linear transformation given by the current transformation matrix. The \texttt{\pdfsetmatrix{(a) (b) (c) (d)}} command creates an internal multiplication with the current matrix, so linear transformations can be composed. The commands \texttt{\pdfsave} and \texttt{\pdfrestore} allow for storing and restoring the current transformation matrix.

OPmac provides the macros

\begin{verbatim}
\pdfscale{(horizontal-factor)}{(vertical-factor)}
\pdfrotate{(angle-in-degrees)}
\end{verbatim}
These macros simply expand to the proper `\pdfsetmatrix` command.

Footnotes and marginal notes

Plain \TeX{}'s macro `\footnote` is not redefined, but a new macro `\note{⟨text⟩}` is defined. The footnote mark is added automatically and it is numbered on each page from one. The `⟨text⟩` is scaled by `\typoscale[800/800]`. The footnote mark is typeset with `\def\thefnote{⟨\locfnum⟩$\{$\locfnum$\}$}` by default. Users can redefine this; for example:

```
\def\thefnote{\ifcase\locfnum\or$^\ddag$\or$^\dag\ddag$\or
$^\dag$\or**\or***\or$^\dag$\fi}
```

The `\note` macro is fully applicable only in “normal outer” paragraphs. It doesn’t work inside boxes (tables for example). If you are in such a case, you can use `\notemark[⟨number⟩]` inside the box (only the footnote mark is generated). When the box is finished you then use `\notetext{⟨text⟩}` to define the text for footnote `⟨number⟩`. The `⟨number⟩` after `\notemark` has to be 1 if only one such command is in the box. The second `\notemark` inside the same box have to use the value 2 etc. The same number of `\notetexts` have to be defined after the box as the number of `\notemarks` inserted inside the box.

Marginal notes can be printed by the macro `\mnote{⟨text⟩}`. The `⟨text⟩` is placed in the right margin on odd pages and the left margin on even pages. This is done after a second \TeX{} run because the relevant information is stored in an external file. If you want to place the notes only to a fixed margin, write `\fixmnotes\right` or `\fixmnotes\left`.

The `⟨text⟩` is formatted as a little paragraph with maximal width `\notessize`, ragged right in the left margins and ragged left in the right margins. The first line of this little paragraph is at the same height as the invisible mark created by `\mnote` in the current paragraph. Exceptions are possible via the `\notesskip` register. You can implement such exceptions to each `\mnote` manually, e.g., in a final printing so that margin notes do not overlap.

\bibTeX{}ing

The command `\cite{⟨label⟩}` makes citations of the form [42]. Multiple citation labels are also allowed, as in `\cite{⟨label1⟩,⟨label2⟩,⟨label3⟩}` producing [15, 19, 26]. If `\shortcitations` is given at the beginning of the document then continuous sequences of numbers are collapsed: [3–5, 7, 9–11].

The printed numbers correspond to the same numbers generated in the list of references. This list can be created manually by `\bib{⟨label⟩}` command for each entry. Example:

```
```

There are two other possibilities which use \bibTeX. The first is based to the command

```
\usebibtex{⟨bib-base⟩}{⟨bst-style⟩}
```

which creates the list of cited entries and entries indicated by `\nocite{⟨label⟩}`. After the first \TeX{} run, `\jobname.aux` is created, so users have to run \bibTeX\ with the command `\bibtex ⟨document⟩`. After a second \TeX{} run, \bibTeX\’s output is read, and after a third run all references are properly created.

The second possibility is based on a pre-generated .bbl file by \bibTeX. You can create the temporary file (`mybase.tex`, let’s say) which looks like this:

```
\input opmac
\genbbl{⟨bib-base⟩}{⟨bst-style⟩}
\end
```

After a first \TeX{} run, `mybase.aux` is generated. Then you can run `\bibtex mybase` which generates the .bbl file with all entries from the (bib-base).bib file. The second \TeX{} run on the file `mybase.tex` generates the printed form of the list of all bib entries with labels. Finally you can insert to your real document one of the following commands:

```
\% print all entries from mybase.bbl (a=all):
\usebbl/a mybase
\% print only \cited and \nocited entries \% sorted by mybase.bbl (b=bbl):
\usebbl/b mybase
\% print only \cited and \nocited entries \% sorted by \cite-order (c=cite):
\usebbl/c mybase
```

Sometimes a pure \bibTeX{} command occurs (unfortunately) in a .bib database or \bibTeX\ style. OPmac users can define such commands in the `\bibtexhook` macro, which is expanded inside the group before the .bbl file is read. Example:

```
\def\bibtexhook{
  \def\emp#1{{\em#1}}
  \def\frac#1#2{{#1/#2}}
}
```

Petr Olšák
Setting the margins

OPmac declares common paper formats: a4, a4l (landscape a4), a5, a5l, b5, and letter; users can declare their own format using \def:

\def{pgs:b5l}{(250,176)mm}
\def{pgs:letterl}{(11,8.5)in}

The \margins command declares the margins of the document. This command has the following parameters:

\margins/{\langle pg \rangle \langle fmt \rangle} {\langle left \rangle \langle right \rangle \langle top \rangle \langle bot \rangle \langle unit \rangle}

For example:

\margins/1 a4 (2.5,2.5,2,2)cm

These parameters are:

- \langle pg \rangle: 1 or 2 specifies single-page or double-page (spread) design.
- \langle fmt \rangle: paper format (a4, a4l, etc.).
- \langle left \rangle, \langle right \rangle, \langle top \rangle, \langle bot \rangle: specifies the left, right, top and bottom margins.
- \langle unit \rangle: unit used for the \langle left \rangle, \langle right \rangle, \langle top \rangle, \langle bot \rangle values.

Any of the parameters \langle left \rangle, \langle right \rangle, \langle top \rangle, \langle bot \rangle can be empty. If both \langle left \rangle and \langle right \rangle are nonempty then \hsize is set. Else \hsize is unchanged. If both \langle left \rangle and \langle right \rangle are empty then typesetting area is centered in the paper format. The analogous case holds when \langle top \rangle or \langle bot \rangle parameter is empty (for \vsize instead of \hsize).

Examples:

% \hsize, \vsize untouched,
% typesetting area centered:
\margins/1 a4 (,,,)cm
% right margin set to 2cm
% \hsize, \vsize untouched,
% vertically centered:
\margins/1 a4 (,,2,,)cm

If \langle pg \rangle=1 then all pages have the same margins. If \langle pg \rangle=2 then the declared margins are used for odd pages, and the margins of even pages are mirrored, i.e. \langle left \rangle is replaced by \langle right \rangle and vice versa.

The command \magscale{\langle factor \rangle} scales the whole typesetting area. The fixed point of such scaling is the so-called “Knuthian origin”: 1in below and 1in right of paper sides. Typesetting (breakpoints etc.) is unchanged. Almost all units are relative after such scaling; only paper format dimensions remain unscaled. Example:

\margins/2 a5 (22,17,19,21)mm
\magscale[1414] \margins/1 a4 (,,,)mm

The first line sets the \hsize and \vsize and margins for final printing at a5 format. The setting on the second line centers the scaled typesetting area to the true a4 paper while breakpoints for paragraphs and pages are unchanged. It may be useful for a proof copy printed at a larger size. After the review is done, the second line can be commented out.

Petr Olšák
Czech Technical University
in Prague,
Czech Republic

OPmac: Macros for plain TeX
The Treasure Chest

This is a list of selected new packages posted to CTAN (http://ctan.org) from November 2012 through March 2013, 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 \TeX{} community. Comments are welcome, as always.

⋄ Karl Berry
tugboat (at) tug dot org
http://tug.org/ctan.html

fonts

\texttt{aecc} in fonts
Almost European Concrete Roman fonts.

\texttt{cabin} in fonts
A humanist sans with true italics and small capitals.

\texttt{garamondx} in fonts
Adds small caps, \texttt{f}-ligatures and old style figures to \URW{} GaramondNo8.

\texttt{*librebaskerville} in fonts
\LaTeX{} support for the Libre Baskerville font family. Article in this \textit{TUGboat}.

\texttt{persian-hm-ftx} in fonts/persian
228 Persian TrueType fonts derived from Farsi\LaTeX{} \METAFONT{}s.

\texttt{quattrocentro} in fonts
A classical typeface with both serif and sans designs.

\texttt{sourcecodepro} in fonts
Adobe’s open-source monospaced font
Source Code Pro in Type 1 and OpenType.

\texttt{sourcesanspro} in fonts
Adobe’s open-source font family Source Sans Pro
in Type 1 and OpenType.

\texttt{schulschriften} in fonts
Handwriting fonts used in German schools,
implemented in \METAFONT{}.

graphics

\texttt{flowchart} in graphics/pgf/contrib
Traditional programming flowcharts.

\texttt{forest} in graphics/pgf/contrib
Linguistic and other trees, with many novel features.

\texttt{logicpuzzle} in graphics/pgf/contrib
Battleship and Bokkusu games, with more to come.

\texttt{makeshape} in graphics/pgf/contrib
Simplifies creation of custom PGF shapes.

\texttt{pdftricks2} in graphics
Automation of \texttt{PSTricks} for pdf\LaTeX{}.

\texttt{pst-fit} in graphics/pstricks/contrib
\texttt{PSTricks} curve fitting.

\texttt{pst-vectorian} in graphics/pstricks/contrib
Drawing ornaments.

\texttt{sa-tikz} in graphics/pgf/contrib
\TikZ{} library for drawing switching architectures.

\texttt{tikzinclude} in graphics/pgf/contrib
Import one image from a file holding multiple images.

\texttt{tikzscale} in graphics/pgf/contrib
Scaling of \TikZ{} and \pgfplots{} graphics without
changing text size.

\texttt{tikzsymbols} in graphics/pgf/contrib
Emoticons, cooking symbols, and trees.

info

\texttt{luainfo} in info/examples
Examples from the German book \textit{Einführung in Lua\LaTeX{} und LuaL\LaTeX{}}.

macros/generic

\texttt{comando} in macros/generic
Expandable iteration over comma-separated and
file name lists.

\texttt{schemata} in macros/generic
Topical diagrams for, e.g., Scholastic thought.

macros/latex/contrib

\texttt{abntex2} in macros/latex/contrib
Brazilian academic theses based on ABNT rules.

\texttt{apptools} in macros/latex/contrib
Customize appendices.

\texttt{autopdf} in macros/latex/contrib
Facilitates on-the-fly conversion to PDF and other
formats supported by pdf\LaTeX{}.

\texttt{backmaur} in macros/latex/contrib
Backus-Naur Form definitions.

\texttt{bropd} in macros/latex/contrib
Writing differential operators and brackets.

\texttt{concepts} in macros/latex/contrib
Managing document-specific formal concepts.

\texttt{contracard} in macros/latex/contrib
Calling cards for contra and square dances.

\texttt{copypaste} in macros/latex/contrib
Dynamically quoting an external document.

\texttt{dvgloss} in macros/latex/contrib
Setting interlinear glossed text.

fonts/aecc
etoc in macros/latex/contrib
   Easily-customizable tables of contents.
listofanswers in macros/latex/contrib
   Similar to list of tables, etc. With Spanish support.
*minift in macros/latex/contrib
   Arithmetic to 7–8 decimal places, and a stack-based programming environment.
mkstmpdad in macros/latex/contrib
   Create custom stamps and use them for drag and drop matching.
multiexpand in macros/latex/contrib
   Macros meant to avoid too many \expandafters.
ocg-p in macros/latex/contrib
   Use PDF’s Optional Content Groups (layers) with pdfEPiTpX and XGEPiTpX.
scalerel in macros/latex/contrib
   Constrained scaling of objects, relative to a reference or absolutely.
tableof in macros/latex/contrib
   Tables of tagged contents.
threadcol in macros/latex/contrib
   Organize document columns into PDF article threads.
uestcthesis in macros/latex/contrib
   Thesis template for the University of Electronic Science and Technology of China.
underoverlap in macros/latex/contrib
   Partly-overlapping math decorations.
xpicture in macros/latex/contrib
   Extending pict2e and curve2e with support for arbitrary reference systems, function graphs, and more. Article in this TUGboat.

macros/latex/contrib/biblatex-contrib

uni-wtal-lin in m/l/c/biblatex-contrib
   BibLaTeX citation style for linguistic studies at the Bergische Universität Wuppertal.

macros/luatex

spelling in macros/luatex/latex
   Aid spell-checking of LuaLaTeX documents, with nearly any checker.

macros/xetex

ptex in macros/xetex/latex
   Similar to \lipsum for Persian: 100 paragraphs of the Shahnameh.
xetex-tibetan in macros/xetex/generic
   TECKit mappings for Unicode Tibetan.

support

arara in support
   \TeX build tool based on metadata in the sources.
dtxgen in support
   Create template for self-extracting .dtx file.

Production notes

Karl Berry

It’s been about ten years since the last production notes (written by Mimi Burbank; we miss you, Mimi), so it seemed time for an update.

For years now, TUGboat production has been via PDF files. We create separate PDFs for each of the four cover pages (front cover, inside front cover, inside back cover, back cover), and all the interior pages. We upload the PDFs to Cadmus, our production printer, via ftp. (TUGboat has used Cadmus (cadmus.com) for some 25 years, and they are still a pleasure to work with.) Cadmus quickly returns proofs (“bluelines”) to us on paper. Although they do have an electronic proof process, paper has been more reliable for us so far.

As always with paper printing, everything costs something, and some things cost more. Naturally, color costs more to print than regular black and white; so we often grayscale images (or have Cadmus grayscale them) when the color is not semantically important.

Another factor, especially for color, is the arrangement of the issue into signatures. Most readers here are likely familiar with this, but just in case: the general idea is that the issue is printed on giant sheets of paper — the size of 32 8.5x11 pages — and then other machines do the necessary cutting, folding and binding. Therefore, it’s ideal to have an issue which is a multiple of 32 pages. 16 is next best, then 8, then 4 (the process requires at least 4 pages in the last signature, and the powers of 2). We expend quite a bit of effort on the final pagination and combination of items to get the best page count possible.

As far as color goes, it’s critical to keep all color within one signature, since the color vs. b&w print costs are incurred on a per-signature basis. So sometimes we end up sacrificing the ideal article ordering to keep color articles together.

In another installment I’ll write some technical details about the production and tools we use. For now, let me switch gears . . .

Introduction to Colophon

The last piece in this issue is the first fiction ever published in TUGboat: a (very) short story by Daniel Quinn. Its subject was apropos for us. Quinn is better-known as the author of Ishmael, The Story of B, Beyond Civilization, and other books that share common themes of a search for truth and living in our world.

His books were transformative for me personally in understanding how and why we live as we do, so when I came across this short-short, I was very glad to bring it to TUGboat and perhaps a few new readers of Quinn’s work. Thanks to our editor and colleague Barbara Beeton for happily acquiescing in printing it, and of course to Daniel Quinn for graciously allowing us to reprint it.

(Colophon to “Colophon”: Lacking Saracen, I chose cmfib8 for the main font, the only time it’s seemed suitable for an article body. Thanks to Don Knuth, too, as always.)
Book review: *The Computer Science of \TeX\ and \LaTeX*  

Boris Veytsman


Victor Eijkhout wrote one of the best \TeX\ reference books — \TeX\ by Topic (http://www.eijkhout.net/tbt). Therefore the publication of his book about the computer science of \TeX\ is an interesting and important event for the community.

The book is a collection of lecture notes for the computer science course the author taught in 2004. They still have the unmistakable look and feel of notes we may have read in our student years: obligatory blurbs about the textbooks used, including their library call numbers; “to do” footnotes, addressed to the professor himself rather than to the students; an unfinished chapter, and even missed sections (nevertheless mentioned in the table of contents).

The idea of teaching computer science based on a large program written by one of the founders of the field is very interesting. It is controversial too: unlike toy programs written in order to illustrate the theory, \TeX\ is a “real life” program, with its real life compromises and errors. It is also a large and intricate program, and thus difficult to separate into parts, each neatly illustrating this or that computer science topic. On the other hand, we do teach students of biology and medicine by dissecting “real” organisms and explaining in this process both the general laws and specifics of this individual. However, the study of complex organisms usually takes place in quite advanced courses; for introductory ones a teacher usually chooses examples with simpler anatomy, like ringed worms.

Eijkhout avoids the complexity of \TeX\ the program by the following device: he never quotes the actual Pascal (or Web) code. Instead he talks about the algorithms and design decisions of \TeX\. Which is, in my opinion, a good decision for an introductory course, but one that makes the rather ambitious title of the book slightly misleading. A more fitting name would be “Introduction to Computer Science with Examples from \TeX\ Algorithms”.

The first chapter of the book contains a concise introduction to \TeX\ and \LaTeX. Most of this material will be familiar within the \TeX\ community, but Eijkhout’s characteristic style makes the chapter good reading. The second chapter, on parsing, introduces generative grammars and automata. While the author mentions the role of parsing in the tokenization process of \TeX, most of the material is illustrated by \lex\ and \yacc — in my opinion, a good decision. \TeX\ becomes prominent in the next chapter, where line-breaking and page-breaking algorithms are discussed. This allows the author to introduce complexity, \NP-complete problems and other more or less standard notions of an introductory computer science course. This is probably one of the best chapters of the course since it blends computer science and \TeX\ in the most natural way. The fourth chapter discusses fonts, curves, rasterization and other geometric-related algorithms used by \METAFONT\ and other font-drawing software. The fifth chapter should discuss \TeX\ macro language — unfortunately, it is unfinished. The author returns to fonts in the sixth chapter, where he introduces encodings, Unicode and the way \LaTeX\ deals with encoding issues. The last chapter is devoted to software engineering as a human activity; the lectures included literate programming (the notes for these are absent), teamwork and related concepts.

Thus, the book is rather uneven in its style. Some chapters, like Chapter 2, are very detailed and well written. Some, like Chapter 5, are unfinished or, like Chapter 7, just sketchy.

I think the best audience for this book is teachers of computer science and related courses. They might find many interesting ideas for explaining complex concepts. This book might be also interesting for the people who know the basics of computer science and want to look at the field from a different angle. It is more difficult for me to imagine a novice using this book for an independent study: it is what it is, lecture notes rather than a full-blown textbook. As long as one does not approach it as a textbook, it makes for very interesting reading.

◊ Boris Veytsman  
School of Systems Biology & Computational Materials Science Center, MS 6A2  
George Mason University  
Fairfax, VA 22030 USA  
borisv (at) lk dot net  
http://borisv.lk.net

Boris Veytsman
Die TEXnische Komödie 4/2012–1/2013

Die TEXnische Komödie is the journal of DANTE e.V., the German-language TeX user group (http://www.dante.de). [Non-technical items are omitted.]

Die TEXnische Komödie 4/2012

WALTER ENTENMANN, Schulschriften — von Sütterlin bis heute [School handwriting fonts — From Sutterlin until today]; pp. 29–57

This article describes the implementation of several handwriting fonts in METAFONT that have been or are used in German schools. These are Sütterlin (1911), the “German Normalschrift” (1941), the “Lateinisches Ausgangsschrift” (1953), the “Schulausgangsschrift” (1968) and the “Vereinfachte Schulausgangsschrift” (1972). The METAFONT sources are bundles in the schulschriften package, which also contains examples and documentation. After loading the package any German text can be typeset in one of the above-mentioned fonts.

The documentation describes, after a short historical review, the implementation step-by-step, from the handwritten prototype to the systematic translation into METAFONT and the organisation of the file structure. Useful parameter and macro definitions aid the design and maintenance of the package.

The connections of the single letters in a writing font are systematically specified by junction points and the introduction of connection levels. Due to the different typographic characteristics of each font there is an individual match design concept for each font. The description of the package also explains the connection between the dimensions of the handwritten prototype and the desired font sizes with the internally used “sharp” variables in METAFONT and the pixel sizes.

The font definition files, the style files for the related system of rules as well as character tables and font examples support and show the practical use of the fonts.

MARKUS KOHM, Briefpapier mit KOMA-Script nachbauen [Creating letter paper with KOMA-Script]; pp. 58–73

Since my article on modern letters with KOMA-Script a long time has passed. Since then there have been a few improvements and extensions. Questions on the adjustment of scrlltr2 for a specific layout are still very frequent. This article shows how to create a layout for Washington State University.

CHRISTINE RÖMER, Das Erstellen eines Glossars [Creating a glossary]; pp. 74–79

To create a glossary LATEX natively offers only a little help. Extensions in document classes, additional tools and packages, however, offer enough functionality to create one. They differ with respect to variability and complexity. The more variability they offer, the more difficult is their use.

HEINER RICHTER, Klausur »Steuerlehre« — LATEX-Einsatz für Sehbehinderte [Exam »Steuerlehre« — Using LATEX for visually handicapped people]; pp. 80–82

An article on the selection of font and fontsize for visually handicapped people.

HERBERT VÖSS, Spezielle Schriften [Special fonts]; pp. 83–90

It has frequently been mentioned in this publication that LuaTEX and XETEX can use OpenType fonts. This article introduces an uncommon font and shows the simple definition of ligatures for an OpenType font.

Die TEXnische Komödie 1/2013

HERBERT VÖSS, Die Schrift Venturis [The Venturis fonts]; pp. 7–15

Several fonts that feature a set of mathematical characters have been introduced in DTK or TUGboat. Compared to the overall number of available fonts the set of fonts usable for typesetting math is quite low, so a newcomer like “Venturis” is to be welcomed.

HEINER RICHTER, Interaktives Steuerlehreprojekt mit dem LATEX-Paket gamebook.sty [Interactive tax education using gamebook.sty]; pp. 16–20

A. Miede’s existing package gamebook.sty (http://mirror.ctan.org/macros/latex/contrib/gamebook), which notably provides the means to lay out gamebooks with LATEX, may be used for individual question-and-answer games in structured e-teaching.

UWE ZIEGENHAGEN, Rollup-Displays mit LATEX erstellen [Creating Rollup-Displays with LATEX]; pp. 21–24

If one visits a trade fair or a conference, one notices different rollup-displays, showing the products and projects of the exhibitors. Since DANTE e.V. hasn’t had one so far, I created a rollup-display for the Froscon 2012 in Sankt Augustin. In this article I briefly explain how the display was made with LATEX.
LATEX is commonly known as a typesetting system for the mathematical/scientific community. In this article I show how it can profitably be used for people who need neither formulas, technical drawings nor many pictures and tables. Among these are, for example, parish priests.

Gerd Neugebauer, CTAN: Relaunch des Web-Auftritts [CTAN: Relaunch of the web portal]; pp. 43–55

[Received from Herbert Vöß.]

Eutypon 28–29, October 2012

Eutypon is the journal of the Greek TeX Friends (http://www.eutypon.gr).

Nick White, Training Tesseract for Ancient Greek OCR; pp. 1–11

This paper discusses the process of training the Tesseract OCR engine to support Ancient Greek. It covers the general procedures involved in training a new language for Tesseract, both training the script with common printed fonts and adding hints about how the language works to improve recognition. It discusses the particular challenges that arose with Ancient Greek, mainly due to Tesseract’s English language heritage. It goes on to describe the various strategies and small programs which were written to overcome these. It concludes with recommendations for changes to Tesseract to make OCR training easier and further improve recognition accuracy. (Article in English.)

Apostolos Syropoulos, Creating ePUBLications; pp. 13–20

Can we create e-books with LATEX? The answer is yes, when we talk about e-books in ePUB format. First we present the ePUB format, and then describe how we can convert LATEX files to ePUB files using the latex2epub converter. We also explain how we can read ePUB files with the Firefox browser. (Article in Greek with English abstract.)

Apostolos Syropoulos, Creating ePUBLications for Ancient Greek; pp. 21–23

Petros Papasarantopoulos has dealt with Greek books since 1977. He has edited hundreds of books, in the past under the imprint “Paratiritis” and now under the imprint “Epikentro”. He has also worked for three decades as a journalist in Greek newspapers and magazines. As someone who knows the Greek book world very well, he accepted an invitation to talk to Eutypon about e-publications in Greece. (Article in Greek with English abstract.)

Apostolos Syropoulos, TEXniques: The problem with the Greek semicolon; pp. 41–42

How can we distinguish a middle dot from a Greek ano teleia (Greek semicolon)? (Article in Greek.)

Book presentations; pp. 43–46


[Received from Dimitrios Filippou and Apostolos Syropoulos.]
Calendar

2013

May 1  TUG election: nominations due. tug.org/election
May 19-24  12th Annual Book History Workshop, Texas A&M University, College Station, Texas. cushing.library.tamu.edu/events/book-history-workshop
Jun 5-12  The 5th International Conference on Typography and Visual Communication (ICTVC), “Against lethe...”, University of Nicosia, Cyprus. www.ictvc.org
Jun 10-Aug 2  Rare Book School, University of Virginia, Charlottesville, Virginia. Many one-week courses on type, bookmaking, printing, and related topics. www.rarebookschool.org/schedule
Jun 24-27  Book history workshop, École de l’institut d’histoire du livre, Lyon, France. ihl.enssib.fr
Jul 8  TUGboat 34:2, submission deadline (regular issue)
Jul 16-19  Digital Humanities 2013, Alliance of Digital Humanities Organizations, University of Nebraska-Lincoln. dh2013.unl.edu
Aug 5-9  Balisage: The Markup Conference, Montréal, Canada. www.balisage.net/
Sep 9  TUG 2013 preprints deadline. tug.org/tug2013
Sep 23-28  7th International ConTeXt Meeting, Brejlov, Czech Republic. meeting.contextgarden.net/2013/

TUG 2013
Tokyo, Japan.
Oct 23-26  The 34th annual meeting of the TEx Users Group. Presentations covering the TEx world. tug.org/tug2013
Nov 2  DANTE Herbsttagung and 40th meeting, Köln, Germany www.dante.de/events.html

Status as of 15 March 2013

For additional information on TUG-sponsored events listed here, contact the TUG office (+1 503 223-9994, fax: +1 815 301-3568, e-mail: office@tug.org). 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.
Other calendars of typographic interest are linked from tug.org/calendar.html.
The 34th Annual Meeting of the \TeX\ Users Group

October 23–26, 2013

Graduate School of Mathematical Sciences, the University of Tokyo
3-8-1 Komaba, Meguro-ku
Tokyo, Japan

http://tug.org/tug2013 • tug2013@tug.org

July 15, 2013 — bursary application deadline
July 15, 2013 — and presentation proposal deadline
July 15, 2013 — and early bird registration deadline
Sept. 9, 2013 — preprint submission deadline
Oct. 22–26, 2013 — conference
Nov. 4, 2013 — deadline for final papers

Sponsored by:
Graduate School of Mathematical Sciences, the University of Tokyo
SANBI Printing
the \TeX\ Users Group, and DANTE e.V.
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, Merceré
Tarragona 102 4ª 2º
08015 Barcelona, Spain
+34 932267827
Email: m.aicart (at) ono.com
Web: http://www.edilatex.com
We provide, at reasonable low cost, LATEX or TEX page layout and typesetting services to authors or publishers world-wide. We have been in business since the beginning of 1990. For more information visit our web site.

Dangerous Curve
PO Box 532281
Los Angeles, CA 90053
+1 213-617-8483
Email: typesetting (at) dangerouscurve.org
Web: http://dangerouscurve.org/tex.html
We are your macro specialists for TEX or LATEX fine typography specs beyond those of the average LATEX macro package. If you use XeTEX, we are your microtypography specialists. We take special care to typeset mathematics well.

Not that picky? We also handle most of your typical TEX and LATEX 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 TEX book.

Latchman, David
4113 Planz Road Apt. C
Bakersfield, CA 93309-5935
+1 518-951-8786
Email: david.latchman (at) texnical-designs.com
Web: http://www.texnical-designs.com
Specializing in: The typesetting of books, monographs, journals and papers to allow your documents and books to look their best; and the creation of style or class files to meet your organization’s book, journal, slide or report formatting needs.

Moody, Trent
1981 Montecito Ave.
Mountain View, CA 94043
+1 650-283-7042
Email: trent.moody (at) ymail.com
Construction of technical documents with mathematical content from handwritten (or partially formatted) sources. Delivered documents will be .tex and .pdf files produced with TEX or/and LEtEX. Delivered documents can be publication ready manuscripts, macro libraries for modular document development, or mathematical libraries for document reuse.
I am an independent contractor with a PhD in mathematical physics from the University of California, Santa Cruz.

Peter, Steve
295 N Bridge St.
Somerville, NJ 08876
+1 732 306-6309
Email: speter (at) mac.com
Specializing in foreign language, multilingual, linguistic, and technical typesetting using most flavors of TEX. I have typeset books for Pragmatic Programmers, Oxford University Press, Routledge, and Kluwer, among others, and have helped numerous authors turn rough manuscripts, some with dozens of languages, into beautiful camera-ready copy. In addition, I’ve helped publishers write, maintain, and streamline TEX-based publishing systems. 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
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
Shanmugan, R. (cont’d)
various projects. I am a software consultant with Master’s Degree. I have sound knowledge in \TeX, \LaTeX, XML, MathML, DTD, XSLT, XSL-FO, Schema, ebooks, OeB, etc.

Sharma, Ganesh Kumar
A - 251 / 1, Opposite Primary School, Shastrinagar, Delhi 110052, India
+91 9810748682, 9013121611
Email: ganeshasharma (at) yahoo.com
I am a Master of Computer Applications (MCA) degree holder. I am well versed with MetaPost, HTML, MathML, Java, CSS, PHP, Unix shell scripting, C++, Ti\kZ, Gnuplot and PostScript etc.

As a consultant and service provider, I am handling \LaTeX and \XeLaTeX composition to technical illustration, editorial services for: project management of conference proceedings; class/style files creation for \LaTeX publications; a full management service for journals including correspondence with authors and issue make-up, including manuscript Preparation (pagination / composition, copy editing and proof reading), scanning and graphics designing, origination from handwritten manuscript or use of author-supplied code (\TeX or word processor), and author support; the supply of HTML, PDF files (including hyperlinks and bookmarks) and other coding for electronic publication. I can typeset the books in Sanskrit and Hindi languages using \LaTeX very well.

Currently, I am giving editorial services to many universities, reputed publishers and multinational companies, research groups etc.

Sievers, Martin
Klaus-Kordel Straße 8, 54296 Trier, Germany
+49 651 4936567-0
Email: info (at) schoenerpublizieren.com
Web: http://www.schoenerpublizieren.com
As a mathematician with ten years of typesetting experience I offer \TeX and \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 (\BibTeX, \biblatex) to typesetting your math, tables or graphics — just contact me with information on your project.

Sofka, Michael
8 Providence St.
Albany, NY 12203
+1 518 331-3457
Email: michael.sofka (at) gmail.com
Skilled, personalized \TeX and \LaTeX consulting and programming services.

I offer over 25 years of experience in programming, macro writing, and typesetting books, articles, newsletters, and theses in \TeX and \LaTeX: Automated document conversion; Programming in Perl, C, C++ and other languages; Writing and customizing macro packages in \TeX or \LaTeX; Generating custom output in PDF, HTML and XML; Data format conversion; Databases.

If you have a specialized \TeX or \LaTeX need, or if you are looking for the solution to your typographic problems, contact me. I will be happy to discuss your project.

Veytsman, Boris
46871 Antioch Pl.
Sterling, VA 20164
+1 703 915-2406
Email: borisv (at) lk.net
Web: http://www.borisv.lk.net
\TeX and \LaTeX consulting, training and seminars. Integration with databases, automated document preparation, custom \LaTeX packages, conversions and much more. I have about seventeen years of experience in \TeX and thirty years of experience in teaching & training. I have authored several packages on CTAN, published papers in \TeX related journals, and conducted several workshops on \TeX and related subjects.

Young, Lee A.
127 Kingfisher Lane
Mills River, NC 28759
+1 828 435-0525
Email: leeayoung (at) morrisbb.net
Web: http://www.latexcopyeditor.net
http://www.editingscience.net
Copyediting your .tex manuscript for readability and mathematical style by a Harvard Ph.D. Your .tex file won’t compile? Send it to me for repair. Experience: edited hundreds of ESL journal articles, economics and physics textbooks, scholarly monographs, \LaTeX manuscripts for the Physical Review; career as professional, published physicist.
TUG membership rates are listed below. Please check the appropriate boxes and mail the completed form with payment (in US dollars) to the mailing address at left. If paying by credit/debit card, you may alternatively fax the form to the number at left or join online at http://tug.org/join.html. The web page also provides more information than we have room for here.

**Status** (check one)  □ New member   □ Renewing member

**Automatic membership renewal** in future years  □

Using the given payment information; contact office to change or cancel.

<table>
<thead>
<tr>
<th>Rate</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>$85</td>
<td>□</td>
</tr>
<tr>
<td>$55</td>
<td>□</td>
</tr>
<tr>
<td>$100</td>
<td>□</td>
</tr>
<tr>
<td>$200</td>
<td>□</td>
</tr>
<tr>
<td>$500</td>
<td>□</td>
</tr>
<tr>
<td>$30</td>
<td>□</td>
</tr>
<tr>
<td>$55</td>
<td>□</td>
</tr>
<tr>
<td>$10</td>
<td>□</td>
</tr>
<tr>
<td>$20</td>
<td>□</td>
</tr>
<tr>
<td>$20</td>
<td>□</td>
</tr>
<tr>
<td>$10</td>
<td>□</td>
</tr>
<tr>
<td>$10</td>
<td>□</td>
</tr>
<tr>
<td>$10</td>
<td>□</td>
</tr>
<tr>
<td>$10</td>
<td>□</td>
</tr>
<tr>
<td>$10</td>
<td>□</td>
</tr>
<tr>
<td>$10</td>
<td>□</td>
</tr>
<tr>
<td>$10</td>
<td>□</td>
</tr>
</tbody>
</table>

**Voluntary donations** (more info at https://www.tug.org/donate.html)

<table>
<thead>
<tr>
<th>Contribution</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>General TUG contribution</td>
<td></td>
</tr>
<tr>
<td>Bursary Fund contribution</td>
<td></td>
</tr>
<tr>
<td>TpX Development Fund contribution</td>
<td></td>
</tr>
<tr>
<td>CTAN contribution</td>
<td></td>
</tr>
<tr>
<td>TpX Gyre fonts contribution</td>
<td></td>
</tr>
<tr>
<td>ETPX 3 contribution</td>
<td></td>
</tr>
<tr>
<td>LuaTeX contribution</td>
<td></td>
</tr>
<tr>
<td>MacTeX contribution</td>
<td></td>
</tr>
</tbody>
</table>

**Total $**

Tax deduction: The membership fee less $40 is generally deductible, at least in the US.

**Multi-year orders:** To join for more than one year at this year’s rate, please multiply.

**Payment** (check one)  □ Payment enclosed   □ Visa   □ MasterCard   □ AmEx

Account Number: _______________________________  Exp. date: ______

Signature: _________________________________

Privacy: TUG uses your personal information only to send products, publications, notices, and (for voting members) official ballots. TUG does not sell or otherwise provide its membership list to anyone.

Name _________________________________

Department _______________________________

Institution _______________________________

Address __________________________________

City ______________________________________  State/Province __________________

Postal code ______________________________  Country __________________

Email address ______________________________

Phone _______________________________  Fax __________________

Position _______________________________  Affiliation ____________________________
TUG financial statements for 2012

Karl Berry, TUG treasurer

The financial statements for 2012 have been reviewed by the TUG board but have not been audited. As a US tax-exempt organization, TUG’s annual information returns are publicly available on our web site: http://tug.org/tax-exempt.

Revenue (income) highlights

Membership dues revenue was down about 2% in 2012 compared to 2011, while product sales revenue was substantially up. Contributions, interest, and advertising income were all slightly down. Overall, 2012 income was up 1%.

Cost of Goods Sold and Expenses highlights, and the bottom line

Payroll, office expenses, and TUGboat and DVD production and mailing continue to be the major expense items. All were nearly as budgeted; overall, 2012 expenses were up about 3% from 2011.

Often we have a “prior year adjustment” early in the year to compensate for an estimate in the previous year; in 2012 the total of adjustments was positive for the bottom line: $222.

The net result for the year was substantially positive: about $7,200.

Balance sheet highlights

TUG’s end-of-year asset total is up around $4,000 (2%) in 2012 compared to 2011.

The Committed Funds are administered by TUG specifically for designated projects: \texttt{E\TeX X3}, the \texttt{E\TeX } development fund, CTAN, and so forth. Incoming donations have been allocated accordingly and are disbursed as the projects progress. TUG charges no overhead for administering these funds.

The Prepaid Member Income category is member dues that were paid in earlier years for the current year (and beyond). Most of this liability (the 2013 portion) was converted into regular Membership Dues in January of 2013.

The payroll liabilities are for 2012 state and federal taxes due January 15, 2013.

Summary

TUG continues to work closely with the other \texttt{T\TeX } user groups and ad hoc committees on many activities to benefit the \texttt{T\TeX } community.

<table>
<thead>
<tr>
<th>TUG 12/31/2012 (vs. 2011) Revenue and Expense</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan - Dec 12</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>Ordinary Income/Expense</td>
</tr>
<tr>
<td>Income</td>
</tr>
<tr>
<td>Membership Dues</td>
</tr>
<tr>
<td>Product Sales</td>
</tr>
<tr>
<td>Contributions Income</td>
</tr>
<tr>
<td>Annual Conference</td>
</tr>
<tr>
<td>Interest Income</td>
</tr>
<tr>
<td>Advertising Income</td>
</tr>
<tr>
<td>Total Income</td>
</tr>
<tr>
<td>Cost of Goods Sold</td>
</tr>
<tr>
<td>TUGboat Prod/Mailing</td>
</tr>
<tr>
<td>Software Production/Mailing</td>
</tr>
<tr>
<td>Postage/Delivery - Members</td>
</tr>
<tr>
<td>Lucida Open Type Font Project</td>
</tr>
<tr>
<td>Lucida Sales Accrual B&amp;H</td>
</tr>
<tr>
<td>Member Renewal</td>
</tr>
<tr>
<td>Total COGS</td>
</tr>
<tr>
<td>Gross Profit</td>
</tr>
<tr>
<td>Expense</td>
</tr>
<tr>
<td>Contributions made by TUG</td>
</tr>
<tr>
<td>Office Overhead</td>
</tr>
<tr>
<td>Payroll Exp</td>
</tr>
<tr>
<td>Lucida OpenType Development</td>
</tr>
<tr>
<td>Total Expense</td>
</tr>
<tr>
<td>Net Ordinary Income</td>
</tr>
<tr>
<td>Other Income</td>
</tr>
<tr>
<td>Prior year adjust</td>
</tr>
<tr>
<td>Total Other Income</td>
</tr>
<tr>
<td>Net Income</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TUG 12/31/2012 (vs. 2011) Balance Sheet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dec 31, 12</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>ASSETS</td>
</tr>
<tr>
<td>Current Assets</td>
</tr>
<tr>
<td>Total Checking/Savings</td>
</tr>
<tr>
<td>Accounts Receivable</td>
</tr>
<tr>
<td>Total Current Assets</td>
</tr>
<tr>
<td>TOTAL ASSETS</td>
</tr>
<tr>
<td>LIABILITIES &amp; EQUITY</td>
</tr>
<tr>
<td>Liabilities</td>
</tr>
<tr>
<td>Committed Funds</td>
</tr>
<tr>
<td>TUG conference</td>
</tr>
<tr>
<td>Prepaid member income</td>
</tr>
<tr>
<td>Payroll Liabilities</td>
</tr>
<tr>
<td>Total Current Liabilities</td>
</tr>
<tr>
<td>TOTAL LIABILITIES</td>
</tr>
<tr>
<td>Equity</td>
</tr>
<tr>
<td>Unrestricted</td>
</tr>
<tr>
<td>Net Income</td>
</tr>
<tr>
<td>Total Equity</td>
</tr>
<tr>
<td>TOTAL LIABILITIES &amp; EQUITY</td>
</tr>
</tbody>
</table>
2013 \TeX{} Users Group election

Barbara Beeton
for the Elections Committee

The positions of TUG President and six members of the Board of Directors will be open as of the 2013 Annual Meeting, which will be held in October 2013 in Japan.

The directors whose terms will expire in 2013: Kaja Christiansen, Jonathan Fine, Steve Grathwohl, Jim Hefferon, Klaus Höppner, and David Walden.

Continuing directors, with terms ending in 2015: Barbara Beeton, Karl Berry, Susan DeMeritt, Michael Doob, Taco Hoekwater, Ross Moore, Cheryl Ponchin, Philip Taylor, and Boris Veytsman.

The election to choose the new President and Board members will be held in Spring of 2013. 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 2013 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 May 2013. Forms may be submitted by FAX, or scanned and submitted by e-mail to 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 \textit{TUGboat} as well as through various \TeX{}-related electronic lists.

2013 TUG Election — Nomination Form

Only TUG members whose dues have been paid for 2013 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. \textbf{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:

\textbf{Name of Nominee:} \\
\textbf{Signature:} \\
\textbf{Date:} \\
for the position of (check one):

\begin{itemize}
  \item \textbf{□ TUG President}
  \item \textbf{□ Member of the TUG Board of Directors}
\end{itemize}

\textbf{for a term beginning with the 2013 Annual Meeting, October 2013}

\begin{itemize}
  \item 1.
  \begin{itemize}
    \item (please print)
    \item (signature) (date)
  \end{itemize}
  \item 2.
  \begin{itemize}
    \item (please print)
    \item (signature) (date)
  \end{itemize}
\end{itemize}

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 May 2013.\footnote{Supplementary material may be sent separately from the form, and supporting signatures need not all appear on the same form.} It is the responsibility of the candidate to ensure that this deadline is met. Under no circumstances will incomplete applications be accepted.

\begin{itemize}
  \item \textbf{□} nomination form
  \item \textbf{□} photograph
  \item \textbf{□} biography/personal statement
\end{itemize}

\TeX{} Users Group  \textbf{FAX:} \quad +1 815 301-3568

\textbf{Nominations for 2013 Election}

P. O. Box 2311
Portland, OR 97208-2311
U.S.A.
Colophon
Daniel Quinn

An embellishment sometimes added on the last page (usually recto) of a specially designed and produced book ... including the facts of production.

The University of Chicago Manual of Style

This book was handset in Saracen, so-called by its creator, Giovanni Cappelini (1762–97), because he felt it embodied the “wicked subtlety” of that pre-Islamic desert people. Although classically balanced and charged with virile grace (rivaling the productions of such contemporaries as Bodoni and Didot), it is not a common choice for designers, who tend to think of it as “Saracen Truncated.”

Type designers are notoriously superstitious about the way they work. For many, every new font is born from a single letter, which imposes its personality on the entire family. For Cappelini, this virgin mother was always the lower-case g, which strictly supervised the upbringing of all the rest. Another letter, the third-to-last in the English alphabet, came to be in a similarly special category for Cappelini because his mother was English and he was the third-to-last of her children. Because he considered this character to be the “key” to the font (of course a punning reference to the Greek chi it resembles), it had to be “turned” (cut) last of all, so as finally to release the font to the waiting world. Fanciful as this may sound to modern ears, this was a perfectly serious matter to Cappelini—quite gravely serious, as matters turned out.

A few days before he was to set about “turning the key” on Saracen, he paid a visit to the foundry where the font would be cast for him. While waiting to speak to the founder, he came across a proof sheet of a new font created by one of his rivals, Antonio Ristavo, a minor talent but a facile imitator. Cappelini froze in horror as he saw, right in the middle of the sheet, a character that could be mistaken for nothing but the key character of his new font, Saracen. For a terrib le moment, he wondered if Ristavo might have hit on precisely the form of his character by some unthinkable miracle of coincidence. But the merest glance at the rest of the font showed this to be impossible. To a professional’s eye, Cappelini’s character stood out like a falcon in a flock of crows, and not even a miracle of coincidence could have put it there.

Cappelini raced across the city, burst into Ristavo’s workshop, thrust the proof sheet in his face, and demanded an accounting. Too startled to do anything else, Ristavo just laughed—an unfortunate misjudgment, for a moment later he lay dead at Cappelini’s feet, felled by a single furious blow.

Ristavo’s laugh had blown the mystery away like a cobweb. The appearance of Cappelini’s character in Ristavo’s new font was a typographer’s gibe, a coded message only the cognoscenti could decipher. In plain, it stated: “Cappelini isn’t the only man in this city with a key to Cappelini’s door!”

Minutes later he confronted his wife with the proof sheet. She gazed at it dumbly, without comprehension. He dragged her to his workshop to show her the drawings for the character. As by degrees she gradually awakened to what had happened, she turned pale. Although plainly innocent of all complicity in the theft itself, she nonetheless recognized that her infidelity had been laid bare beyond doubt or evasion.

“Ristavo has betrayed us both,” she told her husband calmly. “I trust you will at least kill him first.”

“He is already dead,” Cappelini informed her.

In that era, a type designer in his workshop didn’t have to reach far to lay his hand on a razor-sharp blade. After writing a brief confession, which included a final request, Cappelini used the same blade on himself.

The “key” character for Saracen was never cut, save by Ristavo, and it was Cappelini’s last request that this be thrown into the melting pot (along with the rest of Ristavo’s font) to provide metal for the first casting of Saracen—thus giving Cappelini the last word in every sense.

One graphic designer spoke for all when he said that, “by some incalculable magic, Saracen is capable of imparting to a page an air of ineffably delicate savagery.” This means (among other things) that it’s never going to become a default choice for run-of-the-mill books, like, say, Century Schoolbook or Times Roman. But the chief reason for its rarity is that, for the sake of a benefit that seems to them so trifling, not many authors are willing to revise their work to meet the constraints of a twenty-five character alphabet; indeed, more than one has suggested that the vacancy might easily be filled by an import from some other font, but of course a barbarism of this sort could never be countenanced.

Daniel Quinn
http://www.ishmael.org