

# TUGBOAT

Volume 32, Number 2 / 2011

<b>General Delivery</b>	131	From the president / <i>Karl Berry</i>
	131	Editorial comments / <i>Barbara Beeton</i> Boris = books; EuroBach $\TeX$ 2011; 150 years at the US Government Printing Office; The raster tragedy
<b>Software &amp; Tools</b>	132	$\TeX$ Collection 2011 DVD / <i><math>\TeX</math> Collection editors</i>
	133	$\TeX$ works — As you like it / <i>Stefan Löffler</i>
	136	MetaPost 1.750: Numerical engines / <i>Taco Hoekwater</i>
	139	Reading and executing source code / <i>Herbert Voß</i>
<b>Education</b>	145	Teaching $\LaTeX$ to the students of mathematics — the experience from The Jan Kochanowski University / <i>Krzysztof Pszczoła</i>
<b>Graphics</b>	146	Drawing tables: Graphic fun with Lua $\TeX$ / <i>Paul Isambert</i>
<b>Electronic Documents</b>	152	E-books: Old wine in new bottles / <i>Hans Hagen</i>
	158	i $\TeX$ — Document formatting in an ereader world / <i>William Cheswick</i>
<b>Fonts</b>	164	Math alphabets and the <code>mathalfa</code> package / <i>Michael Sharpe</i>
	169	Another incarnation of Lucida: Towards Lucida OpenType / <i>Ulrik Vieth and Mojca Miklavc</i>
	177	MFLua / <i>Luigi Scarso</i>
<b>Macros</b>	185	Macro interfaces and the <code>getoptk</code> package / <i>Michael Le Barbier Grünewald</i>
<b><math>\LaTeX</math></b>	193	The <code>calls</code> package: Multipage tables with decorations / <i>Oleg Parashchenko</i>
	202	Glistings: Ornaments / <i>Peter Wilson</i>
	206	Merciadri packages: An overview / <i>Luca Merciadri</i>
<b>Con<math>\TeX</math>t</b>	211	Con $\TeX$ t basics for users: Paper setup / <i>Aditya Mahajan</i>
<b>Bibliographies</b>	213	Experiences with notes, references, and bibliographies / <i>David Walden</i>
<b>Typography</b>	217	Sixty years of book design at St. Gallen, Switzerland / <i>Paul Shaw</i>
<b>TUG Business</b>	224	TUG institutional members
<b>Book Reviews</b>	225	Book review: <i>A Specimen Portfolio of Wood Type in the Cary Collection</i> / <i>William Adams</i>
	226	Book review: <i>The Art of the Book in the Twentieth Century</i> / <i>Boris Veytsman</i>
	228	Book review: <i><math>\LaTeX</math> Beginner's Guide</i> / <i>Boris Veytsman</i>
	230	An appreciation: <i>The Art of Computer Programming, Volume 4A</i> / <i>David Walden</i>
<b>Hints &amp; Tricks</b>	233	The treasure chest / <i>Karl Berry</i>
<b>Abstracts</b>	235	<i>Les Cahiers GUTenberg</i> : Contents of issue 54–55 (2010)
	235	<i>Die <math>\TeX</math>nische Komödie</i> : Contents of issues 4/2010–1/2011
	236	<i>Zpravodaj</i> : Contents of issue 20(4) (2010)
<b>Advertisements</b>	237	$\TeX$ consulting and production services
<b>Letters</b>	238	Status of the American core CTAN node / <i>Jim Hefferon</i>
<b>News</b>	239	Calendar
	240	TUG 2011 announcement

## T<sub>E</sub>X Users Group

*TUGboat* (ISSN 0896-3207) is published by the T<sub>E</sub>X Users Group.

### Memberships and Subscriptions

2011 dues for individual members are as follows:

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

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

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

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

### Institutional Membership

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

T<sub>E</sub>X is a trademark of the American Mathematical Society.

Copyright © 2011 T<sub>E</sub>X Users Group.

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

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

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

## Board of Directors

Donald Knuth, *Grand Wizard of T<sub>E</sub>X-arcana*<sup>†</sup>  
Karl Berry, *President*<sup>\*</sup>  
Kaja Christiansen\*, *Vice President*  
David Walden\*, *Treasurer*  
Susan DeMeritt\*, *Secretary*  
Barbara Beeton  
Jon Breitenbucher  
Jonathan Fine  
Steve Grathwohl  
Jim Hefferon  
Klaus H $\ddot{o}$ ppner  
Ross Moore  
Steve Peter  
Cheryl Ponchin  
Philip Taylor  
Boris Veytsman  
Raymond Goucher, *Founding Executive Director*<sup>†</sup>  
Hermann Zapf, *Wizard of Fonts*<sup>†</sup>

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

<sup>†</sup>honorary

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

### Addresses

T<sub>E</sub>X Users Group  
P. O. Box 2311  
Portland, OR 97208-2311  
U.S.A.

### Telephone

+1 503 223-9994

### Fax

+1 206 203-3960

### Web

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

### Electronic Mail

(Internet)

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

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

Technical support for  
T<sub>E</sub>X users:  
[support@tug.org](mailto:support@tug.org)

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

### Have a suggestion? Problems not resolved?

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

[printing date: August 2011]

Printed in U.S.A.



## **TUGboat**

This regular issue (Vol. 32, No. 2) is the second issue of the 2011 volume year. No. 3 will contain papers from the TUG 2011 conference in Trivandrum, India.

*TUGboat* is distributed as a benefit of membership to all current TUG members. It is also available to non-members in printed form through the TUG store (<http://tug.org/store>), and online at the *TUGboat* web site, <http://tug.org/TUGboat>. Online publication to non-members is delayed up to one year after an issue's print publication, to give members the benefit of early access.

Submissions to *TUGboat* are reviewed by volunteers and checked by the Editor before publication. However, the authors are still assumed to be the experts. Questions regarding content or accuracy should therefore be directed to the authors, with an information copy to the Editor.

### **Submitting items for publication**

The deadline for receipt of final papers for the proceedings issue is October 31.

As always, suggestions and proposals for *TUGboat* articles are gratefully accepted and processed as received. Please submit contributions by electronic mail to [TUGboat@tug.org](mailto:TUGboat@tug.org).

The *TUGboat* style files, for use with plain  $\text{\TeX}$  and  $\text{\LaTeX}$ , are available from CTAN and the *TUGboat* web site. We also accept submissions using Con $\text{\TeX}$ t. More details and tips for authors are at <http://tug.org/TUGboat/location.html>.

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

### **Other TUG publications**

TUG is interested in considering additional manuscripts for publication, such as manuals, instructional materials, documentation, or works on any other topic that might be useful to the  $\text{\TeX}$  community in general.

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

### **TUGboat advertising**

For information about advertising rates and options, including consultant listings, write or call the TUG office, or see our web pages:

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

<http://tug.org/consultants.html>

### **Trademarks**

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

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

---

## From the President

Karl Berry

### Software

The 2011 releases of  $\text{T}_{\text{E}}\text{X}$  Live,  $\text{MacT}_{\text{E}}\text{X}$ , and  $\text{proT}_{\text{E}}\text{X}$ t are now available online. The  $\text{T}_{\text{E}}\text{X}$  Collection DVD is going to manufacturing around the same time as this *TUGboat* issue, and should be mailed during August. The <http://tug.org/texcollection> web page has links to all the pieces and general information.

### Conferences

The TUG 2011 conference will take place in Trivandrum, Kerala, India, October 19–21, hosted by River Valley Technologies: <http://tug.org/tug2011>. We will continue to accept presentation proposals while space is available. Online registration for all attendees is also open.

Other upcoming conferences: the fifth  $\text{ConT}_{\text{E}}\text{X}$ t user meeting in Charneux, Belgium, Sept. 19–24 (<http://meeting.contextgarden.net/2011>); and  $\text{T}_{\text{E}}\text{X}$ perience 2011 in Zelezná Ruda, Czech Republic, Sept. 28–Oct. 2 (<http://striz9.fame.utb.cz/texperience>).

### Interviews

Since my last column, Dave Walden has completed an interview with Richard Palais, the first president (then called chairman) of TUG, for the Interview Corner (<http://tug.org/interviews>).

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

---

## Editorial comments

Barbara Beeton

### Boris = books

Welcome, Boris Veytsman, to the *TUGboat* production team. Boris has volunteered to obtain book reviews, and, even more importantly, to arrange with publishers of books on typography and related subjects for TUG members to receive discounts on their publications. The first fruits of his efforts can be found in the reviews that appear later in this issue and in the listings of member discounts

in the members' area of the TUG web site: <http://tug.org/members>. Remember — your password is needed to access the members' area; if you've lost or forgotten it, request it from Robin at the TUG office: [memberaccess@tug.org](mailto:memberaccess@tug.org).

To further publicize this benefit, and as usual for non-technical items, book reviews are open immediately to all readers on the TUG web site; they will not be subject to the one-year quarantine to members only.

### EuroBach $\text{T}_{\text{E}}\text{X}$ 2011

The main theme of this year's Euro $\text{T}_{\text{E}}\text{X}$ , held in Bachotek, Poland, was "Aesthetics and effectiveness of the message, cultural contexts". Several papers from the Euro $\text{T}_{\text{E}}\text{X}$  proceedings have been republished in this issue of *TUGboat*; the complete proceedings will be published by GUST, the host of the conference.

### 150 years at the US Government Printing Office

This year, the Government Printing Office (GPO) celebrates its 150th anniversary. An exhibit showing highlights of this span, from hand-set type for the Emancipation Proclamation to the new e-world of hand-held digital devices, is on display in Washington, DC.

A brief introduction, with a five-minute video, is available at <http://paperspecs.com/mainblog/gpos-150th-birthday-exhibit>.

For those of us whose closest exposure to the GPO is the annual income tax forms, it's nice to know that their work includes many more welcome products. The exhibition, "Keeping America Informed", should be worth a visit if you're in the DC area.

### The raster tragedy

The web site <http://www.rastertragedy.com> presents a comprehensive overview of current knowledge about how to render outline fonts competently on low-resolution screens. (The term "raster tragedy" was coined by Peter Karow in the late 1980s, when the typical monitor screen rarely had a resolution of more than a pixel or two per point.)

Although this information (and the web site) has been around for quite a while, it has recently been updated, and a periodic review of the principles of good hinting is beneficial — it will keep one on one's toes in being able to determine whether a font is good or bad, and why, and how.

◇ Barbara Beeton  
<http://tug.org/TUGboat>  
 tugboat (at) tug dot org

## TeX Collection 2011 DVD

TeX Collection editors

The TeX Collection is the name for the overall collection of software distributed by the TeX user groups each year. Please consider joining TUG or the user group best for you (<http://tug.org/usergroups.html>), or making a donation (<https://www.tug.org/donate.html>), to support the effort.

All of these projects are done entirely by volunteers. If you'd like to help with development, testing, documentation, etc., please visit the project pages for more information on how to contribute.

Thanks to everyone involved, from all parts of the TeX world.

### 1 proTeXt (<http://tug.org/protext>)

proTeXt is a TeX system for Windows, based on MiKTeX (<http://www.miktex.org>), with a detailed document to guide your installation and additional Windows-specific tools.

For 2011, proTeXt now has a standard application program (`Setup.exe`) for installation, instead of using an interactive PDF document. Continual changes in Adobe Reader's security policies broke installation from the document, and free (libre) PDF readers didn't implement the necessary features.

Also, TeXnicCenter has been replaced as the standard editor by TeXStudio, formerly TeXMakerX (<http://texstudio.sf.net>).

proTeXt currently has English and German as possible installation languages. Volunteers to make translations to other languages are most welcome.

### 2 MacTeX (<http://tug.org/mactex>)

MacTeX is a TeX system for Mac OS X, based on TeX Live, with a native Mac installer and additional Mac-specific tools. MacTeX 2011 requires at least Mac OS X 10.5 (Leopard), and runs on Leopard, Snow Leopard (10.6), and Lion (10.7), both Intel and PowerPC machines. The package installs the full TeX Live 2011, Ghostscript 9.02, the `convert` utility from ImageMagick 6.6.9-3, and the current versions of BibDesk, L<sup>A</sup>T<sub>E</sub>X<sub>i</sub>T, TeX Live Utility, TeXShop, and TeXworks, as well as the TeX Dist Preference Pane, which allows users to switch easily between different TeX distributions.

Last year, users saw what appeared as two distributions in the Preference Pane, one with 32 bit universal binaries and one with 64 bit binaries. This year the panel shows only one 2011 entry, with a drop down menu to choose between 32 and 64 bit. The 64 bit binaries require Snow Leopard or higher (this year all binaries except `xetex` and `xdv2pdf` have 64

bit versions). Distributions from past years look as they did before in the Preference Pane.

The Collection also includes MacTeXtras (<http://tug.org/mactex/mactextras.html>), which contains many additional items that can be separately installed. This year, software that runs exclusively on Tiger (Mac OS X 10.4) has been removed. The main categories are: bibliography programs; alternative editors, typesetters, and previewers; equation editors; DVI and PDF previewers; and spell checkers.

### 3 TeX Live (<http://tug.org/texlive>)

TeX Live is a comprehensive cross-platform TeX system. It includes support for most Unix-like systems, including GNU/Linux and Mac OS X, and for Windows. Major user-visible changes in 2011 are few:

The `biber` (<http://ctan.org/pkg/biber>) program for bibliography processing is included on common platforms. Its development is coupled with `biblatex` (<http://ctan.org/pkg/biblatex>), a full reimplement of the bibliographical facilities provided by L<sup>A</sup>T<sub>E</sub>X.

The MetaPost (`mpost`) program no longer creates or uses `.mem` files. (The needed files, such as `plain.mp`, are simply read on every run.) This is related to supporting MetaPost as a library, which is another significant though not user-visible change.

The `updmap` implementation in Perl, previously used only on Windows, has been revamped and is now used on all platforms. No user-visible changes are intended, except that it runs much faster.

The biggest change is simply the package and program updates and additions that have accumulated over the past year: TeX Live 2011 is some 250 MB larger.

### 4 CTAN (<http://www.ctan.org>)

CTAN is the Comprehensive TeX Archive Network, a set of servers worldwide making TeX software publicly available.

The CTAN snapshot is about 3.8 GB this year. As usual, it was made from the German node (<http://dante.ctan.org>) and does not include the other components of the Collection. It is available to TUG members (and joint members) from the TUG members area, <https://www.tug.org/members>.

- ◇ TeX Collection editors  
Thomas Feuerstack (proTeXt),  
Dick Koch (MacTeX),  
Herb Schulz (MacTeXtras),  
Karl Berry (TeX Live),  
Manfred Lotz (CTAN)  
<http://tug.org/texcollection>

---

**T<sub>E</sub>Xworks — As you like it**

Stefan Löffler

**Abstract**

T<sub>E</sub>Xworks is an ongoing project to create a simple yet flexible editor that “lowers the entry barrier to the T<sub>E</sub>X world”. This article introduces the new T<sub>E</sub>Xworks 0.4 series and discusses several key advancements, in particular the new scripting support.

**1 Introduction**

The T<sub>E</sub>X world is complex. “Beautifully complex”, the expert will say. “Dreadfully complex”, would be a newcomer’s more likely choice of words. This is where T<sub>E</sub>Xworks comes in. Its motto: “*to lower the entry barrier to the T<sub>E</sub>X world*”.

The T<sub>E</sub>Xworks project was launched in 2007, out of discussions between Jonathan Kew, Karl Berry, Dick Koch, and others at several TUG meetings. The plan was to create a new editor, modeled along the lines of the award-winning T<sub>E</sub>XShop application for Mac OS X, but as an open-source, cross-platform program. TUG sponsored some of the initial development.

After two years of development, the first stable release of T<sub>E</sub>Xworks, 0.2.0, was published in 2009. From that point on, odd minor version numbers (0.1, 0.3, . . .) indicated unstable development code, while even numbers (0.2, 0.4, . . .) indicated stable releases for general use.

The hope from there was to speed up development and produce stable releases more often. Due to other responsibilities, Jonathan had less and less time for coding, however, and the next stable release had to be postponed. In 2010, Jonathan handed over a large part of the development responsibilities to me, a fairly regular contributor of patches for quite a while. After wrapping up the dangling threads, the first of the stable 0.4 releases was published in the first half of 2011. Apart from many bug fixes and enhancements, one theme dominated this series: scripting.

**2 Scripting T<sub>E</sub>Xworks**

All the world’s a stage  
 And all the men and women merely players;  
 They have their exits and their entrances,  
 And one man in his time plays many parts.  
 — William Shakespeare, *As You Like It*

For this project, this can be interpreted as: T<sub>E</sub>Xworks is only the core program, the basis upon which every user can personalize the T<sub>E</sub>X editor to their liking and tailor it to meet individual needs.

With the growing popularity of T<sub>E</sub>Xworks and the wide nature of the T<sub>E</sub>X user base, more and

more requests for specific features started coming in. Either for a generally useful addition, to meet an expert’s specific needs, to use T<sub>E</sub>Xworks in some completely unforeseen way, or sometimes for things relevant only to very few, limited, special situations.

In short, with the growing user base, so came an increasing number of divergent ideas for the project’s growth, and it was clear that not everybody’s wishes could be accommodated. We realized that if we adopted ideas that were too specialized, or introduced too great a complexity into the user interface, we would fail in the primary purpose of providing a straightforward editor — one which would not scare off new users. Thus, we faced a dilemma concerning extending the usefulness and versatility of the project, while somehow keeping the standard interface “clean”.

The only flexible solution seemed to be to allow the actual users to change T<sub>E</sub>Xworks to their own liking and needs. Since not everyone is proficient in C++ programming, and a large amount of forked code would be impossible to maintain, letting users *script* new features that were not part of the core application was identified as the best answer, and a lot of effort has gone into that. Scripts can be added any time as they do not need to be compiled. They are simple text files outside the main code, can be deployed as needed, and work on all platforms.

Early attempts at a scripting engine were rudimentary at best. As a proof of concept, it was possible to insert and modify some text from a script, but the C++ internals were very clumsy and hard to maintain.

The real breakthrough came through the discovery that Qt — the Nokia programming framework on which T<sub>E</sub>Xworks is based — allows dynamic access to almost all parts of the core program. There were some coding tricks involved, but exploiting this existing mechanism saved us the work of creating wrappers for each function and variable that script writers may access.

The Qt framework even allows script writers to create forms and dialog windows to interact with the user directly. Nokia provides a free tool, Qt Creator, an IDE with widgets and components, to create these additional user interfaces.

The second major advance was the restructuring of the scripting code to accommodate plugins for additional scripting languages. Presently, apart from the built-in, JavaScript-like QtScript, Lua and Python are available via plugins (on operating systems that support this mechanism). This approach also enables programmers to easily add additional scripting languages to T<sub>E</sub>Xworks.

### 3 How scripts work

The easiest way to use scripts to adapt  $\text{\TeX}$ works to your liking is to get a ready-made script and simply drop it into the  $\text{\TeX}$ works ‘scripts’ folder. This can easily be found using the “Show Scripts Folder” menu item. After dropping your script files in, click “Reload Script List” and you’re ready to go.

$\text{\TeX}$ works scripts come in two varieties: “standalone” scripts and “hook” scripts. “Standalone” scripts appear as menu items in the “Scripts” menu (or one of its submenus). Running such a script is done by clicking on the menu item, or by using a shortcut key (sequence), if one is assigned.

“Hook” scripts, on the other hand, are not directly invoked by the user. Instead,  $\text{\TeX}$ works runs them automatically in certain situations. For example, a hook script could run automatically after a typeset process completes, parse the log output, and present errors or warnings in a user-friendly way.

Scripts can also access files on your hard disk and even execute system commands. To help spare you any severe security problems should an untrusted script be inadvertently run, these features are disabled by default. This can prevent some (advanced) scripts from working properly, however. To enable these advanced features, a one-time authorization must be made in the  $\text{\TeX}$ works preferences dialog.

Other, existing script libraries can be modified and used (e.g., `phpjs` [4]). Among many other things, script system commands can be (and have been) fashioned to: retrieve information from databases or bibliography citations, interact with utilities like `ImageMagick`, and provide additional visual help for  $\text{\LaTeX}$  and others. Combined with script-writer-designed dialogs and forms, the possibilities are very wide, and no C++ knowledge is required!

If you’re interested in writing scripts of your own to make your or your colleague’s life that little bit easier — whether to insert the same text over and over again by a simple key sequence, or write complex scripts for providing input-driven templates — there are a number of resources out there to learn scripting. Of course, knowing your way around one of the supported languages (JavaScript, Lua, Python) in general helps. Other than that, have a look at the  $\text{\TeX}$ works manual [5] for a general, more in-depth discussion of how to use scripts, and Paul A. Norman’s excellent overview of how to manipulate  $\text{\TeX}$ works from within a script [6].

### 4 Other news about $\text{\TeX}$ works

The 0.4 stable series has brought in many other improvements as well. Those who have used previous releases may notice that quite a lot of effort has been

put into further enhancing usability. For one, the presentation of spell checking languages has been improved significantly. Now, human readable names are shown instead of ISO language codes, and languages no longer show up multiple times. In addition, a “follow focus” option has been implemented that can keep the cursor position in the editor and the preview windows synchronized. Syntax highlighting has also been enhanced — it is now possible to set some font modifiers (bold, italic, etc.) and background colors.

$\text{\TeX}$ works’s core has also seen some improvements. Most notably, a new command line parser and an automatic updating mechanism for  $\text{\TeX}$ works resource files have been added. The command line parser enables better integration with other tools that call  $\text{\TeX}$ works (e.g., other editors, previewers, or the operating system).

The automatic updating mechanism will allow future versions of  $\text{\TeX}$ works to upgrade resources like auto-completion files, or syntax highlighting definitions (provided the user does not intervene, of course). Previously, it was necessary to find and delete files manually to cause their update.

This summary of some of the features that stand out to me is expanded in a more complete overview on  $\text{\TeX}$ works’s home page [2]. The screenshot in figure 1 shows  $\text{\TeX}$ works in typical usage.

### 5 Outlook

Far from this being the end of the development of  $\text{\TeX}$ works itself, or of its scripting support, a number of ideas are being canvassed, such as script bundles that can perform a multitude of (typically related) tasks, or scripts that can run in the background — e.g., to perform or monitor a lengthy task — while the rest of  $\text{\TeX}$ works can be used normally. In addition, there are plans to allow scripts to modify the user interface — e.g., by supplying toolbar icons or context menu entries — instead of just showing up in a long list of items in the scripts menu.

Beyond the 0.4 series scripting development, other great ideas have been piling up as well: preview window improvements, tabbed editing, code folding, and project management support, to name only a few. Development for the near future will happen in the 0.5 series (which will become the 0.6 stable release eventually), and will likely focus on areas that have been put on hold in the attempt to get the best out of scripting for  $\text{\TeX}$ works 0.4.

Hopefully, this has got you interested in the  $\text{\TeX}$ works project (or, if you have already been using it, has been a helpful update). If you want to try  $\text{\TeX}$ works out for yourself or upgrade from an earlier version, head over to the home page [2] and

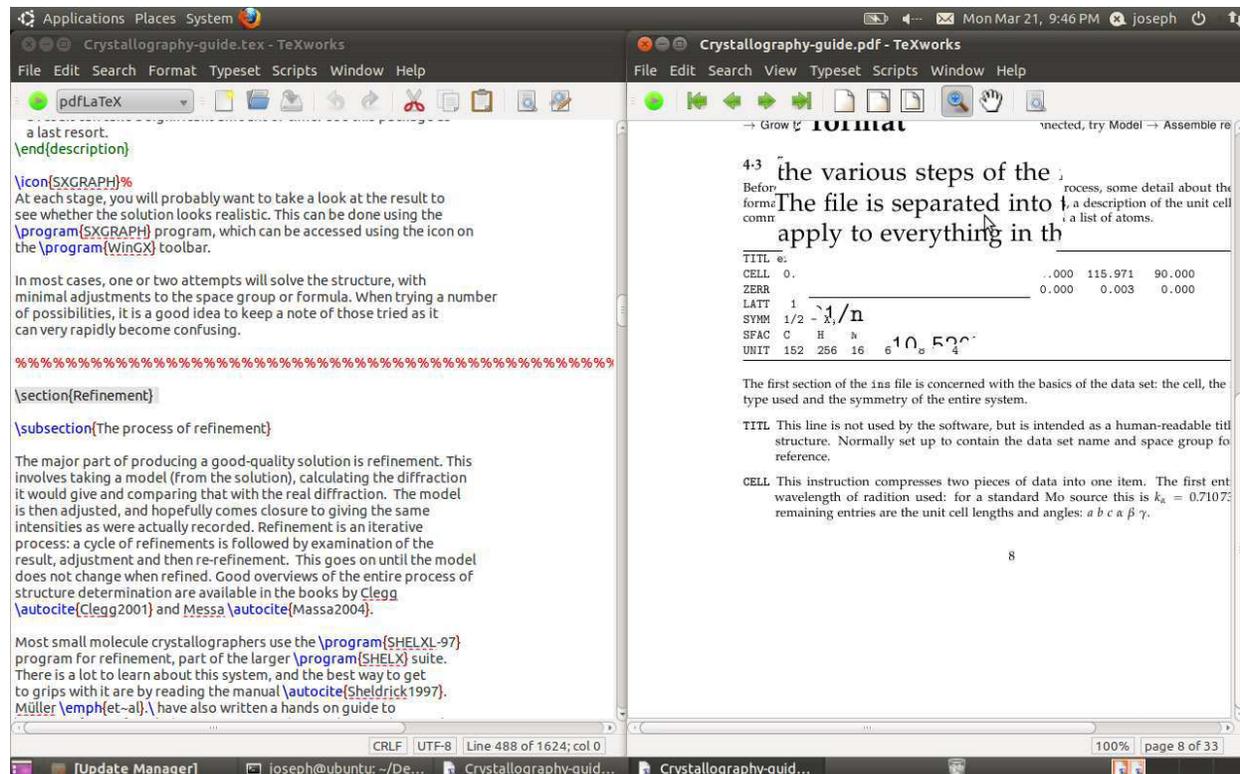


Figure 1: TeXworks source and preview windows on Ubuntu, with area magnified.

grab a copy for your operating system — it’s usually quite simple. And if you want to learn more about TeXworks, or perhaps would like to help in its improvement — by giving feedback, translating, writing manuals, contributing code, or any other way — be sure to check out the pointers given in the References section below.

## 6 Acknowledgements

I want to say a big “thank you” to Paul Norman, who helped in the preparation of this article; to Joseph Wright for providing the screenshot; and to the immensely supportive TeXworks community without which this project wouldn’t be where it is today. And of course to Jonathan Kew for initiating the program, maintaining it, and mentoring me.

- ◇ Stefan Löffler  
Döblinger Hauptstraße 13  
1190 Wien  
Austria  
st.loeffler (at) gmail.com

## References

- [1] TeXworks development home page.  
<http://code.google.com/p/texworks/>.
- [2] TeXworks home page.  
<http://www.tug.org/texworks/>.
- [3] TeXworks mailing list.  
<http://lists.tug.org/texworks>.
- [4] Use PHP functions in JavaScript.  
<http://phpjs.org/>.
- [5] Alain Delmotte and Stefan Löffler. A short manual for TeXworks. Bundled with TeXworks.
- [6] Paul A. Norman. TeXworks scripting.  
<http://twscript.paulanorman.com/docs/index.html>.

---

## MetaPost 1.750: Numerical engines

Taco Hoekwater

### Abstract

After two years of talks about future plans for MetaPost 2.0, finally real progress is being made. This paper introduces a pre-release of MetaPost 2 that can optionally use IEEE floating point for its internal calculations instead of the traditional 32-bit integers.

### 1 Introduction

I am sure some readers are curious to know why it is taking so long before MetaPost 2 comes out, considering that I have been giving talks on the subject for years now. To get started, a recap from the initial project proposal dating back to May 2009:

In the original MetaPost library proposal we wrote in May 2007, one of the big user-side problem points that was mentioned was this:

- All number handling is based on fractions of a 32-bit integer. User input often hits one of the many boundaries that are a result of that. For instance, all numbers must be smaller than 16384, and there is a noticeable lack of precision in the intersection point calculations.

The current proposal aims to resolve that issue once and for all. The goal is to replace the MetaPost internal 32-bit numeric values with something more useful, and to achieve that goal the plan is to incorporate one of these libraries:

GNU MPFR      <http://www.mpfr.org>  
 IBM decNumber <http://www.alphaworks.ibm.com/tech/decnumber>

We have not decided yet which one. MPFR will likely be faster and has a larger development base, but decNumber is more interesting from a user interface point of view because decimal calculus is generally more intuitive. For both libraries the same internal steps need to be taken, so that decision can be safely postponed until a little later in the project. The final decision will be based on a discussion to be held on the MetaPost mailing list.

Since then, there has been a small change to that statement; MetaPost 2 will in fact contain four different calculation engines at the same time:

- scaled 32-bit (a.k.a. compatibility mode)
- IEEE floating point (a.k.a. double)
- MPFR (arbitrary precision, binary)
- decNumber (arbitrary precision, decimal)

The internal structure of the program will also allow further engines to be added in the future.

The traditional scaled 32-bit engine is the default, thus retaining backward compatibility with older versions of MetaPost. The other engines will be selected using a command line switch.

Working backwards from that final goal, some sub-projects could be formulated.

- Because values in any numerical calculation library are always expressed as C pointers, it is necessary to move away from the current array-based memory structure with overloaded members to a system using dynamic allocation (using `malloc()`) and named structure components everywhere, so that all internal MetaPost values can be expressed as C pointers internally.

As a bonus, this removes the last bits of static allocation code from MetaPost so that it will finally be able to use all of the available RAM.

This first sub-project was a major undertaking in itself, and was finally completed when MetaPost 1.5 was released in July 2010.

The current 1.750 release of MetaPost implements most of two other sub-project goals (in fact so far only the PostScript backend has been updated):

- An internal application programming interface layer will need to be added for all the internal calculation functions and the numeric parsing and serialization routines. All such functions will have to be stored in an array of function pointers, thus allowing a start-up switch between 32-bit backward-compatible calculation and the arbitrary precision library.

As a bonus, this will make it possible to add more numerical engines in the future.

- The SVG and PostScript back-ends need to be updated to use double precision float values for exported points instead of the current 32-bit scaled integers.

In the picture export API, doubles are considered to be the best common denominator because there is an acceptable range and precision and they are simple to manipulate in all C code. This way, the actual SVG and PostScript backend implementations and the Lua bindings can remain small and simple.

So, not accounting for hunting for bugs and fixing documentation, there is only one large step that remains to be taken before MetaPost 2 can be released, namely the actual integration of the two arbitrary precision libraries. That is why the version is set at 1.750 at the moment.

## 2 Some internal stuff

One thing that is not immediately obvious from the project goals as written above is that moving all the core arithmetic operations into functions that must be swappable instead of resolved at executable compilation time meant a whole lot of editing work, almost none of which could be automated. This is the main reason why everything took so long. Let me illustrate that with an example.

### 2.1 An example: a simple procedure

Let's look at the `trans` procedure, that applies a transform to a pair of coordinates. It calculates the following formula:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} txx & txy \\ txy & tyty \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} tx \\ ty \end{pmatrix}$$

First, here is the original Pascal implementation of that function:

```
procedure trans(p,q:pointer);
var v:scaled; {the new |x| value}
begin
  v := take_scaled(mem[p].sc,txx)
      + take_scaled(mem[q].sc,txy) + tx;
  mem[q].sc := take_scaled(mem[p].sc,txx)
      + take_scaled(mem[q].sc,tyy) + ty;
  mem[p].sc := v;
end;
```

The meaning of all those variables:

<code>p,q</code>	The variables for the $x$ and $y$ coordinates that have to be transformed
<code>txx,txy,tyx,tyy,tx,ty</code>	The six components of the transformation matrix, in global variables
<code>v</code>	An intermediate value that is needed because $p$ cannot be updated immediately: its old value is used in the calculation of the new $q$
<code>mem[]</code>	The statically allocated memory table where Pascal MetaPost stored all its variables
<code>mem[].sc</code>	The structure object that holds the scaled value of a variable
<code>take_scaled(a,b)</code>	This function calculates $p = \lfloor (a \cdot b) / 2^{16} + \frac{1}{2} \rfloor$

In the conversion of MetaPost from Pascal web to C web (in version 1.2), not that much has changed:

```
static void mp_trans (MP mp,pointer p, pointer q) {
  scaled v; /* the new x value */
  v = mp_take_scaled(mp, mp->mem[p].sc,mp->txx)
      +mp_take_scaled(mp, mp->mem[q].sc,mp->txy)
      +mp->tx;
  mp->mem[q].sc
  = mp_take_scaled(mp,mp->mem[p].sc,mp->txx)
      +mp_take_scaled(mp,mp->mem[q].sc,mp->tyy)
      +mp->ty;
  mp->mem[p].sc = v; }
```

The only big difference here is the use of a global `mp` object instead of global variables. MetaPost 1.5 uses dynamic allocation instead of the `mem` array, and that makes the function a lot easier to understand:

```
static void
mp_trans (MP mp, scaled * p, scaled * q) {
  scaled v; /* the new |x| value */
  v = mp_take_scaled (mp, *p, mp->txx)
      + mp_take_scaled (mp, *q, mp->txy)
      + mp->tx;
  *q = mp_take_scaled (mp, *p, mp->tyx)
      + mp_take_scaled (mp, *q, mp->tyy)
      + mp->ty;
  *p = v;
}
```

It would be great if that could stay, but unfortunately, when numerical variables become objects, it is no longer allowed to use the simple `C +` operator for addition. In turn, that means that more local variables are needed to store intermediate results. To make matters even worse, these local variables have to be allocated and released.

The end result is that the same function looks like this in MetaPost 1.750:

```
static void
mp_number_trans (MP mp, mp_number p,
                 mp_number q) {
  mp_number pp, qq;
  mp_number r1, r2;
  new_number (pp);
  new_number (qq);
  new_number (r1);
  new_number (r2);
  take_scaled (r1, p, mp->txx);
  take_scaled (r2, q, mp->txy);
  number_add (r1, r2);
  set_number_from_addition(pp, r1, mp->tx);
  take_scaled (r1, p, mp->tyx);
  take_scaled (r2, q, mp->tyy);
  number_add (r1, r2);
  set_number_from_addition(qq, r1, mp->ty);
  number_clone(p,pp);
  number_clone(q,qq);
  free_number (pp);
  free_number (qq);
  free_number (r1);
  free_number (r2);
}
```

The variables `r1`, `r1`, `pp` and `qq` exist only for storing intermediate results. To be honest, `qq` is not really needed, but it adds a nice bit of symmetry and the overhead is negligible.

The new arithmetic functions do not return a value since that would force the introduction of even more `new_number` and `free_number` calls. Instead,

they adjust their first argument. Stripped down to only the actual actions, the function looks like this:

```
r1 = p * mp->txx;
r2 = q * mp->txy;
r1 = r1 + r2;
pp = r1 + mp->tx;
```

```
r1 = p * mp->tyx;
r2 = q * mp->tyy;
r1 = r1 + r2;
qq = r1 + mp->ty;
```

```
p = pp;
q = qq;
```

Where the first four lines match the first statement in the previous versions of the function, the next four lines the second statement, and the last two lines do the final assignments.

In the listing above, all those identifiers like `new_number` and `take_scaled` are not really functions. Instead, they are C preprocessor macros with definitions like this:

```
#define take_scaled(R,A,B) \
  (mp->math->take_scaled)(mp,R,A,B)
```

Here the right-hand side `take_scaled` is one of the function fields in the structure `mp->math`. Each of the arithmetic engines defines a few dozen such functions for its own type of `mp_number`. With this new internal structure in place adding a new arithmetic engine is not much more work than defining a few dozen — mostly very simple — functions.

### 3 Using 1.750

As said, there are currently only two engines: scaled 32-bit and IEEE double. Switching to IEEE double is done on the command-line by using

```
mpost --math=double mpm
```

#### 3.1 Warning checks

In MetaPost 1, the parameter `warningcheck` can be set to a positive value. This will downgrade the limit on numerical ranges from 16384 to 4096, but it has the advantage that it guards against various internal cases of overflow.

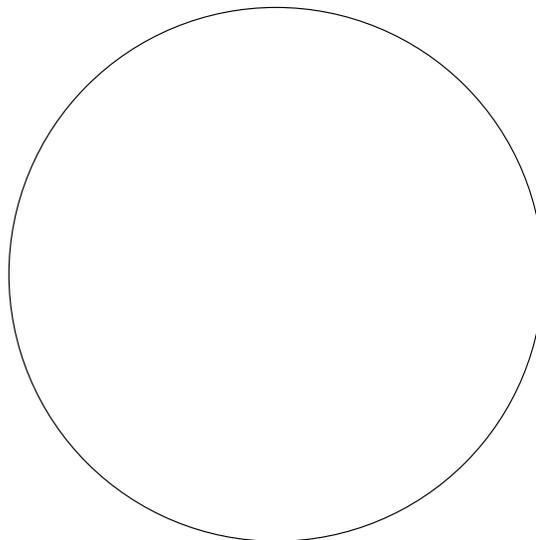
With the `double` numerical engine, numerical values can range up to  $1.0E+307$ . The warning check could be set at something like  $2.5E+306$ , but that is actually not the most important point for a warning to take place.

Because of the way double values are stored internally in the hardware, it is possible to store a

certain range of integers *exactly*. However, when an integer value gets above a threshold (it has to fit in 52 bits), precision is lost. For this reason, `warningcheck` now kicks in a little below this limit, and thus is set at  $4.5E+15$ .

#### 3.2 An example

```
beginfig(1);
warningcheck:=0;
path p;
p = fullcircle scaled 23.45678888E-200;
p := p scaled 1E201;
draw p;
currentpicture := currentpicture scaled .5;
endfig;
end.
```



#### 3.3 Before you try ...

- The current version is of alpha ‘quality’, so lots of bugs are expected.
- Some internals, like `intersectiontimes`, do not take advantage of the extra precision yet.
- The SVG backend is not up to speed yet: it outputs unusable SVG files.

### 4 Planning

A beta release with all four engines is planned for the Summer, then a gamma release with memory leaks fixed (Autumn/Winter), and finally, MetaPost 2.0 (for T<sub>E</sub>X Live 2012).

◇ Taco Hoekwater  
<http://metapost.org>

## Reading and executing source code

Herbert Voß

### Abstract

A frequent question that arises in the various forums is whether specific regions of source code can be displayed both verbatim and with the output of its execution. The packages `fancyvrb` and `listings` support writing to external files and partial reading of source code of arbitrary type. Further packages, such as `showexpl`, allow executing parts of a  $\text{\LaTeX}$  source. This article shows how to apply this to arbitrary types of code.

### 1 Introduction

When creating a manuscript for an article or book, the text is, depending on the subject, augmented by examples that often refer to the output created by a particular programme. It can be beneficial to control the source code for these programmes from within the document to make sure that any changes are reflected in both the source code and the output in the final document. This can avoid mistakes, especially in longer documents.

### 2 Simple $\text{\LaTeX}$ sequences

#### 2.1 Areas of source code

For  $\text{\LaTeX}$  examples, only the source code between `\begin{document}` and `\end{document}` is relevant. The packages `fancyvrb` and `listings` both support specifying an area by line numbers. Such numbers need to be changed, however, when lines are added to or removed from the source code. It therefore makes more sense to specify a string of characters for start and end of the area. The package `listings` provides the option `linenrange`; the specification of the interval is in principle the same as specifying line numbers. Only special characters have to be escaped by prefixing them with a backslash: `\begin\{document\}`. The option `includerangemarker=false` omits the output of the string marking the area; otherwise, the `\begin{document}` and `\end{document}` would appear in the output.

```
\lstinputlisting[
  linenrange=\begin\{document\}-\end\{document\},
  includerangemarker=false]{demo.tex}
```

The command above yields the following source code of a  $\text{\LaTeX}$  document, which will be used as an example throughout this article.

```
\begin{tabular}{@{}
  m{0.5\linewidth}@{}
  >{\lstinputlisting[
    includerangemarker=false,
```

```
  rangeprefix=\%,
  linenrange=START-STOP]{\jobname.tmp}}
  m{0.5\linewidth} @{}
\begin{Example}
\pspicture(3,2)
%START
\psframe*[linecolor=blue!30](3,2)
%STOP
\endpspicture
\end{Example}
& \tabularnewline

\begin{Example}
\pspicture(3,2)
%START
\psframe*[linecolor=red!30](3,2)
\endpspicture
%STOP
\end{Example}
& \tabularnewline
\end{tabular}
```

The same can be achieved with the package `fancyvrb`. The area can be specified through the options `firstline` and `lastline`. The following example outputs its own text body.

```
\documentclass{article}
\usepackage{fancyvrb}
\begin{document}
\VerbatimInput[frame=single,
  fontsize=\footnotesize,
  firstline=\string\begin{document},
  lastline=\string\end{document},
]{\jobname.tex}
\end{document}
```

The `firstline` and `lastline` options define macros `\FancyVerbStartString` and `\FancyVerbStopString`. In special cases, these can be manipulated directly. The macro definition must contain leading white-space if it is present in the source code. The macros do not exist and therefore need to be defined through `\newcommand` or `\edef` if  $\text{\TeX}$ -specific special characters are used, as in this case. The following example outputs the preamble of our sample document. The source document contains two spaces in front of `\begin{document}`, which have to be taken care of through `\space`.

```
\edef\FancyVerbStartString{%
  \string\documentclass{article}}
\edef\FancyVerbStopString{%
  \space\space\string\begin{document}}% 2 spaces
\VerbatimInput[frame=single,fontsize=\footnotesize]
{demo.tex}
```

```
\makeatletter
\let\pc\@percentchar
\makeatother
\usepackage{pstricks,fancyvrb,array,listings}
\lstset{basicstyle=\ttfamily\small}

\def\endExample{\end{VerbatimOut}}
```

```
\def\START{}\def\STOP{}\input{\jobname.tmp}
\newcommand\Example{%
\VerbatimEnvironment
\begin{VerbatimOut}{\jobname.tmp}}
```

## 2.2 Source code and output

Another frequent use case is the display of source code and the result of its compilation with L<sup>A</sup>T<sub>E</sub>X, as in the following example.

foo  
  
bar

```
foo\newline\mbox{%
\put(7,2){%
\circle*{\strip@pt\normalbaselineskip}}}%
\newline
bar
```

The entire source code is not always of interest, as in the example above, which actually contains two additional lines.

```
\makeatletter
%START
foo\newline\mbox{%
\put(7,2){%
\circle*{\strip@pt\normalbaselineskip}}}%
\newline
bar
%STOP
\makeatother
```

To restrict the output to the result of compiling the actual lines, the so-called markers %START and %STOP were added to define the relevant area. They do not affect the result of the compilation as they are prefixed with the L<sup>A</sup>T<sub>E</sub>X comment character %. Of course the comment character should be changed according to the language being used.

Here, to typeset the source code and the output side by side a table was used. The right-hand column is explicitly left blank. The respective command was added to the column definition and only the column separator & must be specified, even if no other material appears in the table.

```
\begin{tabular}{@{}
m{0.2\linewidth}@{}
>{\lstinputlisting[includeonly=false,
rangeprefix=\\,
linerange=START-STOP]{\jobname.tmp}}
m{0.8\linewidth}@{}}
\begin{Example}
\makeatletter
%START
foo \put(12,0){\circle*{\strip@pt\normalbaselineskip}}
\hspace{2\normalbaselineskip}bar
%STOP
\makeatother
\end{Example}
& \tabularnewline
\end{tabular}
```

The environment `Example` uses `fancyvrb` to write everything to a temporary file which is read immediately afterwards through `\input` and thus executed.

```
\newcommand\Example{%
\VerbatimEnvironment
\begin{VerbatimOut}{\jobname.tmp}}
\def\endExample{%
\end{VerbatimOut}
\input{\jobname.tmp}}
```

Instead of a table, a `minipage` could have been used to achieve the arrangement as well. In neither case, however, can page breaks occur within examples. If the output should appear below the source, a different definition must be used. In the following example, a table with normal table header and partial source code is output.

A table without using `tabularx` which is as wide as the line. This is created with this source below, which contains several line breaks.

**Table 1:** Example for calculated table width

foo	bar	baz
and now	and now a	and now a somewhat
a some-	somewhat	longer text to show
what	longer text	line breaks
longer	to show line	
text to	breaks	
show line		
breaks		

```
\begin{tabular}{@{}
>{\RaggedRight}p{1.5cm}|
>{\Centering}p{2cm} |
>{\RaggedLeft}p{\linewidth-3.5cm-4\tabcolsep-0.8pt}
@{}}\hline
foo & bar & baz\\\hline
and now a somewhat longer text to show line breaks &
and now a somewhat longer text to show line breaks &
and now a somewhat longer text to show line breaks\\
\hline
\end{tabular}
```

The source code can now be arbitrarily long as page breaks are possible. The package `fancyvrb` does not support UTF-8 characters; they remain active and would be output in their expanded form. The `inputenx` package provides a workaround, but by default non-ASCII characters have to be specified in their T<sub>E</sub>X-notation, for example `\"u`. The corresponding example environment `ExampleB` for the above example looks like the following:

```
\newcommand\ExampleB{%
\VerbatimEnvironment
\begin{VerbatimOut}{\jobname.tmp}}
\def\endExampleB{%
```

```

\end{VerbatimOut}
{\centering \input{\jobname.tmp}}
\lstinputlisting[
  includerangemarker=false,
  rangeprefix=%,
  linerange=START-STOP]{\jobname.tmp}}

\begin{ExampleB}
\begin{table}[!htb]
\hrulefill\par
A table without using \texttt{tabularx} which is as wide
as the line. This is shown by this text, which contains
several line breaks.
\caption{Example for calculated table width}
%START
\begin{tabular}{@{}
  >{\RaggedRight}p{1.5cm}|
  >{\Centering}p{2cm} |
  >{\RaggedLeft}p{\linewidth-3.5cm-4\tabcolsep-.8pt}
  @{}}\hline
foo & bar & baz\\\hline
and now a somewhat longer text to show line breaks &
and now a somewhat longer text to show line breaks &
and now a somewhat longer text to show line breaks\\\hline
\end{tabular}
%STOP
\end{table}
\end{ExampleB}

```

The reverse order of the output can be achieved by swapping the `\input` and `\lstinputlisting`. For outputting source code, the corresponding command `\VerbatimInput` from the package `fancyvrb` can also be used. Not using it here was an arbitrary decision.

### 2.3 Entire documents

To show the source code and result of entire  $\LaTeX$  document or non- $\LaTeX$  code, a different approach must be taken — a simple `\input` does not work any more. A general solution would be to include the result of the execution of the source code as a figure through `\includegraphics`. If the same font is used as in the document, there will be no difference compared to using `\input`, even for pure text. To identify the externally created figures, a custom counter is defined: `\newcounter{FigureCounter}`. The files are created as `\jobname-\theFigureCounter.tex` and can easily be assigned to source code.

A `Makefile` can be used to simplify the entire procedure of creating the figures independently. After a first `pdf $\LaTeX$`  run, which can use the option `-draftmode` for improved speed, all files with names `\jobname-*` can be run with the respective programme through the `Makefile`. In the example below, `PSTricks` code is processed with `X $\LaTeX$`  to be able to get PDF output. To remove any white margin from the figure, it is processed with `pdfcrop` after the `X $\LaTeX$`  run. The extension of the written files can be used to identify the programme to process

them with, for example, `.cpp` for a C++ example. After all the external files have been created, `pdf $\LaTeX$`  is run again to read the created PDF figures.

The PDF files do not exist at the time of the first `pdf $\LaTeX$`  run. To avoid error messages because of this, their presence is checked through `\IfFileExists`. We now have the following:

```

\newcounter{FigureCounter}
\newcommand\ExampleC{%
\refstepcounter{FigureCounter}%
\VerbatimEnvironment
\begin{VerbatimOut}{\jobname-\theFigureCounter.tex}
\def\endExampleC{%
  \end{VerbatimOut}
  \IfFileExists{%
    \jobname-\theFigureCounter.pdf}% PDF exists?
    {\includegraphics{\jobname-\theFigureCounter.pdf}}%
    {\fbox{PDF missing!}}% no, output message
  \lstinputlisting[
    linerange=\begin\{document\}-\end\{document\},
    includerangemarker=false]%
    {\jobname-\theFigureCounter.tex}}

```

This is only suitable for  $\LaTeX$  or  $X\LaTeX$  documents. The preamble and postamble typical for  $\LaTeX$  are defined as the macros `\FVB@VerbatimOut` and `\FVE@VerbatimOut` from the package `fancyvrb` to avoid the user having to specify them every time.

```

\renewcommand\FVB@VerbatimOut[1]{%
  \@bsphack%
  \begingroup
  \FV@UseKeyValues%
  \FV@DefineWhiteSpace%
  \def\FV@Space{\space}%
  \FV@DefineTabOut%
  \def\FV@ProcessLine##1{%
    \toks@{##1}\immediate\write\FV@OutFile{\the\toks@}}%
  \immediate\openout\FV@OutFile #1\relax%
  \WritePSTricksPreamble%<<=== write preamble
  \let\FV@FontScanPrep\relax
  \let\@noligs\relax%
  \FV@Scan}
\renewcommand\FVE@VerbatimOut{%<<=== write postamble
  \WriteLine{\string\end\{document\}}% <<
  \immediate\closeout\FV@OutFile\endgroup\@esphack}

```

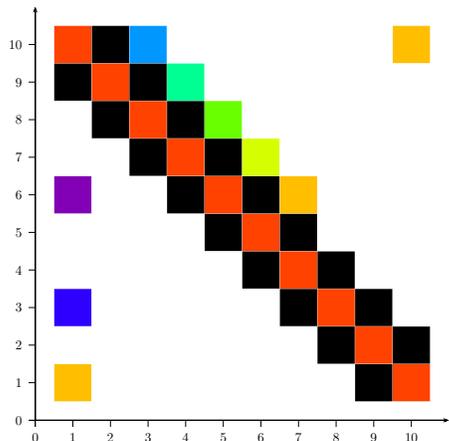
The macro `\WriteLine` allows us to use a specific preamble every time; in the following example, for `PSTricks` code. For a C++ example, a different preamble would be defined.

```

\newcommand\WriteLine[1]{%
  \begingroup%
  \let\protect\@unexpandable@protect%
  \edef\reserved@a{\immediate\write\FV@OutFile{#1}}%
  \reserved@a%
  \endgroup}
\newcommand\WritePSTricksPreamble{%
  \WriteLine{\string\documentclass\{article\}}%
  \WriteLine{\string\usepackage\{pstricks-add\}}%
  \WriteLine{\string\pagestyle\{empty\}}%
  \WriteLine{\string\begin\{document\}}%
}

```

These definitions provide the preliminaries for using the new environment `ExampleC`. The example shown here is a so-called surface plot.



```
\psscalebox{0.5}{%
\begin{pspicture}(-0.5,-0.75)(11,11)
\psaxes[ticksize=-5pt 0]{->}(11,11)
\psMatrixPlot[colorType=5,dotsize=1.1cm,
xStep=1,yStep=1,
dotstyle=square*]{10}{10}{matrix1.data}
\end{pspicture}}
```

To make the preamble more flexible and be able to output parts of it as code, two additional macros can be used to specify the invisible and the visible part of the preamble. Here, the invisible part is output into the external file first, but this is arbitrary. The macro `\FVB@VerbatimOut`, which was modified above already, is changed again to omit a preamble for a special case (`\WritePSTricksPreamble`), instead now including a template preamble (`\WritePreamble`). The template preamble does not load additional packages; they have to be loaded by the user. This allows for more flexibility. The package `listings` allows not only specifying a region, but also a sorted, comma-separated list in curly braces.

```
\lstinputlisting[
linerange={\%PSTART-\%PSTOP,%
\begin{\document}\-\end{\document}\},
includerangemarker=false]{%
\jobname-\theFigureCounter.tex}%
```

In this case, everything in the source document between `%START` and `%STOP` and also between `\begin{document}` and `\end{document}` is output. To distinguish between preamble and text body, they are output with different background colours and a small additional space between them. The part to insert the code is now

```
\IfFileExists{\jobname-\theFigureCounter.pdf}%
{\begin{center}\expandafter\includegraphics%
\expandafter\GraphicxOptions}%
```

```
{\jobname-\theFigureCounter.pdf}
\end{center}}%
{\fbox{PDF missing!}}%
\def\GraphicxOptions{%
\lstinputlisting[backgroundcolor=\color{black!10},
linerange=\%PSTART-\%PSTOP,
includerangemarker=false,]%
{\jobname-\theFigureCounter.tex}
\lstinputlisting[backgroundcolor={},
linerange=\begin{\document}\-\end{\document}\},
includerangemarker=false]%
{\jobname-\theFigureCounter.tex}%
\gdef\Invisible@Part{}%
\gdef\Visible@Part{}%
```

The macro `\GraphicxOptions` saves the optional parameter of the `ExampleD` environment, which may contain key/value pairs for `\includegraphics`. The invisible part of the preamble is passed as an argument to the macro `\PreambleInvisible` and the visible part to `\PreambleVisible`. The external  $\text{\LaTeX}$  document now has the following preamble.

```
\newcommand\WritePreamble{%
\WriteLine{\string\documentclass{article}}%
\WriteLine{\string\pagestyle{empty}}%
\WriteLine{\Invisible@Part}
\WriteLine{@percentchar PSTART}
\WriteLine{\Visible@Part}%
\WriteLine{@percentchar PSTOP}
\WriteLine{\string\begin{document}}%
}
```

A page break is now possible after the figure and within the code output, as shown by the following example.

The binding energy in the liquid drop model is composed of the following parts.

- the surface part,
- the volume part,

$$E = a_v A + -a_f A^{2/3} + -a_c \frac{Z(Z-1)}{A^{1/3}} + -a_s \frac{(A-2Z)^2}{A} + E_p \quad (1)$$

- the Coulomb part,
- the asymmetry part,
- and a pairing part.

```
\usepackage{tgpagella}
\usepackage{pst-node}
```

```
\psset{nodesep=3pt}
The binding energy in the liquid drop model is composed
of the following parts.
\begin{itemize}
\item the \rnode{b}{surface part},
\item the \rnode{a}{volume part},\ll[1cm]
\def\xstrut{\vphantom{\frac{(A)^1}{(B)^1}}}
\begin{equation}
E =
\rnode[t]{ae}{\psframebox*[fillcolor=black!8,
```

```

linestyle=none}{\xstrut a_vA}} +
\rcode[t]{be}{\psframebox*[fillcolor=black!16,
linestyle=none]{\xstrut -a_fA^{2/3}}} +
\rcode[t]{ce}{\psframebox*[fillcolor=black!24,
linestyle=none]{\xstrut -a_c\frac{Z(Z-1)}{A^{1/3}}}} +
\rcode[t]{de}{\psframebox*[fillcolor=black!32,
linestyle=none]{\xstrut -a_s\frac{(A-2Z)^2}{A}}} +
\rcode[t]{ee}{\psframebox*[fillcolor=black!40,
linestyle=none]{\xstrut E_p}}
\end{equation}\[0.25cm]
\item the \rcode{c}{Coulomb part},
\item the \rcode{d}{asymmetry part},
\item and a \rcode{e}{pairing part}.
\end{itemize}
\ncurve[angleA=-90,angleB=90]{->}{a}{ae}
\ncurve[angleB=45]{->}{b}{be}
\ncurve[angleB=-90]{->}{c}{ce}
\ncurve[angleB=-90]{->}{d}{de}
\ncurve[angleB=-90]{->}{e}{ee}

```

The macro `\Preamble`, which saves the invisible and the visible part of the preamble, is somewhat more complex because the special characters like `\`, `$`, `&`, `#`, `^`, `_`, `%` and `~` and line endings must be handled separately. If an arbitrary optional argument is specified, it is assumed that it is the invisible part of the preamble.

```

\def\MakeVerbatimNewLine{^^}
\begingroup
\catcode'\^^M=\active %
\gdef\obeylines@Preamble{\catcode'\^^M\active
\let^^M\MakeVerbatimNewLine}%
\endgroup

\newcommand\Preamble{%
\par
\begingroup
\makeatother
\let\do\@makeoother
\do\ \do\ \do\ \$\do\&\do\#\do\^\do\_ \do\~ \do\%
\obeylines@Preamble
\@ifnextchar[\PreambleInvisible@{\PreambleVisible@{}}]
\long\def\PreambleInvisible@#1#2{%
\long\xdef\@tempa{#2}%
\endgroup\let\Invisible@Part\@tempa}
\long\def\PreambleVisible@#1#2{\long\xdef\@tempa{#2}%
\endgroup\let\Visible@Part\@tempa}

```

This can be used to control the output of the preamble in the example code. Only the parts which are of interest to the reader can be output while other things can be defined as well and written into the exported `TeX` file, but do not appear as source code in the final document. For the example above:

```

\PreambleInvisible{\usepackage[T1]{fontenc}
\usepackage{mathpazo}
\usepackage{pstricks}
}

\Preamble{\usepackage{tgpagella}
\usepackage{pst-node}}

```

### 3 Arbitrary source code type

It has already been mentioned that in principle any language can be used in the exported file. The `Makefile` can do the appropriate processing based on the extension of the file. Only the example environment must know the type of file to be exported. Our final example shows an external Perl programme, which is written from this document and executed. The output of the programme is saved with the same base name and the extension `.out`. Finally, the output is inserted back into this document as pure text.

A standardised Perl code could have the following header (preamble).

```

\newcommand\SchreibePerlPraeambel{%
\WriteLine{\numbersign !/usr/bin/perl}%
\WriteLine{\numbersign }%
\WriteLine{\numbersign Herbert Voss 20110201}%
\WriteLine{use strict;}%
\WriteLine{\Invisible@Part}
\WriteLine{\numbersign PSTART}
\WriteLine{\Visible@Part}%
\WriteLine{\numbersign PSTOP}
\WriteLine{\numbersign }%
\WriteLine{\numbersign bodystart!!}%
}

```

The definition of the example environment is in principle the same as the `LATEX` version shown above. Instead of including a generated PDF file, the text output created by the external Perl programme is input with `\lstinputlisting`. The optional argument of the environment `ExampleE` can be used to specify the formatting; it is passed through to `\lstinputlisting`.

```

\newcommand\ExampleE[1][{}%
\def\lstOptions{#1}%
\refstepcounter{FigureCounter}%
\VerbatimEnvironment
\begin{VerbatimOut}{\jobname-\theFigureCounter.pl}
\def\endExampleE{%
\end{VerbatimOut}
\IfFileExists{\jobname-\theFigureCounter.out}%
{\expandafter\lstinputlisting\expandafter{\lstOptions}%
{\jobname-\theFigureCounter.out}}%
{\fbox{Output missing!}}%
\medskip
\def\lstOptions{%
\lstinputlisting[backgroundcolor=\color{black!10},
linenrange=#PSTART-#PSTOP,
includerangemarker=false,]%
{\jobname-\theFigureCounter.pl}
\lstinputlisting[
backgroundcolor={},
linenrange=\#bodystart!!-\#bodyend!!,
includerangemarker=false]%
{\jobname-\theFigureCounter.pl}%
\gdef\Invisible@Part{}%
\gdef\Visible@Part{}%
}

```

The following example determines so-called Kaprekar constants (see [http://en.wikipedia.org/wiki/Kaprekar\\_constant](http://en.wikipedia.org/wiki/Kaprekar_constant)). These are natural numbers with the following properties: If the digits are sorted ascending and descending, the result is a largest and a smallest number whose difference is the same as the original number. The algorithm here uses a brute force approach to keep the code simple; all numbers are generated and tested.

```

Determining Kaprekar constants
1 digits:
2 digits:
3 digits: 495,
4 digits: 6174,
5 digits:

```

```

### Determining Kaprekar constants ###
my $number = 1;
my $start = 1;
my $end = 10;

```

```

print("Determining_Kaprekar_constants\n");
while ($number < 6) {
  print "$number_digits:_" ;
  foreach ($start...$end) { # for each line $_
    my @chars = split(//,$_);
    my $Min = join("",sort(@chars));
    my $Max = reverse($Min);
    my $Dif=$Max-$Min;
    if($_ eq $Dif) { print $_,","; }
  }
  $number = $number+1;
  $start = $start*10;
  $end = $end*10;
  print "\n"; }

```

The output of the entire Perl programme is the same as for a  $\text{\LaTeX}$  example as well.

```

\Preamble[Invisible]{# Example for a Kaprekar constant}
\Preamble{### Determining Kaprekar constants ###}
my $number = 1;
my $start = 1;
my $end = 10;

\begin{ExampleE}[basicstyle=\ttfamily\footnotesize,
                frame=LR]
print("Determining_Kaprekar_constants\n");
while ($number < 6) {
  print "$number_digits:_" ;
  foreach ($start...$end) { # for each line $_
    my @chars = split(//,$_);
    my $Min = join("",sort(@chars));
    my $Max = reverse($Min);
    my $Dif=$Max-$Min;
    if($_ eq $Dif) { print $_,","; }
  }
  $number = $number+1;
  $start = $start*10;
  $end = $end*10;
  print "\n"; }
\end{ExampleE}

```

## 4 Creating this document

This document creates several external example files which are then run by  $\text{\XeLaTeX}$  and Perl. The created PDFs from  $\text{\XeLaTeX}$  are cropped to eliminate the whitespace and then inserted as PDF graphics, while the output from the Perl program is inserted as a text file. All this is done for Linux with the following simple shell script:

```

#!/bin/sh
pdflatex voss2011 # main doc
xelatex voss2011-1.tex # 1st created external file
pdfcrop voss2011-1 # cut whitespace
mv voss2011-1-crop.pdf voss2011-1.pdf # rename
xelatex voss2011-2.tex # 2nd created external file
pdfcrop voss2011-2 # cut whitespace
mv voss2011-2-crop.pdf voss2011-2.pdf # rename
perl voss2011-3.pl > voss2011-3.out # 3rd external file
pdflatex voss2011 # main doc

```

## 5 Summary

This article has shown how to create external source files of arbitrary types and execute them through a Makefile after a first  $\text{\LaTeX}$  run. The file extension of the created file should designate the type of programme used for its execution. The output of the programmes can be included in subsequent  $\text{\LaTeX}$  runs as figures or text. The author retains the full control over example programmes. If there is a large number of examples, the created file can be written into a temporary directory and compared with an existing file through the Unix `diff` command to avoid executing the programme again if the source code has not changed.

There are more optional parameters possible for inserting the output into the document to, for example, specify left/right alignment. Figures could be processed with `pdfcrop` to remove white margins.

## References

- [1] Carsten Heinz: *The listings package*, Version 1.4, Feb. 2007; [mirror.ctan.org/macros/latex/contrib/listings/](http://mirror.ctan.org/macros/latex/contrib/listings/)
- [2] Rolf Niepraschk: *The showexpl package*, Version 0.3h, Feb. 2007; [mirror.ctan.org/macros/latex/contrib/showexpl/](http://mirror.ctan.org/macros/latex/contrib/showexpl/)
- [3] Timothy Van Zandt: *The fancyvrb package — Fancy verbatims in  $\text{\LaTeX}$* , Version 2.8, May 2010; [mirror.ctan.org/macros/latex/contrib/fancyvrb/](http://mirror.ctan.org/macros/latex/contrib/fancyvrb/)

◇ Herbert Voß  
 Wasgenstraße 21  
 14129 Berlin, Germany  
 herbert (at) dante dot de  
<http://tug.org/PSTricks>

---

## Teaching L<sup>A</sup>T<sub>E</sub>X to the students of mathematics — The experience from the Jan Kochanowski University

Krzysztof Pszczoła

### Abstract

Two years of teaching L<sup>A</sup>T<sub>E</sub>X to the students of the mathematical institute and checking almost 200 students' papers gave me some fresh thoughts. The main observation concerning the aesthetics and effectiveness of the message may be formulated as follows: the issues concerning the microtypography are usually missed, but — after pointing the students' attention to them — are appreciated.

In my talk I discuss, besides the ideas concerning the perception of microtypography by students, other aspects of the university course preparing the students of mathematics to typeset their theses in L<sup>A</sup>T<sub>E</sub>X. It might be interesting especially for those who teach similar courses (but not only to them).

### 1 Introduction

Currently in many universities in Poland, theses in mathematics must be written in L<sup>A</sup>T<sub>E</sub>X. There are special courses to help students learn L<sup>A</sup>T<sub>E</sub>X. I have been teaching such a course in the Institute of Mathematics, Jan Kochanowski University in Kielce, for two years and I'd like to share some experiences.

In my L<sup>A</sup>T<sub>E</sub>X course I speak not only about L<sup>A</sup>T<sub>E</sub>X, but also about typography and mathematical editing in general. In many cases I start with a description of the effect we want to obtain before introduction of L<sup>A</sup>T<sub>E</sub>X commands. Sometimes we also talk about more general issues, such as “what is the real size of the 11pt font” or “why A4 paper is just 210x297mm”.

### 2 Missed microtypography

Let's look at the following example: find the difference between these two lines:

Some text *italic text* and some more text

Some text *italic text* and some more text

(Obtained with `\textit{italic text}` and `{\it italic text}`.)

Usually in the class of approx. 15 students, it takes 2–3 minutes to get the first answer, which sounds something like “the second one is shorter”. Sometimes I ask: “Which of these lines is more comfortable to read?”. Usually the first answers are “the same”, but after a while, someone hits the point: “the first one is more comfortable to read, because all the spaces there have equal width”. And this is

how we've reached the concept of italic correction, and even more: microtypography.

For students it is a very new concept that details which are so hard to notice are important for the reading comfort. But they accept the explanation of this fact. The human eye (or, speaking more precisely, the eye+brain system) gets accustomed to the font shape, word-spacing and so on. And if we change something — we need to get accustomed again. So we can conclude: issues concerning microtypography are usually missed, but — after drawing the students' attention to them — they are appreciated.

### 3 Some details

The course is held for the students of the first year of mathematics. The whole course takes 15 hours and at the end students have to present something written in L<sup>A</sup>T<sub>E</sub>X. The students' works are carefully checked and the author gets a detailed report, with editorial, typographical and L<sup>A</sup>T<sub>E</sub>X-specific mistakes pointed out.

The main goal of this course is obvious: to teach students L<sup>A</sup>T<sub>E</sub>X to an extent such that they could write their theses using this system. Surely, when one is able to write a thesis in L<sup>A</sup>T<sub>E</sub>X, he is also able to write an article and other materials. Additionally, I talk about creating presentations (`beamer` class) and posters (`a0poster` class). Also issues concerning the installation of T<sub>E</sub>X are discussed.

The second goal is to develop some basic editorial and typographic skills.

The next goal is to introduce a basis of editing mathematical texts.

Finally, I hope that some of my students will like L<sup>A</sup>T<sub>E</sub>X after attending this course.

At the beginning I try to “familiarize” students with basic concepts related to T<sub>E</sub>X, L<sup>A</sup>T<sub>E</sub>X, and the T<sub>E</sub>XnicCenter editor we use in computer labs. My experience is that a less formal introduction makes the whole course more effective. Then I talk about the most basic L<sup>A</sup>T<sub>E</sub>X concepts like sectioning, cross-references, creation of the bibliography, tabular material, graphics, floats, writing mathematical equations and theorem-like environments. At the end I discuss a few less basic concepts, like the `beamer` class, creating posters with the `a0poster` class, and a few words about MetaPost.

◇ Krzysztof Pszczoła  
Instytut Matematyki Uniwersytetu  
Humanistyczno-Przyrodniczego  
Jana Kochanowskiego w Kielcach  
`pszczoła (at) fr dot pl`

---

## Drawing tables: Graphic fun with LuaTeX

Paul Isambert

### Introduction

I was deeply interested by Paweł Jackowski’s paper in *TUGboat* 32:1. Paweł explained how graphic manipulations could be made clean and simple and powerful in LuaTeX. He also mentioned a partial PostScript interpreter, so he can draw in PostScript directly. The idea appealed to me — until I remembered I don’t know PostScript; so I thought: why use PostScript at all? Why not Lua as a language for graphics?

I set to work and discovered a wonderful world. Not being a mathematician (and computer graphics require a good deal of math, if not especially advanced), and doing little graphics anyway, the code I’ve written might never become a public package, and I’d hate to see the ideas wasted. So I’ll describe some of them here; meanwhile, it will also let us learn a lot about tables, Lua’s principal basic data structure, among other Lua and LuaTeX features.

### State of the art

Drawing requires an input language (for the user to describe what s/he wants) and an output language (for the viewer to render); in an ideal world, both coincide, as is the case with PostScript.

On the other hand, the PDF format is quite lame (on purpose) when it comes to drawing. The PDF format doesn’t know anything about variables, functions, or flow control. You want to draw the same line three times in different places? Well, you have to describe it in its entirety three times; you can’t say such things as “let  $l$  be a length, draw a line of length  $l$  here and there”, not to mention something like “if this is the first line, draw it in red, otherwise draw it in blue”. What does PDF understand, then? Not much more than “go here”, “draw a line or a Bézier curve from here to there” (with *here* and *there* as full coordinates, not variables), “fill this shape”, “use a line of width  $w$ ”. To put it simply, PDF is unusable as an input language.

On the other hand, more than thirty years after its conception, TeX still can’t draw at all (also on purpose). It’s not even good at basic mathematical operations required by drawing, e.g. computing a cosine. That’s why graphic extensions have been developed to provide a comfortable environment to users. The three best-known are (I can’t do them

justice in these brief descriptions — please read their manuals for a better view):

—*MetaPost* is a language and an interpreter, inspired by METAFONT, which produces PostScript graphics (and SVG since v.1.2); the figures are then added to the DVI file with `\special`, to be inserted directly in the PostScript file (with `dvips`), and the latter is converted to PDF with `ps2pdf`. The MetaPost compiler is embedded in LuaTeX, and thus it no longer requires an external program (or external files), but its output still requires conversion to PDF.

—*PSTricks* is a TeX interface for the PostScript language; the same road as with MetaPost must be used to produce PDF. PSTricks also provides such basic structures as loops, missing in TeX.

—*PGF/TikZ* directly produces either PostScript or PDF code, depending on the driver with which it is used. Hence no conversion is needed. Like PSTricks, PGF offers a proper input language.

The strengths of these three approaches are well-known, and they are justly popular. What then would be their weaknesses (with a little bad faith), or rather, what would be the advantages of a Lua-driven approach?

—*No need for a new language.* There is Lua, let’s use Lua; variables or flow control are readily available. Well, of course, you have to learn Lua, but using LuaTeX without knowing anything about Lua would be a shame. Also, Lua is known for its simplicity and is intended to be easy to learn.

—*Benefit from a real programming language.* Lua has the usual mathematical functions which are (of course) very useful when drawing (like computing a cosine). But there is more. Suppose you want to plot data, and those data are in an external file in whatever format. It is easy to make Lua parse the file and extract the information. In other words, when you need programming that is not directly related to drawing, no need to consider dirty tricks or to pray that a package exists: just use the same language you’re already using.

—*Access to TeX’s internals.* Of course LuaTeX provides a Lua interface to TeX; the contents of a box, the current font or the position on the page can be queried at once. There is no disconnection between drawing and typesetting.

—*Transparency.* That is, to me, the most important feature. As we will see in what follows, you can keep the implementation quite transparent, in the sense that what is constructed can be manipulated directly (i.e. without especially-tailored functions),

because objects (points, paths, ...) are simple tables. That means that the user can be given maximum control; actually, we don't *give* anything, but simply avoid taking it away.

### From Lua to PDF

Now, we need to define some basic functions to produce PDF output. We'll focus on a few PDF operators (there aren't many more anyway):

`x y m` Move to point  $(x, y)$ , which becomes the current point.

`x y l` Append a straight line from the current point to  $(x, y)$ . In this operation and the next, the ending point becomes the current point.

`x1 y1 x2 y2 x3 y3 c` Append a cubic Bézier curve from the current point to  $(x3, y3)$  with control points  $(x1, y1)$  and  $(x2, y2)$ .

`h` Close the current path.

`l w` Sets the line width to  $l$  (to be used by the next `S` statement).

`S` Stroke the current path.

PDF's default unit is the PostScript point (the big point, `bp`, in `TEX`), and that's what we'll use here, even though it would be quite simple (and actually quite necessary) to define a whatever-unit-to-`bp` function.

As an example, let's draw a simple triangle:

```
0 0 m 20 0 l 10 15 l h S
```



Let's now define a Lua interface to these operators:

```
function pdf_print (...)
  for _, str in ipairs({...}) do
    pdf.print(str .. " ")
  end
  pdf.print("\string\n")
end
function move (p)
  pdf_print (p[1], p[2], "m")
end
function line (p)
  pdf_print(p[1], p[2], "l")
end
function curve(p1, p2, p3)
  pdf_print(p1[1], p1[2],
            p2[1], p2[2],
            p3[1], p3[2], "c")
end
function close ()
  pdf_print("h")
end
function linewidth (w)
  pdf_print(w, "w")
end
```

```
function stroke ()
  pdf_print("S")
end
```

The syntax, which uses tables, will be explained in the next section. We use the `pdf.print()` function, which makes sense only in a `\latelua` call. So let's define our macro to tell `TEX` we're switching to Lua drawing (it is `\long` to allow blank lines in the code; the resulting `\par`'s are filtered in `\latelua`):

```
\long\def\luadraw#1 #2{%
  \vbox to #1bp{%
    \vfil
    \latelua{pdf_print("q") #2 pdf_print("Q")}%
  }%
}
```

(The `q` and `Q` operator ensures the graphic state remains local; this is PDF's grouping mechanism, so to speak.) And now the triangle can be drawn more simply as follows. The Lua syntax `foo{...}` is a function call, equivalent to `foo({...})` when the function takes a single table as its argument.

```
\luadraw 17 {
  move{0, 0}
  line{20, 0} line{10, 15}
  close() stroke()
}
```

Well, simple? Maybe not. But at least we can use loops. Here are three triangles growing in height (from now on I won't show the enclosing `\luadraw`):

```
for i = 1, 3 do
  local x, y = i * 30, 15 + 5 * i
  move{x, 0}
  line{x + 20, 0} line{x + 10, y}
  close() stroke()
end
```



The reader might think that explicitly specifying the height (and width, if things were to be done properly) for each picture is annoying; can't it be computed automatically? Well, yes and no. Yes, because you can analyse the points in the picture and retrieve the bounding box; no, because with `\latelua` this would be done at shipout time, long after the box has been constructed (so the information comes too late).

To do things properly, one should compile the code with `\directlua` and store all drawing statements in a `\latelua` node created on the fly; so relevant information could be retrieved (in both directions) when needed, whereas the PDF code is written at shipout time, as it should be. I won't investigate that tricky issue here.

### More to the point

The functions above assume that points are denoted as tables with (at least) two entries: the entry at index 1 is the x-coordinate, the one at index 2 is the y-coordinate. This is already much more powerful than it seems; for starters, you can define and reuse variables. Here's a Bézier curve with the end points attached to the control points:

```
local a, b, c, d = {0,0}, {10,20},
                  {35,25}, {45,5}
move(a) curve(b, c, d) stroke()
move(a) line(b)
move(d) line(c)
linewidth(.2) stroke()
```



But the real power comes from the ease of use of such structures. Suppose you want to scale a picture by  $x$  in the x-direction and  $y$  in the y-direction. The function to do that (working here on points) is utterly simple:

```
function scale (p, x, y)
  p[1], p[2] = p[1] * x, p[2] * y
end
```

Then if we add `scale(a, 2, 1)`, `scale(b, 2, 1)`, `scale(c, 2, 1)` and `scale(d, 2, 1)` to our previous code, we get:

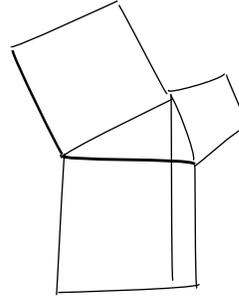


Other transformations, like rotation or translation, can be as easily defined, as can any operations involving tables. This means that any drawing system in Lua is highly extensible, and most importantly that it can be mastered deeply by the users without much effort. That is what I meant above by *transparency*.

Instructive playtime: let's illustrate Pythagoras' theorem with a hasty quill.

```
local function rand ()
  return math.random(-100, 100) / 60
end
local function randomline(p1, p2)
  local c1 = {p1[1] + rand(), p1[2] + rand()}
  local c2 = {p2[1] + rand(), p2[2] + rand()}
  p1[1], p1[2] = p1[1] + rand(), p1[2] + rand()
  p2[1], p2[2] = p2[1] + rand(), p2[2] + rand()
  linewidth(math.max(.5, rand()/1.5))
  move(p1) curve(c1, c2, p2) stroke()
end
local a, b, c = {20,50}, {60,70}, {70,50}
local ab1, ab2 = {0,90}, {40,110}
local bc1, bc2 = {80,80}, {90,60}
local ca1, ca2 = {70,0}, {20,0}
```

```
randomline(a,b) randomline(b,c) randomline(c,a)
randomline(a,ab1) randomline(ab1,ab2)
  randomline(ab2,b)
randomline(b,bc1) randomline(bc1,bc2)
  randomline(bc2,c)
randomline(c,ca1) randomline(ca1,ca2)
  randomline(ca2,a)
randomline(b, {b[1], ca1[2]})
```



The `randomline()` function turns a line from `p1` to `p2` into a Bézier curve with the same endpoints, albeit slightly displaced, and control points close to those endpoints, so the curve approximates a line. The line width is randomized too. All in all, it boils down to manipulating table entries.

Here I have set the points so a right triangle is drawn with a square on each side. It is not difficult to find those points automatically, once `a` and `b` are given, and such functions are of course vital (e.g. "find a point that is on a line perpendicular to another line"); again that is table manipulation with a bit of math. That's what I have done for the endpoint of the vertical line from the right angle: it depends on other points.

Let's get back to tables. The reader might have remarked that the `scale()` function above didn't return anything, so one could not assign its result to a variable. What, then, if one wants to have a point `P` which is `p` scaled, but leaving `p` untouched? The reader might think of something like this:

```
local P = p; scale(P, 2, 1)
```

That is a very bad idea: variables only point to tables, so in this case `p` and `P` point to the same table, and changing `P` also changes `p`. If one wants a table similar to another one, one should copy it:

```
function copy_table (t)
  local T = {}
  for k, v in pairs(t) do
    if type(v) == "table" then
      v = copy_table(v)
    end
    T[k] = v
  end
  setmetatable(T, getmetatable(t))
  return T
end
```

This function creates a new table and sets all its entries to the values in the original table; if a value is a table, the function calls itself, so all subtables are properly copied too. The `set/getmetatable()` functions will be explained presently.

The function also illustrates the `pairs()` iterator: given a table, it loops over all entries, in no particular order, returning the key and the value for each. There also exists the `ipairs()` iterator, which browses only entries with an integer key, from 1 to the first missing integer (for instance, a table with entries at indices 1, 2 and 4 will be scanned with `ipairs()` for the first two entries only).

Having to declare `P = copy_table(p)` is a bit of an overkill (although it can't be avoided sometimes); instead, all functions manipulating tables should copy them beforehand, and return the new table if necessary. So we could rewrite `scale()` as:

```
function scale (p, x, y)
  local P = copy_table(p)
  P[1], P[2] = P[1] * x, P[2] * y
  return P
end
```

Now one can say `local P = scale(p,2,1)` and `p` will be left unmodified; and if one wants to keep the same variable, then: `p = scale(p,2,0)`. If the original table denoted by `p` isn't referred to by another variable, it will eventually be deleted to save memory.

### Metatables: The fast lane to paths

Up to now, we have drawn lines and curves one by one, and that is not very convenient; it would be simpler if one could define a sequence of points to describe several lines and curves. Moreover, it would be better still if one could assign paths to variables instead of drawing them at once; then one would be able to manipulate and reuse them.

What should be the structure of such a path? For Lua, a table with subtables representing points is a natural choice. However, not all points are equal: some are endpoints to lines, some do not define a line but a movement, some are control points. So they should have an entry, say `type`, to identify themselves.

As an example, let's make a table to represent moving to (10,10), then drawing a line to (20,10), then a curve to (0,0) with control points (20,5) and (5,0). This also illustrates how entries in tables are declared: either with an explicit key, like `type`, or implicitly at index  $n$  if the entry is the  $n$ th implicit one.

```
local path = {
  type = "path",
  {10, 10, type = "move"},
  {20, 10},
  {20, 5, type = "control"},
  {5, 0, type = "control"},
  {0, 0}
}
```

Note that endpoints have no `type` entry, so they are considered the default points; on the other hand, the path itself has a `type`, to be used below. Before implementing such a construction, we need a new function to draw the path. It will work as follows: if a point is of the `move` type, use `move()` with it; if it is of the `control` type, store it; finally, if it has no `type`, use `line()` with it, or `curve()` if control points have been stored. Also, the first point necessarily triggers a `move()` command, whatever its type — it wouldn't make sense otherwise. We could also add dummy point of type `close` to call the `close()` function, but let's stick to the essentials. Here we go:

```
function draw (path)
  local controls
  for n, p in ipairs(path) do
    if n == 1 then
      move(p)
    elseif p.type == "move" then
      move(p)
    elseif p.type == "control" then
      controls = controls or {}
      table.insert(controls, p)
    else
      if controls then
        curve(controls[1],
              controls[2] or controls[1],
              p)
      else
        line(p)
      end
      controls = nil
    end
  end
  stroke()
end
```

The `controls[2] or controls[1]` construct means: use the second control point if it exists, the first one otherwise; i.e. draw a curve with overlapping control points. (A better alternative would be for a single control point to signal a quadratic Bézier curve; then with a little bit of math we could render it with a cubic.)

Now, how shall we define paths? We can use explicit tables, as above, but it's obviously inconvenient. We could use a `path()` function which, given any number of points with an associated type, would

return a path. But the top-notch solution would be to be able to use a natural syntax, such as:

```
local path = a .. b - c - d .. e + f .. g
```

meaning: move to **a**, append a line to **b**, then a curve from **b** to **e** with control points **c** and **d**, then move to **f** and append a line to **g**.

Alas, the Lua `..` operator is meant to concatenate strings, whereas the arithmetic operators obviously require numbers ... and we have defined points as tables. Shall we find another way?

No, definitely not: we'll make tables behave as we want. To do so, we need metatables. What is that? A metatable `mt` is a table with some special entries (along with ordinary entries) which determine how a table `t` with `mt` as its metatable should behave under some circumstances. The best-known of those entries is `__index`, a function (or table) called when one queries a nonexistent entry in `t` (an obvious application is inheritance).

Here we should define points as tables with a metatable with `__concat`, `__add` and `__sub` entries, which determine the behavior when the tables are passed as operands to those operators. The syntax should be as follows: if two points are connected by one of those operators, they should produce a path; if two paths, or a point and a path, are the operands, the same result should occur. In the example above, `a .. b` should produce a path, to which then `c` is added, etc.\* Here are the functions:

```
local metapoint = {}
local function addtopath (t1, t2, type)
  t1 = t1.type == "path" and t1 or {t1}
  t2 = t2.type == "path" and t2 or {t2}
  local path = {type = "path"}
  setmetatable(path, metapoint)
  for _, p in ipairs(t1) do
    table.insert(path, copy_table(p))
  end
  local p = copy_table(table.remove(t2, 1))
  p.type = type
  table.insert(path, copy_table(p))
end
```

---

\* Things are a bit more complicated: the minus sign has precedence over the concatenation operator, so that given `a .. b - c`, first the path with `b` and `c` is constructed, then `a` is added. Also, `..` is right associative, so that `a .. b .. c` also creates the `b-to-c` path first. There is nothing wrong with that, except that we can't make do with a naive implementation where paths are expected only as the left operand. We wouldn't do that anyway since we want to be able to merge two already constructed paths into one.

```
for n, p in ipairs(t2) do
  table.insert(path, copy_table(p))
end
return path
end
function metapoint.__concat (t1, t2)
  return addtopath(t1, t2)
end
function metapoint.__sub (t1, t2)
  return addtopath(t1, t2, "control")
end
function metapoint.__add (t1, t2)
  return addtopath(t1, t2, "move")
end
```

The main function is `addtopath`: it creates a table with type `path`, and adds all the points in the two tables it connects by simply looping over all the entries (if one of the tables is a point, it is put into a table so we can use `ipairs()` on it). Special care is taken for the first point of the second path, which is the one concerned with the operator at hand; its type is set to the third argument. With `..`, which calls `__concat`, there is no third argument, hence the point is a type-less default point (more precisely, `nil` is assigned to `type`, which is equivalent to doing nothing). On the other hand, `__sub` and `__add` call `__concat` with the associated types. We systematically copy tables (representing points), so a path doesn't depend on the points it is defined with; if the latter are modified, the path isn't.

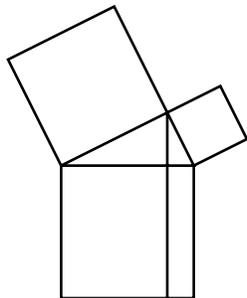
We can no longer declare points as simple two-element tables, because we must set `metapoint` as their metatable. So we'll use a function:

```
function point(x, y)
  local t = {x, y}
  setmetatable(t, metapoint)
  return t
end
```

Here we go: let's redraw Pythagoras' theorem, properly this time!

```
local a, b, c = point(20, 50), point(60, 70),
               point(70, 50)
local ab1, ab2 = point(0, 90), point(40, 110)
local bc1, bc2 = point(80, 80), point(90, 60)
local ca1, ca2 = point(70, 0), point(20, 0)
local triangle = a .. b .. c .. a
local ab_sq = a .. ab1 .. ab2 .. b
local bc_sq = b .. bc1 .. bc2 .. c
local ca_sq = c .. ca1 .. ca2 .. a
draw(triangle + ab_sq + bc_sq + ca_sq
     + b .. point(b[1], ca1[2]))
```

And the output follows:



We could easily rewrite our `scale()` function to work on paths instead of points. But wouldn't it be nice if we could write `path * 2` or `path * {2,3}` to mean, respectively, `scale(path, 2, 2)` and `scale(path, 2, 3)`? The reader probably can guess the answer: of course we can! But wait a minute; `path` is assumed to be a table with a proper metatable to behave correctly with an operator like multiplication. But that is obviously not the case for `{2, 3}`, an anonymous table without a metatable, let alone the number 2, which isn't even a table! As for the last interrogation (well, exclamation), all types can have metatables in Lua, although only tables can be assigned metatables outside the C API.\* But that is no trouble: given two operands around an operator, it suffices that one has the right metatable for the operation to occur; so if paths have a metatable with the `__mul` entry, the shorthand to scaling will work. I leave it as an exercise to the reader. (*Hint*: Don't forget to check the type of the second argument.)

Another thing I'll mention only briefly here. I use a syntax like this:

```
local path = a .. b^{linewidth = 1}
             .. c^{color = {1, 0, 0}}
```

Meaning: draw a line from `a` to `b` with a line width of 1, then a line from `b` to `c` in red (the color model, RGB here, being automatically detected by the number of values in the `color` table). The metatable

---

\* One can also use the `debug` library, but as its name indicates, it is not designed for ordinary programming and shouldn't be used for that, at least not in code meant to be public.

magic involved here should be clear to the reader (using the `__pow` entry), although one must rewrite the `draw()` function to take into account the information thus attached to points. But there is a difficulty, not related to Lua but to PDF: such parameters as line width, color, etc., attach to the stroking statement, not to the elements of a path. In other words, if some lines and curves are stroked together, then they will share the same parameters. We could of course stroke them one by one, hence allowing different parameters for each, but then PDF wouldn't automatically join lines; this is illustrated below by a one-stroke drawing next to a two-stroke drawing.



So we have to mimic PDF styles of joining lines, and/or invent our own. That is doable (I have implemented it), but explaining it here would double the size of this paper.

## Conclusion

I hope to have convinced the reader, if not to switch at once to a Lua-based graphic interface, at least that the simple addition of Lua to  $\text{\TeX}$  (besides all the wonderful opening up that takes place in  $\text{\LuaTeX}$ ) is by itself a formidable move. Lua is easy and powerful at once; here it is put to use with graphics, but tables could also be used for index or bibliography generation, and more generally to store organized information. The key-value interface that is so often (re)implemented in the  $\text{\TeX}$  world is available almost immediately with tables, not to mention metatables for default values. And of course, most of the  $\text{\TeX}$  interface in Lua is organized in tables.

Finally, although that might not be obvious in this paper,  $\text{\LuaTeX}$  brings yet another fundamental change to  $\text{\TeX}$ : it has become good at math!

◇ Paul Isambert  
zappathustra (at) free dot fr

## E-books: Old wine in new bottles

Hans Hagen

### 1 Introduction

When Dave Walden asked me if  $\text{\TeX}$  (or  $\text{\ConTeXt}$ ) can generate ebooks we exchanged a bit of mail on the topic. Although I had promised myself never to fall into the trap of making examples for the sake of proving something I decided to pick up an experiment that I had been doing with a manual in progress and look into the HTML side of that story. After all, occasionally on the  $\text{\ConTeXt}$  list similar questions are asked, like “Can  $\text{\ConTeXt}$  produce HTML?”.

### 2 Nothing new

When you look at what nowadays is presented as an ebook document, there is not much new going on. Of course there are very advanced and interactive documents, using techniques only possible with recent hardware and programs, but the average ebook is pretty basic. This is no surprise. When you take a novel, apart from maybe a cover or an occasional special formatting of section titles, the typesetting of the content is pretty straightforward. In fact, given that formatters like  $\text{\TeX}$  have been around that can do such jobs without much intervention, it takes quite some effort to get that job done badly. It was a bit shocking to notice that on one of the first e-ink devices that became available the viewing was quite good, but the help document was just some word processor output turned into bad-looking PDF. The availability of proper hardware does not automatically trigger proper usage.

I can come up with several reasons why a novel published as an ebook does not look much more interesting and in many cases looks worse. First of all it has to be produced cheaply, because there is also a printed version and because the vendor of some devices also want to make money on it (or even lock you into their technology or shop). Then, it has to be rendered on various devices so the least sophisticated one sets the standard. As soon as it gets rendered, the resolution is much worse than what can be achieved in print, although nowadays I’ve seen publishers go for quick and dirty printing, especially for reprints.

Over a decade ago, we did some experiments with touch screen computers. They had a miserable battery life, a slow processor and not much memory, but the resolution was the same as on the now fashionable devices. They were quite suitable for reading but even in environments where that made sense (for

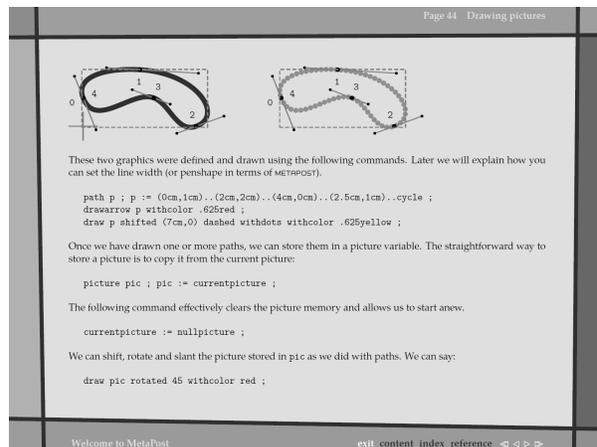


Figure 1: A page from the Metafun manual.

instance to replace carrying around huge manuals), such devices never took off. Nowadays we have wireless access and USB sticks and memory cards to move files around, which helps a lot. And getting a quality comparable to what can be done today was no big deal, at least from the formatting point of view.

If you look in the  $\text{\ConTeXt}$  distribution you will find several presentation styles that can serve as bases for an ebook style. Also some of the  $\text{\ConTeXt}$  manuals come with two versions: one for printing and one for viewing on the screen. A nice example is the Metafun manual (see figure 1) where each page has a different look.

It must be said that the (currently only black and white) devices that use electronic ink have a perceived resolution that is higher than their specifications, due to the semi-analog way the ‘ink’ behaves. In a similar fashion clever anti-aliasing can do wonders on LCD screens. On the other hand they are somewhat slow and a display refresh is not that convenient. Their liquid crystal counterparts are much faster but they can be tiresome to look at for a long time and reading a book on it sitting in the sun is a no-go. Eventually we will get there and I’m really looking forward to seeing the first device that will use a high resolution electrowetting CMYK display.<sup>1</sup> But no matter what device is used, formatting something for it is not the most complex task at hand.

### 3 Impact

Just as with phones and portable audio devices, the market for tablets and ebook-only devices is evolving rapidly. While writing this, at work I have one ebook device and one tablet. The ebook device is sort of obsolete because the e-ink screen has deteriorated

This article is reprinted from the Euro $\text{\TeX}$  2011 proceedings.

<sup>1</sup> <http://www.liquavista.com/files/LQV0905291LL5-15.pdf>

even without using it and it's just too slow to be used for reference manuals. The tablet is nice, but not that suitable for all circumstances: in the sun it is unreadable and at night the backlight is rather harsh. But, as I mentioned in the previous section, I expect this to change.

If we look at the investment, one needs good arguments to buy hardware that is seldom used and after a few years is obsolete. Imagine that a family of four has to buy an ebook device for each member. Add to that the cost of the books and you quickly can end up with a larger budget than for books. Now, imagine that you want to share a book with a friend: will you give him or her the device? It might be that you need a few more devices then. Of course there is also some data management needed: how many copies of a file are allowed to be made and do we need special programs for that? And if no copy can be made, do we end up swapping devices? It is hard to predict how the situation will be in a few years from now, but I'm sure that not everyone can afford this rapid upgrading and redundant device approach.

A friend of mine bought ebook devices for his children but they are back to paper books now because the devices were not kid-proof enough: you can sit on a book but not on an ebook reader.

The more general devices (pads) have similar problems. I was surprised to see that an iPad is a single user device. One can hide some options behind passwords but I'm not sure if parents want children to read their mail, change preferences, install any application they like, etc. This makes pads not that family friendly and suggests that such a personal device has to be bought for each member. In which case it suddenly becomes a real expensive adventure. So, unless the prices drop drastically, pads are not a valid large scale alternative for books yet.

It might sound like I'm not that willing to progress, but that's not true. For instance, I'm already an enthusiastic user of a media player infrastructure.<sup>2</sup> The software is public, pretty usable, and has no vendor lock-in. Now, it would make sense to get rid of traditional audio media then, but this is not true. I still buy CDs if only because I then can rip them to a proper lossless audio format (FLAC). The few FLACs that I bought via the Internet were from self-publishing performers. After the download I still got the CDs which was nice because the booklets are among the nicest that I've ever seen.

Of course it makes no sense to scan books for ebook devices so for that we depend on a publishing

network. I expect that at some point there will be proper tools for managing your own electronic books and in most cases a simple file server will do. And the most open device with a proper screen will become my favourite. Also, I would not be surprised if ten years from now, many authors will publish themselves in open formats and hopefully users will be honest enough to pay for it. I'm not too optimistic about the latter, if only because I observe that younger family members fetch everything possible from the Internet and don't bother about rights, so we definitely need to educate them. To some extent publishers of content deserve this behaviour because more than I like I find myself in situations where I've paid some 20 euro for a CD only to see that half a year later you can get it for half the price (sometimes it also happens with books).

Given that eventually the abovementioned problems and disadvantages will be dealt with, we can assume that ebooks are here and will stay forever. So let's move on to the next section and discuss their look and feel.

#### 4 Interactivity

The nice thing about a paper book is that it is content and interface at the same time. It is clear where it starts and ends and going from one page to another is well standardized. Putting a bookmark in it is easy as you can fall back on any scrap of paper lying around. While reading you know how far you came and how much there is to come. Just as a desktop on a desktop computer does not resemble the average desktop, an ebook is not a book. It is a device that can render content in either a given or more free-form way.

However, an electronic book needs an interface and this is also where at the moment it gets less interesting. Of course the Internet is a great place to wander around and a natural place to look for electronic content. But there are some arguments for buying them at a bookshop, one being that you see a lot of (potentially) new books, often organized in topics in one glance. It's a different way of selecting. I'm not arguing that the Internet is a worse place, but there is definitely a difference: more aggressive advertisements, unwanted profiling that can narrow what is presented to a few choices.

Would I enter a bookshop if on the display tables there were stacks of (current) ebook devices showing the latest greatest books? I can imagine that at some point we will have ebook devices that have screens that run from edge to edge and then we get back some of the appeal of book designs. It is that kind of future devices that we need to keep in mind when we design electronic documents, especially when after

<sup>2</sup> The software and hardware was developed by Slim-Devices and currently is available as Logitech Squeezeserver. Incidentally I can use the iPad as an advanced remote control.

some decades we want them to be as interesting as old books can be. Of course this is only true for documents that carry the look and feel of a certain time and place and many documents are thrown away. Most books have a short lifespan due to the quality of the paper and binding so we should not become too sentimental about the transition to another medium.

Once you're in the process of reading a book not much interfacing is needed. Simple gestures or touching indicated areas on the page are best. For more complex documents the navigation could be part of the design and no screen real estate has to be wasted by the device itself. Recently I visited a school-related exhibition and I was puzzled by the fact that on an electronic schoolboard so much space was wasted on colorful nonsense. Taking some 20% off each side of such a device brings down the effective resolution to 600 pixels so we end up with 10 pixels or less per character (shown at about 1 cm width). At the same exhibition there were a lot of compensation programs for dyslexia advertised, and there might be a relationship.

## 5 Formatting

So how important is the formatting? Do we prefer reflow on demand or is a more frozen design that suits the content and expresses the wish of the author more appropriate? In the first case HTML is a logical choice, and in the second one PDF makes sense. You design a nice HTML document but at some point the reflow gets in the way. And yes, you can reflow a PDF file but it's mostly a joke. Alternatively one can provide both which is rather trivial when the source code is encoded in a systematic way so that multiple output is a valid option. Again, this is not new and mostly a matter of a publisher's policy. It won't cost more to store in neutral formats and it has already been done cheaply for a long time.

Somewhat interfering in this matter is digital rights management. While it is rather customary to buy a book and let friends or family read the same book, it can get complicated when content is bound to one (or a few) devices. Not much sharing there, and in the worst case, no way to move your books to a better device. Each year in the Netherlands we have a book fair and bookshops give away a book specially written for the occasion. This year the book was also available as an ebook, but only via a special code that came with the book. I decided to give it a try and ended up installing a broken application, i.e. I could not get it to load the book from the Internet, and believe me, I have a decent machine and the professional PDF viewer software that was a prerequisite.

## 6 Using T<sub>E</sub>X

So, back to Dave's question: if ConT<sub>E</sub>Xt can generate ebooks in the ePub format. Equally interesting is the question if T<sub>E</sub>X can format an ePub file into a (say) PDF file. As with much office software, an ePub file is nothing more than a zip file with a special suffix in which several resources are combined. The layout of the archive is prescribed. However, by demanding that the content itself is in HTML and by providing a stylesheet to control the renderer, we don't automatically get properly tagged and organized content. When I first looked into ePub, I naively assumed that there was some well-defined structure in the content; turns out this is not the case.

Let's start by answering the second question. Yes, ConT<sub>E</sub>Xt can be used to convert an ePub file into a PDF file. The natural followup question is if it can be done automatically, and then some more nuance is needed: it depends. If you download the ePub for *A tale of two cities* from Charles Dickens from the Gutenberg Project website and look into a chapter you will see this:

```
<h1 id="pgepubid00000">A TALE OF TWO CITIES</h1>
<h2 id="pgepubid00001">A STORY OF THE FRENCH
  REVOLUTION</h2>
<p><br/></p>
<h2>By Charles Dickens</h2>
<p><br/>
<br/></p>
<hr/>
<p><br/>
<br/></p>
<h2 id="pgepubid00002">Contents</h2>
```

What follows is a table of contents formatted using HTML tables and after that

```
<h2 id="pgepubid00004">I. The Period</h2>
```

So, a level two header is used for the subtitle of the book as well as a regular chapter. I must admit that I had to go on the Internet to find this snippet as I wanted to check its location. On my disk I had a similar file from a year ago when I first looked into ePub. There I have:

```
<html xmlns="http://www.w3.org/1999/xhtml"
  xml:lang="en">
<head>
  <title>I | A Tale of Two Cities</title>
  . . . .
</head>
<body>
  <div class="body">
    <div class="chapter">
      <h3 class="chapter-title">I</h3>
      <h4 class="chapter-subtitle">The Period</h4>
```

I also wanted to make sure if the interesting combination of third and fourth level head usage was still there but it seems that there are several variants available. It is not my intention to criticize the coding, after all it is valid HTML and can be rendered as intended. Nevertheless, the first snippet definitely looks worse, as it uses breaks instead of CSS spacing directives and the second wins on obscurity due to the abuse of the head element.

These examples answer the question about formatting an arbitrary ePub file: “no”. We can of course map the tagging to ConT<sub>E</sub>Xt and get pretty good results but we do need to look at the coding.

As such books are rather predictable it makes sense to code them in a more generic way. That way generic stylesheets can be used to render the book directly in a viewer and generic ConT<sub>E</sub>Xt styles can be used to format it differently, e.g. as PDF.

Of course, if I were asked to set up a workflow for formatting ebooks, that would be relatively easy. For instance the Gutenberg books are available as raw text and that can be parsed to some intermediate format or (with MkIV) interpreted directly.

Making a style for a specific instance, like the Dickens book, is not that complex either. After all, the amount of encoding is rather minimal and special bits and pieces like a title page need special design anyway. The zipped file can be processed directly by ConT<sub>E</sub>Xt, but this is mostly just a convenience.

As ePub is just a wrapper, the next question is if ConT<sub>E</sub>Xt can produce some kind of HTML and the answer to that question is positive. Of course this only makes sense when the input is a T<sub>E</sub>X source, and we have argued before that when multiple output is needed the user might consider a different starting point. After all, ConT<sub>E</sub>Xt can deal with XML directly.

The main advantage of coding in T<sub>E</sub>X is that the source remains readable and for some documents it's certainly more convenient, like manuals about T<sub>E</sub>X. In the reference manual ‘ConT<sub>E</sub>Xt Lua Documents’ (CLD) there are the following commands:

```
\setupbackend
  [export=yes]
\setupinteraction
  [title=Context Lua Documents,
  subtitle=preliminary version,
  author=Hans Hagen]
```

At the cost of at most 10% extra runtime an XML export is generated in addition to the regular PDF file. Given that you have a structured T<sub>E</sub>X source the exported file will have a decent structure as well and you can therefore transform the file into something else, for instance HTML. But, as we already have a good-looking PDF file, the only reason

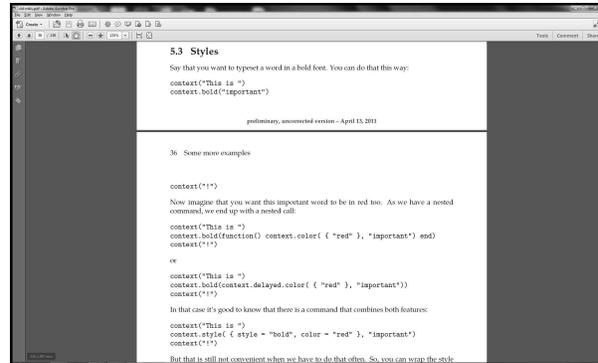


Figure 2: A page from the CLD manual in PDF.

to have HTML as well is for reflowing. Of course wrapping up the HTML into an ePub structure is not that hard. We can probably even get away from wrapping because we have a single self-contained file.

The `\setupbackend` command used in the CLD manual has a few more options:

```
\setupbackend
  [export=cld-mkiv-export.xml,
  xhtml=cld-mkiv-export.xhtml,
  css={cld-mkiv-export.css,mathml.css}]
```

We explicitly name the export file and in addition specify a stylesheet and an alternative XHTML file. If you can live without hyperlinks the XML file combined with the cascading style sheet will do a decent job of controlling the formatting.

In the CLD manual chapters are coded like this:

```
\startchapter[title=A bit of Lua]
\startsection[title=The language]
```

The XML output of this

```
<division detail='bodypart'>
<section detail='chapter' location='aut:3'>
  <sectionnumber>1</sectionnumber>
  <sectiontitle>A bit of Lua</sectiontitle>
  <sectioncontent>
    <section detail='section'>
      <sectionnumber>1.1</sectionnumber>
      <sectiontitle>The language</sectiontitle>
      <sectioncontent>
```

The HTML version has some extra elements:

```
<xhtml:a name="aut_3">
  <section location="aut:3" detail="chapter">
```

The table of contents and cross references have `xhtml:a` elements too but with the `href` attribute. It's interesting to search the web for ways to avoid this, but so far no standardized solution for mapping XML elements onto hyperlinks has been agreed upon. In fact, getting the CSS mapping done was not that much work but arriving at the conclusion that (in 2011) these links could only be done in a

robust way using HTML tags took more time. (In this example we see the reference `aut:3` turned into `aut_1`. This is done because some browsers like to interpret this colon as a url.) Apart from this the CSS has enough on board to map the export onto something presentable. For instance:

```
sectioncontent {
  display: block ;
  margin-top: 1em ;
  margin-bottom: 1em ; }
section[detail=chapter], section[detail=title] {
  margin-top: 3em ;
  margin-bottom: 2em ; }
section[detail=chapter]>sectionnumber {
  display: inline-block ;
  margin-right: 1em ;
  font-size: 3em ;
  font-weight: bold ; }
```

As always, dealing with verbatim is somewhat special. The following code does the trick:

```
verbatimblock {
  background-color: #9999FF ;
  display: block ;
  padding: 1em ;
  margin-bottom: 1em ;
  margin-top: 1em ;
  font-family: "Lucida Console",
    "DejaVu Sans Mono", monospace ; }
verbatimline {
  display: block ;
  white-space: pre-wrap ; }
verbatim {
  white-space: pre-wrap ;
  color: #666600 ;
  font-family: "Lucida Console",
    "DejaVu Sans Mono", monospace ; }
```

The spacing before the first and after the last one differs from the spacing between lines, so we need some extra directives:

```
verbatimlines+verbatimlines {
  display: block ;
  margin-top: 1em ; }
```

This will format code like the following with a bluish background and inline verbatim with its complement:

```
<verbatimblock detail='typing'>
  <verbatimlines>
    <verbatimline>function sum(a,b)</verbatimline>
    <verbatimline> print(a, b, a+b)</verbatimline>
    <verbatimline>end</verbatimline>
  </verbatimlines>
</verbatimblock>
```

The hyperlinks need some attention. We need to make sure that only the links and not the anchors get special formatting. After some experimenting I arrived at this:

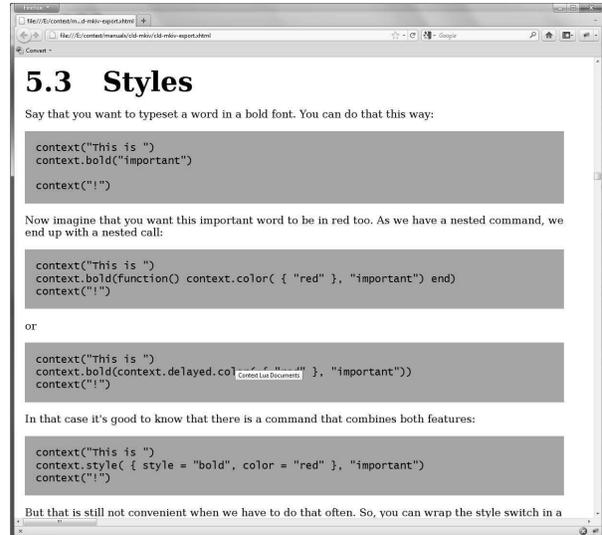


Figure 3: A page from the CLD manual in HTML.

```
a[href] {
  text-decoration: none ;
  color: inherit ; }
a[href]:hover {
  color: #770000 ;
  text-decoration: underline ; }
```

Tables are relatively easy to control. We have `tabulate` (nicer for text) and `natural tables` (similar to the HTML model). Both get mapped into HTML tables with CSS directives. There is some detail available so we see things like this:

```
tablecell[align=flushleft] {
  display: table-cell ;
  text-align: left ;
  padding: .1em ; }
```

It is not hard to support more variants or detail in the export but that will probably only happen when I find a good reason (a project), have some personal need, or when a user asks for it. For instance images will need some special attention (conversion, etc.). Also, because we use MetaPost all over the place that needs special care as well, but a regular (novel-like) ebook will not have such resources.

As an extra, a template file is generated that mentions all elements used, like this:

```
section[detail=summary] {
  display: block ; }
```

with the inline and display properties already filled in. That way I could see that I still had to add a couple of directives to the final CSS file. It also became clear that in the CLD manual some math is used that gets tagged as MathML, so that needs to be covered

as well.<sup>3</sup> Here we need to make some decisions as we export Unicode and need to consider support for less sophisticated fonts. On the other hand, the W3C consortium has published CSS for this purpose so we can use these as a starting point. It might be that eventually more tuning will be delegated to the XHTML variant. This is not much extra work as we have the (then intermediate) XML tree available. Thinking of it, we could eventually end up with some kind of CSS support in ConT<sub>E</sub>Xt itself.

It will take some experimenting and feedback from users to get the export right, especially to provide a convenient way to produce so-called ePub files directly. There is already some support for this container format. If you have enabled XHTML export, you can produce an ePub archive afterwards with:

```
mtxrun --script epub yourfile
```

For testing the results, open source programs like `calibre` are quite useful. It will probably take a while to figure out to what extent we need to support formats like ePub, if only because such formats are adapted on a regular basis.

## 7 The future

It is hard to predict the future. I can imagine that given the user interface that has evolved over ages paper books will not disappear soon. Probably there will be a distinction between read-once and throw-away books and those that you carry with you your whole life as visible proof of that life. I can also imagine that (if only for environmental reasons) ebooks (hopefully with stable devices) will dominate. In that case traditional bookshops will disappear and with them the need for publishers that supply them. Self-publishing will then be most interesting for authors and maybe some of them (or their helpful friends) will be charmed by T<sub>E</sub>X and tinkering with the layout using the macro language. I can also imagine that at some point new media (and I don't consider an ebook a new medium) will dominate. And how about science fiction becoming true: downloading stories and information directly into our brains.

It reminds me of something I need to do some day soon: get rid of old journals that I planned to read but never will. I would love to keep them electronically but it is quite unlikely that they are available and if so, it's unlikely that I want to pay for them again. This is typically an area where I'd consider using an ebook device, even if it's suboptimal. On the other hand, I don't consider dropping my

newspaper subscription yet as I don't see a replacement for the regular coffeestop at the table where it sits and where we discuss the latest news.

The nice thing about an analogue camera is that the image carrier has been standardized and you can buy batteries everywhere. Compare this with their digital cousins: all have different batteries, there are all kinds of memory cards, and only recently has some standardization in lenses shown up. There is a wide range of resolutions and aspect ratios. Other examples of standardization are nuts and bolts used in cars, although it took some time for the metric system to catch on. Books have different dimensions but it's not hard to deal with that property. Where desktop hardware is rather uniform everything portable is different. For some brands you need a special toolkit with every new device. Batteries cannot be exchanged and there are quite some data carriers. On the other hand, we're dealing with software and if we want we can support data formats forever. The Microsoft operating systems have demonstrated that programs written years ago can still run on updates. In addition Linux demonstrates that users can take and keep control and create an independence from vendors. So, given that we can still read document sources and given that they are well structured, we can create history-proof solutions. I don't expect that the traditional publishers will play a big role in this if only because of their short term agendas and because changing ownerships works against long term views. And programs like T<sub>E</sub>X have already demonstrated having a long life span, although it must be said that in today's rapid upgrade cycles it takes some courage to stay with it and its descendants. But downward compatibility is high on the agenda of its users and user groups which is good in the perspective of discussing stable ebooks.

Let's finish with an observation. Books often make a nice (birthday) present and finding one that suits is part of the gift. Currently a visible book has some advantages: when unwrapped it can be looked at and passed around. It also can be a topic of discussion and it has a visible personal touch. I'm not so sure if vouchers for an ebook have the same properties. It probably feels a bit like giving synthetic flowers. I don't know what percentage of books is given as presents but this aspect cannot be neglected. Anyway, I wonder when I will buy my first ebook and for who. Before that happens I'll probably have generated lots of them.

<sup>3</sup> Some more advanced MathML output will be available when the matrix-related core commands have been upgraded to MkIV and extended to suit today's needs.

◇ Hans Hagen  
<http://pragma-ade.com>

---

## iTeX — Document formatting in an ereader world

William Cheswick

### Abstract

TeX and other traditional text layout markup languages are predicated on the assumption that the final output format would be known to the nanometer. Extensive computation and clever algorithms let us optimize the presentation for a high standard of quality, designed by artists who are experts at document layout.

But ebooks are here, and the iPad sold millions of units in the first few months after its introduction. Book readers offer a new way to store and read documents, but they are a challenge to high quality text layout. Ebook users are accustomed to selecting reader orientation, typeface, and font size. We probably cannot run TeX over a document every time a reader shifts position in his chair.

iTeX is an experimental document bundle format and a free iPad application that present documents exactly as they were rendered by TeX. The bundle (a tar of a standardized directory structure) contains precomputed page images for portrait and landscape layouts, in standard and large type versions. I hope this may be a suitable standard to encourage similar applications on devices like the Nook or the many versions of the Kindle, included in the same bundle.

### 1 Introduction

In these days of author self-publishing, we must not forget the value of professionals. *Don Knuth* [6]

In March 2010 I served on three program committees, reading and grading dozens of papers. March is also when my iPad arrived, and it would have been a handy device for the job.

But the papers were all in PDF, formatted for either US or European standard page sizes. Devices like the iPad and Kindle are not large enough to show an entire traditional page, unless the image is shrunk to uselessness or the user scans the text using the device as a sort of large magnifying glass. And the PDF layout comes in a single version, with no thought of portrait *vs.* landscape viewing.

Just-in-time (JIT) page layouts are common now, including HTML, EPUB, and other “reflowable” display engines. These page layouts are fine for most web pages and novels, but the authors of technical works usually need fine control, or at least final editorial veto, over the final appearance of the document.

William Cheswick

This is certainly not possible if the user is selecting his favorite typeface and font size while reading the document. And JIT readers do not render mathematics well, often serving up images of formulae in non-matching typefaces.

There are efforts to bring higher-quality layouts to JIT formats. For example, the EPUB 2.1 Working Group Charter seeks native support for mathematics [3]. They are far behind TeX, which is now over thirty years old and has a large corpus and user community.

Good text layout is hard, a job for professionals, and users shouldn’t be making these decisions. The designer knows the size and resolution of the device, and which fonts are appropriate. The designer’s decision is likely to be different for portrait and landscape viewing modes. And the authors should be able to see the document in its final form, before it is published.

It *is* reasonable for a user to require larger type sizes to remedy vision impairment and suboptimal reading milieu. Book designers are familiar with large type versions of books and newspapers, and with iTeX can provide that option.

The iTeX iPad application and bundle format presented here are an attempt to bring these ideas together into a reader and a simple document bundle format. The bundle is a standard tar file of a directory containing several files with specific names and formats. (See Appendix A for layout details.) The application can display portrait and landscape orientations of normal and large type versions of a document, formatted specifically for the iPad. It can display `.itex` documents retrieved from the web, or copied into the iPad from iTunes. There is no iPhone version envisioned: that form factor is simply too small for most technical documents. (The additions to the application would be minor, and I am willing to be persuaded that such support would be useful.) I hope the bundle format will be adopted and used for reading software on other devices.

The rest of the paper is laid out as follows: §2 discusses related work. The details of the iTeX implementation are covered in §3. A general discussion appears in §4 and future work (of which there is a lot) is in §5. Finally, §6 concludes, and some useful details are found in the Appendices.

### 2 Related work

There are a growing number of reader applications for the iPad. Most support reflowable text from EPUB sources, and display PDF files. Unlike iTeX, they have large, expert development groups and are quite full-featured.

Apple’s *iBooks* and the Kindle application are two major readers. *iBooks* will display purchases from the iTunes store (some are free), as well as PDF and EPUB texts. The latter are now available from their *Pages* program, as well. PDFs are displayed as a shrunken page and one can use a two-finger gesture to zoom in. PDFs formatted for the iPad portrait size are displayed correctly. Double-taps zoom in a bit. Landscape display shows the portrait image, letterboxed, *i.e.* it is much smaller. *iBooks* uses page numbers, with different document lengths for the two orientations, which is also what *iTeX* does. It is not entirely clear to me how the page correspondence between portrait and landscape is accomplished, but it comes out about right.

The Kindle normally displays books purchased from Amazon’s store. There is also a conversion service for downloading PDF documents for a small charge. PDFs can be loaded into the Kindle through a USB device without charge. Page position on the Kindle is shown as a percentage of progress through the document. Rotation of the iPad preserves the first word on the page.

The *Stanza* application offers access to a number of booksellers as well as open sources like Project Gutenberg. The portrait/landscape positions correlate well. Page numbers are confusing, and progress percentage is also shown.

*arXiver* offers access to the arXiv of scientific preprints [4]. It displays PDF versions of the documents, computed for page-size display and reduced on the iPad.

Kaveh Bazargan anticipated this *iTeX* work by a year [1]. At River Valley Technologies they process the original “author”  $\TeX$  into XML, and then have an engine that automatically processes the XML into “slave”  $\TeX$  and then PDF. These latter steps are amenable to implementation in an iPhone, and they demonstrate a nice result. The images are rapidly recomputed on-the-fly for orientation changes. This is quite a *tour de force*, but it seems like a lot of extra work, and the translation to a simpler  $\TeX$  may induce errors. This seems a bit unwieldy to me, but perhaps they are on the right track.

The present work should not be confused with Don Knuth’s vaporware, *i(whistle)TeX* [7].

### 3 Implementation

#### 3.1 Where to process the $\TeX$ ?

*iTeX*’s goals raise some interesting design decisions. In particular, where and when should  $\TeX$  be run, and what exactly should the reader application do?

One possibility is to feed the document to the iPad and implement  $(\mathbb{A})\TeX$  in its full glory there.

That certainly makes all the information available to the reader, and the document is compact. But the  $\TeX$  system is large, and the iPad is meant to be a simple device, mostly dealing with user interface issues. The CPU is fast, but not that fast, and there are battery life concerns when using a lot of CPU power.

Also, the author doesn’t get to see the final result without an iPad. And document generation often involves a number of other utilities. In our firewalls book [2] we use *gv*, *bibtex*, *grap*, *pic*, *sed*, *grep*, *awk*, *convert*, and about a dozen scripts for fixing the glossary, acronyms, generating the index, etc. These would not be available on the iPad: it is not a software development machine.

A second approach is to download the DVI files to the reader, and perform the rendering step there. This seems more promising: the layout work has been done, the DVI file is easily compressed, and all that is left is the font generation and placement. One could even use the resident Apple fonts, which would add a consistent feel to the application with Apple’s efforts towards a uniform user experience.

It does mean that special fonts will have to be computed and loaded. In normal  $\TeX$  processing, this may involve Metafont calls and libraries of pre-computed font information. This seems like a lot to have to load into the iPad, and a source of reader disappointment if the proper fonts are missing.

It also means that files needed by DVI `\special` commands have to be downloaded somehow as well, and the iPad/DVI processor has to keep up with new specials.

For these reasons I chose to generate the images at the source, where the user is already accustomed to formatting his documents. The interface is much cleaner, which simplifies the application considerably. (It also would make redaction foolproof, since there is no hidden text.) But this approach does discard information that a good reader could use, so it will have to be augmented in the future (see §5).

#### 3.2 Page image production

The production of quality page images from DVI would seem easy, but I ran into problems. When open source’s thousand flowers bloom, which one do you pick?

I started with *dvi2bitmap*, but that turned out to be fairly rudimentary, creating output reminiscent of DECwriter printers. Greyscale and anti-aliasing were not available, nor was processing of most important `\special` commands.

*Dvips* post-processed with *gv* to generate PNG files worked fairly well, and could have been made

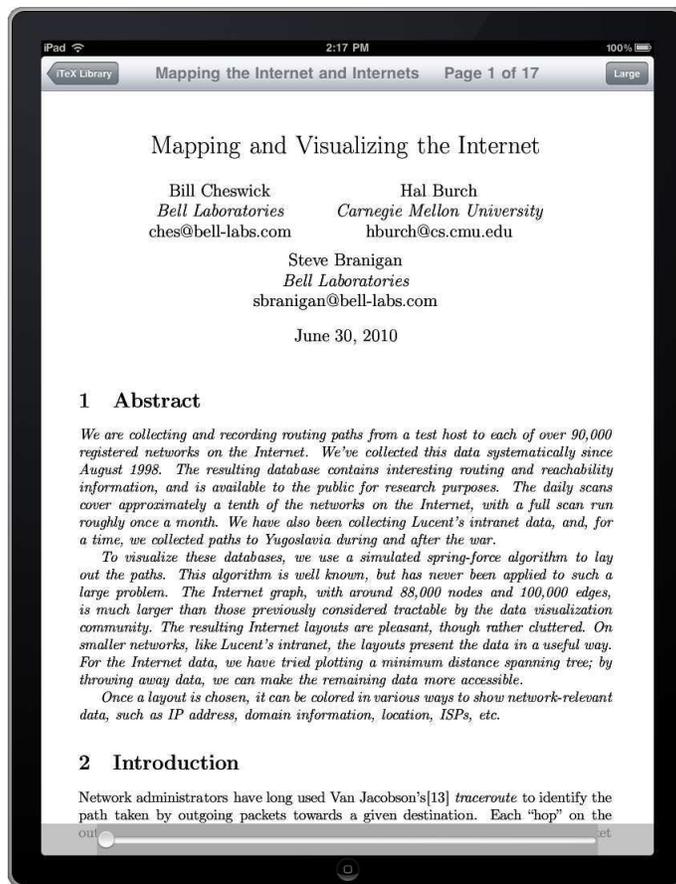


Figure 1: Portrait view of a document page, with navigation bar and page selection slider.

to work. But there was a lot of mechanism there, and it was slow and fairly unwieldy. It does support all the PostScript-related `\special` commands, and may ultimately be the process of choice.

Finally I found *dvipng*, and it works well, creating the anti-aliased characters that rival Apple's own on the iPad. (The *dvi2bitmap man* page should have a pointer to *dvipng*.) *Dvipng* does have some bugs — one of my PostScript images causes it to crash, for example — but it will be the basis for a *dviitex* (see below). The program *genitex* is a crude prototype script that generates an iTeX bundle from many L<sup>A</sup>T<sub>E</sub>X files. See <http://www.cheswick.com/itex> for details.

### 3.3 The iPad application

The iPad application is a pretty standard reader. It has a library screen (currently unsortable) and traditional navigation bars to change screens. Pages are turned with a flick of the finger, and a tap on the document display shows or hides the page slider at the bottom and navigation bar at the top. A navigation bar entry selects different versions, such as

large type or others that may be supplied. Figures 1 and 2 display portrait and landscape displays of a version of this paper.

The app displays bundles downloaded into the app or by a web browser. Any file with an extension of `.itex` or `.iTeX` will be scanned and installed in the iTeX library on the iPad. Malformed bundles are skipped, with an error message.

The iTeX library is displayed when the app is started. Bundles may be downloaded, replaced, or deleted with the usual iPad gestures. iTeX documents are keyed with the title and author: identical title/author pairs are deemed to be the same document, and only one is stored in the library.

## 4 Discussion

iTeX requires the generation of a fair number of final documents. In the simplest case, a technical document is likely to be generated for 8.5x11 inch format (or A4), plus two versions for the iPad. If a large type version is generated, that is two more L<sup>A</sup>T<sub>E</sub>X runs. As more devices are supported, perhaps

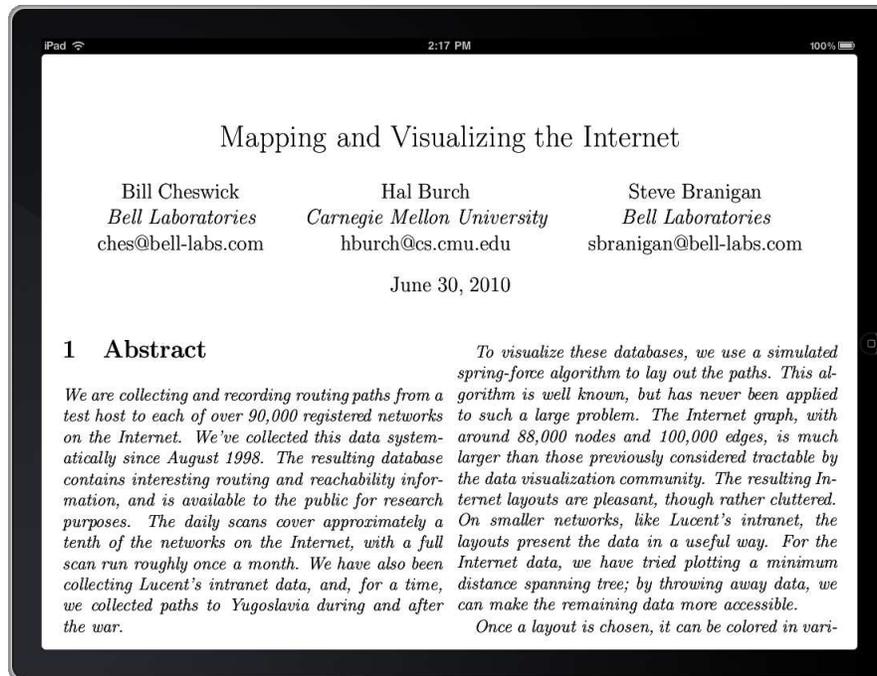


Figure 2: Landscape view of a document page, with large text size selected.

including a “retina” version of the iPad someday, the iTeX bundle will grow to considerable size.

I don’t consider this size to be a real problem. Download times will increase, though probably not significantly for WiFi and hardwired downloads. The iPad app uses only the relevant versions and discards the rest after download. If size becomes a problem, separate iTeX bundles could be offered for differing devices.

More important is the problem of generating multiple documents of varying format from the same source. There is trouble enough chasing down all the TeX nits (like overfull hboxes) when the final version of a document is ready to go. iTeX offers many more opportunities for these, since one is now generating a page-size version of the document, plus four versions just for the iPad. For really serious typographers, this requires human judgement for each version, and it is possible that the problem is over-constrained, that there is no single L<sup>A</sup>TeX source file that will look good in all versions. The page designer may have to cut TeX a little extra slack in the layout settings of some of the versions for author usability and automation schemes. It remains to be seen if the automated results work well enough.

Page numbers present a problem. In the current iTeX implementation, a document can have four different page lengths. When one changes orientation or type size, what is the appropriate page number to

switch to? At present iTeX estimates based on the percentage distance through the document, which seems to work well enough for the reader.

But how do we convey document position to someone else? “There is a typo at the bottom of page 4” doesn’t work unless we refer to the device, device version, orientation, and text size version as well.

It would be nice to have a url format that gives word- or at least paragraph-position. TeX could collect this information by marking each word and passing the information on in the DVI file for processing.

iTeX has been available in the Apple app store since February 2011. There have been quite a number of downloads from non-English speaking countries. This raises the point that iTeX documents and the app have almost no linkage to the English language. Other than file names (UTF-8 is processed) and error messages, and the order in which pages are presented, iTeX has no need to know what language is presented, since they are just a series of images. It might be useful to supply an option in the iTeX bundle indicating that the document creator wishes the document to be displayed starting at the “right” end.

Apple does provide an easy means for internationalizing text in the error messages, and I should implement that.

## 5 Future work

### 5.1 Improving the application

The iTeX iPad application is an implementation of a simple reader, the product of my first Objective C project. While it has the basic features of a reader and is quite usable as such, it could use a fair amount of polish.

At present, the iTeX bundle stores a version of the document as a sequence of PNG images in a directory. These files could be replaced by a single PDF for each version and orientation. The next version will support this. More importantly, there is a lot of semantic data available during TeX processing that is or could be captured in the DVI file and provided to the application in a separate file, one per page, regardless of PDF/PNG choice.

For example, one cannot zoom into the pre-computed text in iTeX. If the designer did his job correctly, it shouldn't be necessary, and large type versions should help those with vision impairments. But images and graphs certainly should be selectable, and viewed at arbitrary zoom levels. I expect to include EPS versions of these, and let the reader access them.

Other features are needed: words, footnotes, bibliographic references, glossary entries, and urls should be active and tap-able. Selected text should be paste-able elsewhere on the iPad. No search function is available at this time. And it would be nice to have a note-taking and commentary feature, something that would have been very handy while I was reviewing papers.

The page images that iPad handles contain none of this information — it will have to be supplied separately. It will contain a list of rectangles for each word on the page and properties for each rectangle, and should be easily processed by the application each time the screen is touched. The precise format and contents of this information needs to be determined. I plan to modify *dvipng* into a new program, *dviitex*, to provide this data.

It might be useful to compute even more versions of a document, perhaps with extra-large font. This adds to the bundle length and generation time, but might be worthwhile for some reason. iTeX places no specific limit on the number and names of versions available, although the navigation bar software will eventually have too much to handle nicely.

### 5.2 New devices

Since the iPhone 4 doubled its screen resolution over previous versions (the “retina” version), the same is likely to happen on some future version of the iPad.

Document designers will care — the current iPad has only 132 DPI and hence needs low-resolution fonts like Lucida Bright. The iPad application will need modifications to deal with this.

If the iTeX idea is successful, there ought to be applications for the Kindle, and other reading devices. These devices have other challenges for the document style designers, such as monochrome-only displays with limited greyscale support.

### 5.3 TeX library access

Adrian Petrescu has been thinking along similar lines, for the Kindle [8]. But he has suggested making a browser for arXiv, with a server that translates the arXiv text and downloads it into the device. This is a terrific idea, and I am planning to implement an arXiv browser window in the iTeX application and put up such a translator (with appropriate security, of course). I hope our two efforts can merge. The automated translator *genitex* mentioned in §3 could use some help from people with good L<sup>A</sup>TeX hacking skills.

I understand that Project Gutenberg may also be converting to L<sup>A</sup>TeX. If so, a similar browser would be appropriate for their library as well.

## 6 Conclusion

iTeX was a useful project to help get me up to speed on programming iPhones and the iPad. After several months of intensive programming, I have a fair grasp of the Objective C language and many of the Apple libraries and UI design ideas. It was the first radically different language I have learned in twenty-five years.

But I think iTeX may fill a need, and perhaps will form the basis for a useful documentation publication format. I am prepared to work hard on supporting that effort, if there is sufficient interest. If not, I have other applications in mind, and this project was fun.

### Acknowledgements

Without Dave Kormann's help, I never would have made a dent in Objective C: the documentation just isn't that clear.

Adrian Petrescu mentioned ideas about the Kindle and especially the arXiv access. I hope we can work together on this.

My thanks also to Barbara Beeton, Boris Veytsman, Will Robertson, Alexis Shaw, and to several other helpful folks who offered suggestions and comments at the TeX Users Group Meeting in San Francisco in June 2010. Barbara Beeton and Brian Clapper provided editing help as well.

## 7 Availability

The iTeX app is available for free in the iTunes store. Additional information, sample iTeX bundles, and source code are available at <http://www.cheswick.com/itex>.

## References

- [1] Kaveh Bazargan. T<sub>E</sub>X as an eBook reader. *TUGboat* 30(2): 272–273, 2009.
- [2] William Cheswick, Steve Bellovin, and Avi Rubin. *Firewalls and Internet Security; Repelling the Wily Hacker*, second edition. Addison Wesley Longman, 2003.
- [3] EPUB 2.1 Working Group Charter-DRAFT 0.11, (7 May 2010), item 9.
- [4] Paul Ginsparg. First steps towards electronic research communication. *Computers in Physics*, 8(4): 390–396, 1994.
- [5] Brian Kernighan and Rob Pike. *The Unix Programming Environment*, Addison Wesley, 1984.
- [6] Donald E. Knuth, Tracy L. Larrabee, and Paul M. Roberts. *Mathematical Writing*, p. 14. Mathematical Association of America, Washington, D.C., 1989.
- [7] Donald E. Knuth. An Earthshaking Announcement. *TUGboat* 31(2): 121–124, 2010.
- [8] Adrian Petrescu. Personal communication.

◇ William Cheswick  
 AT&T Shannon Lab  
 Florham Park, NJ, USA  
 ches (at) research dot att dot com

## A iTeX bundle format

An iTeX bundle is a file with the extension `.itex` that is actually an uncompressed POSIX tar file. (These are quite easy to unpack on the iPad.)

The root directory contains:

<b>Title</b>	A file whose first line contains the document title, in UTF-8. The rest of the file is ignored.
<b>Author</b>	Similarly the authors on the first line, UTF-8.
<b>Options</b>	An optional one-line file containing the string <code>r1</code> if the document is to be presented starting from the right-most page.
<i>dir</i>	One or more directories containing device-dependent versions of the document.

Each directory *dir* is named for the document type it contains. The names may be one of:

<b>iPad</b>	original iPad
<b>iPad3</b>	“retina” version of iPad
<b>iPhone</b>	original iPhone
<b>iPhone4</b>	“retina” version of the iPhone
<b>Kindle</b>	original Kindle
<b>Kindle2</b>	newer Kindle
<b>xoom</b>	Motorola android Xoom

At this time, the iTeX app only installs and uses the iPad directory. Non-Apple names are tentative, to be chosen based on display details for the various devices.

Each of these device directories contains a sub-directory for each version of the document. The version names are displayed on the top of the document display, and may have any short name. By convention:

<b>normal</b>	the default document version
<b>large</b>	the large-type version.

Neither the names nor the number of versions is enforced, but the app may have trouble displaying them usefully if there are too many, or they are too long. All these names honor UTF-8 format.

Each version contains two directories, `p` and `l` for the portrait and landscape orientations of the version of the document. At present, each of these directories contains a series of page images as PNG files of the form `%04d.png`, starting with `0001.png` as generated by *dvipng*. In future versions of the iTeX app there will also be page-related data in files named `0001.dat`, `0002.dat`, etc. in these directories. The multiple PNG files will be replaceable by a single PDF file. There will also be EPS files for images in the document, and perhaps other related files.

---

## Math alphabets and the `mathalfa` package

Michael Sharpe

### Abstract

This is both a survey of existing math alphabets and a brief description of a unified method of calling them via the package `mathalfa`.

### 1 Introduction

For the purposes of this article, a math alphabet is one normally selected in math mode using the  $\LaTeX$  macros `\mathcal`, `\mathscr`, `\mathbb`, `\mathfrak`, or their bold counterparts `\mathbcal`, `\mathbscr`, `\mathbbb` and `\mathbfrak`.

Regular and bold weights of a math calligraphic font are built into Computer Modern, occupying the upper case letter slots in the `cm[b]sy` family. The *fraktur* and *blackboard bold* alphabets were (I believe) introduced with the AMS fonts, and the `mathrsfs` package introduced the term `\mathscr` in order to provide a script font with more elaborate shapes, as is customary in a number of areas in math and physics, in addition to an ordinary calligraphic alphabet. The Unicode specification lists the fonts under *Mathematical Alphanumeric Symbols* (U1D400–U1D7FF), though a number of special, commonly used glyphs fall under the heading *Letterlike Symbols* (U2100–U214F). The Unicode names for the alphabets are:

#### MATHEMATICAL SCRIPT:

a.k.a. script, swash, calligraphic.

#### MATHEMATICAL DOUBLE-STRUCK:

a.k.a. double-struck, blackboard bold, openface.

#### MATHEMATICAL FRAKTUR:

a.k.a. *fraktur*, *blackletter*, *gothic*.

### 2 `Mathalfa`

The `mathalfa` package in most cases bypasses the usual font-loading mechanisms for these math alphabets and substitutes its own, allowing it to use common terminology and, in all cases, allow arbitrary scaling. (Many  $\LaTeX$  packages that load fonts have not been modified since the days when Metafont was the predominant font format, and it was desirable to restrict the set of sizes at which the bitmaps were generated, thus limiting the possibility of fine scaling.) In a number of cases, the original math alphabet fonts were never set up with the metrics appropriate for math mode, leading to awkward placement of accents and subscripts, and inappropriate spacing. This package corrects such deficiencies by supplying virtual fonts with my preferences for those metrics following, by and large, the appearance of `mtpro2` (*MathTime Pro II*).

The line

```
\usepackage[showoptions]{mathalfa}
```

throws an error and shows all alphabet names understood by the package. As an example, to load Zapf Chancery scaled up by 15% as the output of `\mathcal` and Fourier Double-Struck scaled down by 4% as the output of `\mathbb`, you enter

```
\usepackage[cal=zapfc,calscaled=1.15,
             bb=fourier,bbscaled=.96]
{mathalfa}
```

after loading other math packages. You may also set as options

```
frak=...[,frakscaled=...],
scr=...[,scrscaled=...],
```

to enable the macros

```
\mathfrak
\mathbfrak % if there is a bold fraktur
\mathscr
\mathbscr  % if there is a bold script
```

The names available for the alphabets are listed separately below by type. Any *Mathematical Script* name may serve as a target for both `cal` and `scr`. In all cases, if a bold version is available, then the corresponding bold variant is also defined.

The `mathalfa` package does not provide the PostScript fonts required for activating all its options. The metrics and virtual fonts are publicly available, but are useless without the `.pfb` files which you must acquire. See the `mathalfa` documentation for detailed descriptions of sources.

#### 2.1 BOONDOX

The `boondox` package is a reworking of the STIX calligraphic, *fraktur* and double-struck alphabets with virtual fonts and metrics suitable for math mode. (In the USA, *the boondocks* and *the sticks* are essentially synonymous.) When the  $\LaTeX$  support files for the STIX fonts are made public, the `boondox` package will most likely become obsolete except to those who may prefer its metrics.

#### 2.2 ESSTIX

The other relatively unknown font package here is `esstix`, an unfinished math font collection produced by Elsevier, never officially released and subsequently donated to the STIX consortium, serving as a precursor to their STIX font family. The ESSTIX collection is now under the same license as the STIX collection — the liberal SIL Open Font License, version 1.1. Though STIX regards the ESSTIX collection as deprecated, the math alphabets it contains have some unique elements which, in my opinion, should not be

allowed to become extinct. The ESSTIX fonts and support files are now available from CTAN and have become part of T<sub>E</sub>X Live.

The ESSTIX fonts in their original forms may be loaded via `mathalfa` using the option `esstix`, but there is now an updated version of the ESSTIX calligraphic font. The metrics are identical to the original fonts, but the font has been modified in several respects (using `FontForge`) so that it now validates properly following modifications to repair font outline points and the font `BlueScale` parameter.

In addition, a bold version has been created, following modifications to a small number of glyphs to prevent outline self-intersections. (The original was quite light, rather of book weight, and the bold is more correctly a demi-bold.) ESSTIX calligraphic and its update, dubbed `DutchCal`, are visually almost identical at regular weight, though the latter is hinted better and is the only one shown in the samples below.

### 3 Mathematical script

The following choices are available for `cal` and `scr`, listed according to general appearance.

UPRIGHT:

```
euler % euscript
mtc % MathTime Curly (commercial)
```

RESTRAINED:

```
cm % Computer Modern Math Italic (cmmi)
lucida % From Lucida New Math (commercial)
zapfc % Adobe Zapf Chancery (or URW clone)
mma % Mathematica fonts
```

EMBELLISHED:

```
mt % MathTime (commercial)
rsfso % based on rsfs, much less sloped
mathpi % Adobe Mathematical Pi (commercial)
esstix % ESSTIX-thirteen
dutchcal % modification of ESSTIX13
boondoxo % based on boondox, less sloped
```

HEAVILY SLOPED:

```
rsfs % Ralph Smith Formal Script
boondox % SCRIPT alphabet from stix fonts
```

Script font samples follow in the above order.

#### 3.1 Upright

`euler` (Euler script):

*A B C D E F G H I J K L M*  
*N O P Q R S T U V W X Y Z*

`euler` (Euler script-bold):

***A B C D E F G H I J K L M***  
***N O P Q R S T U V W X Y Z***

`mtc` (MathTime Pro 2 Curly script):

*A B C D E F G H I J K L M*  
*N O P Q R S T U V W X Y Z*

#### 3.2 Restrained

`cm` (CM calligraphic, `cmsy`):

*A B C D E F G H I J K L M*  
*N O P Q R S T U V W X Y Z*

`cm` (CM calligraphic-bold, `cmbsty`):

***A B C D E F G H I J K L M***  
***N O P Q R S T U V W X Y Z***

`zapfc` (Zapf Chancery):

*A B C D E F G H I J K L M*  
*N O P Q R S T U V W X Y Z*

`lucida` (Lucida calligraphic):

*A B C D E F G H I J K L M*  
*N O P Q R S T U V W X Y Z*

`lucida` (Lucida calligraphic-bold):

***A B C D E F G H I J K L M***  
***N O P Q R S T U V W X Y Z***

`mma` (Mathematica script):

*A B C D E F G H I J K L M*  
*N O P Q R S T U V W X Y Z*

`mma` (Mathematica script-bold):

***A B C D E F G H I J K L M***  
***N O P Q R S T U V W X Y Z***

#### 3.3 Embellished

`mt` (MathTime Pro 2 script):

*A B C D E F G H I J K L M*  
*N O P Q R S T U V W X Y Z*

`mt` (MathTime Pro 2 script-bold):

***A B C D E F G H I J K L M***  
***N O P Q R S T U V W X Y Z***

`mathpi` (Mathpi script):

*A B C D E F G H I J K L M*  
*N O P Q R S T U V W X Y Z*

`dutchcal` (DutchCal):

*A B C D E F G H I J K L M*  
*N O P Q R S T U V W X Y Z*

`dutchcal` (DutchCal-bold):

***A B C D E F G H I J K L M***  
***N O P Q R S T U V W X Y Z***

`rsfso` (RSFS oblique):

*A B C D E F G H I J K L M*  
*N O P Q R S T U V W X Y Z*

`boondoxo` (BOONDOX calligraphic oblique):

*A B C D E F G H I J K L M*  
*N O P Q R S T U V W X Y Z*

`boondoxo` (BOONDOX calligraphic oblique-bold):

***A B C D E F G H I J K L M***  
***N O P Q R S T U V W X Y Z***

### 3.4 Heavily sloped

`boondox` (BOONDOX calligraphic):

*A B C D E F G H I J K L M*  
*N O P Q R S T U V W X Y Z*

`boondox` (BOONDOX calligraphic-bold):

***A B C D E F G H I J K L M***  
***N O P Q R S T U V W X Y Z***

`rsfs` (RSFS standard):

*A B C D E F G H I J K L M*  
*N O P Q R S T U V W X Y Z*

### 3.5 Notes

- Lucida fonts generally need to be reduced in scale to match other math and text fonts.
- Zapf Chancery needs to be scaled up by 15% or so. This font is not ideally suited for use as a math alphabet due to the disparate heights and depths and the long tails on some glyphs. Use with care.
- The calligraphic fonts break down into four natural groups:
  - (i) the upright styled Euler and Curly;
  - (ii) the rather restrained CM, Lucida, Zapf Chancery, ESSTIX and `mma`;
  - (iii) the moderately sloped but more embellished Mathpi, MathTime (borderline case), `rsfs` and `boondox`;
  - (iv) the heavily sloped `rsfs` and the slightly less sloped `boondox`.

My preference, if not using `euler` or `lucida` for math, would be to set `\mathcal` to one from group (ii) and `\mathscr` to one from group (iii).

## 4 Mathematical double-struck

Double-struck font samples follow.

### 4.1 Normal weight

BLACKBOARD BOLD (WITH SERIFS):

`ams` (AMS bb):

**A B C D E F G H I J K L M**  
**N O P Q R S T U V W X Y Z**

`math` (MathTime Pro 2 Holey Roman):

**A B C D E F G H I J K L M**  
**N O P Q R S T U V W X Y Z**

`pazo` (Mathpazo bb):

**A B C D E F G H I J K L M**  
**N O P Q R S T U V W X Y Z**

BLACKBOARD BOLD (SANS SERIF):

`lucida` (Lucida bb):

**A B C D E F G H I J K L M**  
**N O P Q R S T U V W X Y Z**

`mathpi` (Mathpi bb):

**A B C D E F G H I J K L M**  
**N O P Q R S T U V W X Y Z**

`mt` (MathTime Pro 2 bb):

**A B C D E F G H I J K L M**  
**N O P Q R S T U V W X Y Z**

`mma` (Mathematica bb):

**A B C D E F G H I J K L M**  
**N O P Q R S T U V W X Y Z**

`fourier` (Fourier bb):

**A B C D E F G H I J K L M**  
**N O P Q R S T U V W X Y Z**

`esstix` (ESSTIX bb):

**A B C D E F G H I J K L M**  
**N O P Q R S T U V W X Y Z**

`boondox` (BOONDOX bb):

**A B C D E F G H I J K L M**  
**N O P Q R S T U V W X Y Z**

### 4.2 Bold weight

BLACKBOARD BOLD (WITH SERIFS):

`math` (MathTime Pro 2 Holey Roman-bold):

**A B C D E F G H I J K L M**  
**N O P Q R S T U V W X Y Z**

BLACKBOARD BOLD (SANS SERIF):

`mt` (MathTime Pro 2-bold):

**A B C D E F G H I J K L M**  
**N O P Q R S T U V W X Y Z**

`mma` (Mathematica bb-bold):

**A B C D E F G H I J K L M**  
**N O P Q R S T U V W X Y Z**

`boondox` (BOONDOX bb-bold):

**C D H N P Q R Z** (few glyphs available)

### 4.3 Notes

- Blackboard bold can look poor on the screen in many cases. Perhaps the thin parallel lines in the figures are a challenge to anti-aliasing mechanisms, at some resolutions. For example, here is the result of a screenshot from my 94 dpi LCD screen of a fragment containing `ams` double-struck E at 153%, magnified by a factor of 4.

$E^{\mu}\{f$

The glyph displays unevenly, and appears to be of a weight different to its neighbors. In my experience, blackboard bold is the most problematic alphabet for screen rendering, and AMS bb and Holey Roman bb are the most likely to show up as a bit ghostly (gray and indistinct) on the screen compared to other math glyphs. Both

seem to be formed by removing the interiors of glyphs from a bold serifed font. MathTime Pro 2 Holey Roman-bold is a much better fit to most math fonts of weight heavier than Computer Modern. Other such hollowed-out fonts which are occasionally used as a double-struck font, such as Caslon OpenFace and Goudy Hand-Tooled, have to my eye either a similar problem, or have insufficient hollowing-out to distinguish them from an ordinary bold font.

- Fourier, Mathpi, esstix and boondox bb appear to be very close in style, with Mathpi bb a bit less sharp. These are geometric shapes, and because of screen-rendering issues, you may find that the font rendered there appears asymmetric even though there is no problem on paper, at least for resolutions over 300 dpi.
- Mathpazo bb and Mathematica bb have a heavier appearance and should work well with fonts other than Computer Modern, but the uneven weights of their strokes can lead to unsightly screen artifacts.
- In my opinion, for best appearance on screen and on paper, the best-looking blackboard bold glyphs (matching the weights of fonts heavier than Computer Modern) are (i) BOONDOX bb-bold; (ii) MathTime Pro 2 Holey Roman-bold. In both cases, there is no ghostly appearance, but in case (i), the glyph selection is limited.

## 5 Mathematical fraktur

Fraktur font samples follow, arranged in order of blackness.

**esstix** (ESSTIX fraktur):

$\mathfrak{A}$   $\mathfrak{B}$   $\mathfrak{C}$   $\mathfrak{D}$   $\mathfrak{E}$   $\mathfrak{F}$   $\mathfrak{G}$   $\mathfrak{H}$   $\mathfrak{I}$   $\mathfrak{J}$   $\mathfrak{K}$   $\mathfrak{L}$   $\mathfrak{M}$   
 $\mathfrak{N}$   $\mathfrak{O}$   $\mathfrak{P}$   $\mathfrak{Q}$   $\mathfrak{R}$   $\mathfrak{S}$   $\mathfrak{T}$   $\mathfrak{U}$   $\mathfrak{V}$   $\mathfrak{W}$   $\mathfrak{X}$   $\mathfrak{Y}$   $\mathfrak{Z}$

**mathpi** (Mathpi fraktur):

$\mathfrak{A}$   $\mathfrak{B}$   $\mathfrak{C}$   $\mathfrak{D}$   $\mathfrak{E}$   $\mathfrak{F}$   $\mathfrak{G}$   $\mathfrak{H}$   $\mathfrak{I}$   $\mathfrak{J}$   $\mathfrak{K}$   $\mathfrak{L}$   $\mathfrak{M}$   
 $\mathfrak{N}$   $\mathfrak{O}$   $\mathfrak{P}$   $\mathfrak{Q}$   $\mathfrak{R}$   $\mathfrak{S}$   $\mathfrak{T}$   $\mathfrak{U}$   $\mathfrak{V}$   $\mathfrak{W}$   $\mathfrak{X}$   $\mathfrak{Y}$   $\mathfrak{Z}$

**mt** (MathTime Pro 2 fraktur):

$\mathfrak{A}$   $\mathfrak{B}$   $\mathfrak{C}$   $\mathfrak{D}$   $\mathfrak{E}$   $\mathfrak{F}$   $\mathfrak{G}$   $\mathfrak{H}$   $\mathfrak{I}$   $\mathfrak{J}$   $\mathfrak{K}$   $\mathfrak{L}$   $\mathfrak{M}$   
 $\mathfrak{N}$   $\mathfrak{O}$   $\mathfrak{P}$   $\mathfrak{Q}$   $\mathfrak{R}$   $\mathfrak{S}$   $\mathfrak{T}$   $\mathfrak{U}$   $\mathfrak{V}$   $\mathfrak{W}$   $\mathfrak{X}$   $\mathfrak{Y}$   $\mathfrak{Z}$

**euler** (Euler fraktur):

$\mathfrak{A}$   $\mathfrak{B}$   $\mathfrak{C}$   $\mathfrak{D}$   $\mathfrak{E}$   $\mathfrak{F}$   $\mathfrak{G}$   $\mathfrak{H}$   $\mathfrak{I}$   $\mathfrak{J}$   $\mathfrak{K}$   $\mathfrak{L}$   $\mathfrak{M}$   
 $\mathfrak{N}$   $\mathfrak{O}$   $\mathfrak{P}$   $\mathfrak{Q}$   $\mathfrak{R}$   $\mathfrak{S}$   $\mathfrak{T}$   $\mathfrak{U}$   $\mathfrak{V}$   $\mathfrak{W}$   $\mathfrak{X}$   $\mathfrak{Y}$   $\mathfrak{Z}$

**lucida** (Lucida fraktur):

$\mathfrak{A}$   $\mathfrak{B}$   $\mathfrak{C}$   $\mathfrak{D}$   $\mathfrak{E}$   $\mathfrak{F}$   $\mathfrak{G}$   $\mathfrak{H}$   $\mathfrak{I}$   $\mathfrak{J}$   $\mathfrak{K}$   $\mathfrak{L}$   $\mathfrak{M}$   
 $\mathfrak{N}$   $\mathfrak{O}$   $\mathfrak{P}$   $\mathfrak{Q}$   $\mathfrak{R}$   $\mathfrak{S}$   $\mathfrak{T}$   $\mathfrak{U}$   $\mathfrak{V}$   $\mathfrak{W}$   $\mathfrak{X}$   $\mathfrak{Y}$   $\mathfrak{Z}$

**mma** (Mathematica fraktur):

$\mathfrak{A}$   $\mathfrak{B}$   $\mathfrak{C}$   $\mathfrak{D}$   $\mathfrak{E}$   $\mathfrak{F}$   $\mathfrak{G}$   $\mathfrak{H}$   $\mathfrak{I}$   $\mathfrak{J}$   $\mathfrak{K}$   $\mathfrak{L}$   $\mathfrak{M}$   
 $\mathfrak{N}$   $\mathfrak{O}$   $\mathfrak{P}$   $\mathfrak{Q}$   $\mathfrak{R}$   $\mathfrak{S}$   $\mathfrak{T}$   $\mathfrak{U}$   $\mathfrak{V}$   $\mathfrak{W}$   $\mathfrak{X}$   $\mathfrak{Y}$   $\mathfrak{Z}$

**boondox** (BOONDOX fraktur):

$\mathfrak{A}$   $\mathfrak{B}$   $\mathfrak{C}$   $\mathfrak{D}$   $\mathfrak{E}$   $\mathfrak{F}$   $\mathfrak{G}$   $\mathfrak{H}$   $\mathfrak{I}$   $\mathfrak{J}$   $\mathfrak{K}$   $\mathfrak{L}$   $\mathfrak{M}$   
 $\mathfrak{N}$   $\mathfrak{O}$   $\mathfrak{P}$   $\mathfrak{Q}$   $\mathfrak{R}$   $\mathfrak{S}$   $\mathfrak{T}$   $\mathfrak{U}$   $\mathfrak{V}$   $\mathfrak{W}$   $\mathfrak{X}$   $\mathfrak{Y}$   $\mathfrak{Z}$

**euler** (Euler fraktur-bold):

$\mathfrak{A}$   $\mathfrak{B}$   $\mathfrak{C}$   $\mathfrak{D}$   $\mathfrak{E}$   $\mathfrak{F}$   $\mathfrak{G}$   $\mathfrak{H}$   $\mathfrak{I}$   $\mathfrak{J}$   $\mathfrak{K}$   $\mathfrak{L}$   $\mathfrak{M}$   
 $\mathfrak{N}$   $\mathfrak{O}$   $\mathfrak{P}$   $\mathfrak{Q}$   $\mathfrak{R}$   $\mathfrak{S}$   $\mathfrak{T}$   $\mathfrak{U}$   $\mathfrak{V}$   $\mathfrak{W}$   $\mathfrak{X}$   $\mathfrak{Y}$   $\mathfrak{Z}$

**mma** (Mathematica fraktur-bold):

$\mathfrak{A}$   $\mathfrak{B}$   $\mathfrak{C}$   $\mathfrak{D}$   $\mathfrak{E}$   $\mathfrak{F}$   $\mathfrak{G}$   $\mathfrak{H}$   $\mathfrak{I}$   $\mathfrak{J}$   $\mathfrak{K}$   $\mathfrak{L}$   $\mathfrak{M}$   
 $\mathfrak{N}$   $\mathfrak{O}$   $\mathfrak{P}$   $\mathfrak{Q}$   $\mathfrak{R}$   $\mathfrak{S}$   $\mathfrak{T}$   $\mathfrak{U}$   $\mathfrak{V}$   $\mathfrak{W}$   $\mathfrak{X}$   $\mathfrak{Y}$   $\mathfrak{Z}$

**mt** (MathTime Pro 2 fraktur-bold):

$\mathfrak{A}$   $\mathfrak{B}$   $\mathfrak{C}$   $\mathfrak{D}$   $\mathfrak{E}$   $\mathfrak{F}$   $\mathfrak{G}$   $\mathfrak{H}$   $\mathfrak{I}$   $\mathfrak{J}$   $\mathfrak{K}$   $\mathfrak{L}$   $\mathfrak{M}$   
 $\mathfrak{N}$   $\mathfrak{O}$   $\mathfrak{P}$   $\mathfrak{Q}$   $\mathfrak{R}$   $\mathfrak{S}$   $\mathfrak{T}$   $\mathfrak{U}$   $\mathfrak{V}$   $\mathfrak{W}$   $\mathfrak{X}$   $\mathfrak{Y}$   $\mathfrak{Z}$

**boondox** (BOONDOX fraktur-bold):

$\mathfrak{A}$   $\mathfrak{B}$   $\mathfrak{C}$   $\mathfrak{D}$   $\mathfrak{E}$   $\mathfrak{F}$   $\mathfrak{G}$   $\mathfrak{H}$   $\mathfrak{I}$   $\mathfrak{J}$   $\mathfrak{K}$   $\mathfrak{L}$   $\mathfrak{M}$   
 $\mathfrak{N}$   $\mathfrak{O}$   $\mathfrak{P}$   $\mathfrak{Q}$   $\mathfrak{R}$   $\mathfrak{S}$   $\mathfrak{T}$   $\mathfrak{U}$   $\mathfrak{V}$   $\mathfrak{W}$   $\mathfrak{X}$   $\mathfrak{Y}$   $\mathfrak{Z}$

## 5.1 Notes

- While mma is easy to read, I find it less attractive than other fraktur faces as it seems to have random variations in heights and depths, and seems overly dark. Some of these comments might also apply to lucida.
- lucida fraktur is one of the most idiosyncratic of the fraktur fonts, seeming to be considerably more influenced by Duc de Berry than by traditional fraktur sources.
- boondox fraktur is very attractive, but a bit heavy for all but the blackest text fonts (Times, Arno Pro, Lucida), while esstix is a bit too light for all but fonts like Computer Modern and Goudy Old Style.

## 6 A tool for making virtual fonts

TeX does not make it easy to take a text font (e.g., a script font) and construct from it a virtual font suitable for use in math mode, with accents properly positioned, width adjusted to match other math glyphs, and subscripts properly placed. The tool I used to do this in several cases is TeXFontUtility, available for Mac OS X as TeXFontUtility.dmg from <http://math.ucsd.edu/~msharp>. The program is specific to Mac OS X, but the output may be used in any L<sup>A</sup>T<sub>E</sub>X installation.

With this tool, the math metrics can be adjusted visually. For example, adjusting the left and right side-bearings, subscript position and accent position for the glyph A of boondox script is simply a matter of selecting and dragging the line segments (the hat for the accent) to your personal tastes. See figure 1.

This tool has many other uses, most importantly, serving as a graphical interface to fontinst for purposes such as substituting an italic alphabet for the math italic glyphs in a virtual font based on mtpro2

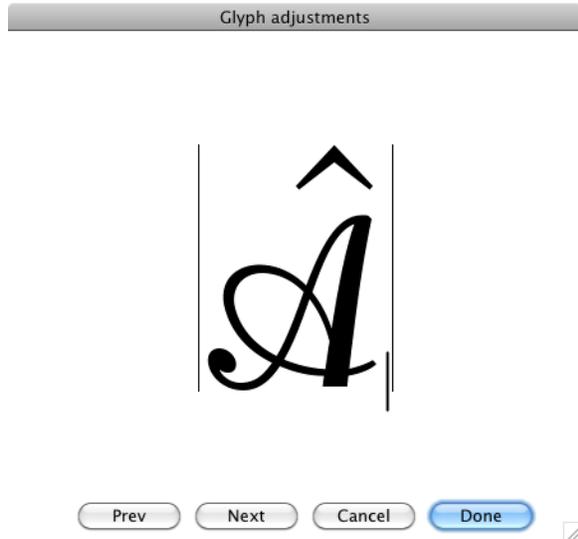


Figure 1: Adjusting a glyph in TeXFontUtility.

(and making the appropriate visual adjustments to the metrics for math mode), and diagnosing common errors made in adding packages to  $\text{\TeX}$  Live. All the “frankenfonts” in the file `mathsamples.pdf` mentioned below were created using this program.

## 7 In conclusion

- Several of the packages described above are advertised as suitable companions for Times, a rather heavy font that is not generally considered the best choice for book text. (It was designed for newspaper use, where it would remain legible even after the ink soaked unevenly into cheap newsprint paper, and was compact enough for the narrow columns.) Nonetheless, there is a high variance in the weights of those companions. For example, both `boondox` fraktur and `mt` fraktur were designed for Times, and there is a great difference between their weights. In my opinion, this means that the issue of matching math alphabet weight to text font weight is not critical, and there are many examples of successful use of the `mtpro2` fonts with text fonts having a multiplicity of weights. For examples, see the file `mathsamples.pdf` at <http://math.ucsd.edu/~msharpe/>.
- If your interest in math fonts goes beyond the basic level (e.g., you are writing a book or lecture notes with mathematical content), you should look into the commercial products Lucida from

<http://www.tug.org/lucida> and MathTime Pro 2 from <http://pctex.com>. Both are high quality products, and are excellent values for the prices. Even if you only use small pieces of the collections, these are much better buys than most commercial text fonts.

- The Mathematica fonts are not of very high quality as a collection (i.e., not suitable for professional use as they stand), but they have some excellent components. In particular, the calligraphic font may be turned into a good target for `\mathcal` after its metrics have been fine-tuned. You are missing out on some good stuff if you don’t install this free collection. Sadly, Jens-Peer Kuska, the theoretical physicist who devised the  $\text{\LaTeX}$  support files for these fonts, died before his time in 2009, and it seems unlikely that they will be revised in the near future.
- The `rsfs` package is not well-suited to `\mathcal`, being much too slanted. The best options for `\mathcal` are, in my opinion, `rsfso`, `dutchcal`, `boondoxo` and `mt`, the latter requiring the (non-free) `mtpro2` collection.
- If you own the `mtpro2` collection, look into the ‘curly’ script font, which seems useful, though a bit heavy.
- It is questionable whether there is value in the Mathpi fonts given that each of its constituents may be approximated closely by free alternatives, and the fonts can be tricky to install.
- The `boondox` calligraphic font is quite handsome. I prefer it to be less sloped, along the lines of `rsfso`. This is provided by the option `boondoxo`, which uses virtual fonts sloped much like `rsfso`.
- It is possible to produce a candidate for a math script font starting from any script text font, but it is very difficult to locate suitable text script fonts. The most frequent problem is long tails on some glyphs. Another is that the glyphs can be over-elaborate, more suited to a wedding invitation than a scientific publication.

◇ Michael Sharpe  
 Math Dept, UCSD  
 La Jolla, CA 92093-0112  
 USA  
[msharpe \(at\) ucsd dot edu](mailto:msharpe@ucsd.edu)  
<http://math.ucsd.edu/~msharpe/>

## Another incarnation of Lucida: Towards Lucida OpenType

Ulrik Vieth and Mojca Miklavc

### Abstract

$\TeX$  has been in existence for more than 30 years. During this time,  $\TeX$  engines and device drivers have gone through multiple generations of font technology: METAFONT, PostScript Type 1, and OpenType. While choices for text fonts have greatly increased, there have always been few choices for math fonts and even fewer for complete font families.

The Lucida family of typefaces by Bigelow & Holmes is a notable exception, as it provides not only a math font, but also a complete font family, consisting of a large repertoire of serif, sans-serif, monospace, and fancy variants. Unfortunately, the current distribution of Lucida fonts dates back to 1993 and is limited by the use of PostScript Type 1 fonts, which support only 8-bit character sets.

In this article, we report the current status of an ongoing joint project of Bigelow & Holmes and TUG to develop a new distribution of Lucida OpenType with better Unicode language and math support.

### 1 Historical perspective of font technology

$\TeX$  has been in existence for more than 30 years. During this time,  $\TeX$  engines and device drivers have evolved through multiple generations of font technology (METAFONT, Type 1, OpenType) and multiple generations of output formats (DVI, PostScript, PDF).

**The era of METAFONT fonts** When  $\TeX$  was first developed in the late 1970s and early 1980s, there were no established standards of font technology which could be used, so METAFONT was developed as a companion to  $\TeX$ , and all related file formats for font metrics (TFM) and bitmap fonts (PK) were invented as well.

As it turned out, METAFONT never caught on with font designers, so there were very few choices for fonts available for use with  $\TeX$  in this era, both for text and math typesetting. Besides the earliest instances of METAFONT fonts, Computer Modern and AMS Euler, there were only a few more, most of them variants of the above, such as Concrete and CM Bright.

**The era of PostScript Type 1 fonts** When PostScript printers came into use in the early 1990s,  $\TeX$  entered another era of font technology, as scalable Type 1 fonts became the preferred format. It became possible to use the commercial offerings of Type 1 fonts from many font vendors, which could

now be set up for use with  $\TeX$  using `fontinst` or `afm2tfm`.

This development continued when PDF $\TeX$  was developed and on-screen viewing of PDF files came into common use in the late 1990s.

While the use of METAFONT-generated bitmap fonts packaged into Type 3 fonts was still acceptable for printing, such bitmap fonts turned out to be inadequate for screen rendering in PDF viewers. As a result, the use of METAFONT fonts became unpopular, and efforts were undertaken to provide Type 1 replacements for METAFONT fonts.

In this era, choices of text fonts were increased significantly, but choices of math fonts remained limited. Besides CM and AMS Euler, converted from METAFONT, there were only the commercial offerings of MathTime and Lucida New Math at first.

It was only much later in this era that more choices of math fonts eventually became available, such as `txfonts`, `pxfonts`, `mathpazo`, `fourier`, and `mathdesign`, providing companion math fonts for use with popular text typefaces, such as Times, Palatino, Utopia, Charter, and Garamond.

Nevertheless, as we are nearing the end of this era, these choices of math fonts are still very few compared to the vast number of available text fonts, and it took more than a decade to get there.

**The era of OpenType fonts** In recent years,  $\TeX$  has entered yet another era of font technology, as OpenType fonts are now becoming the preferred font format to support the needs of Unicode.

Many font vendors have switched their commercial offerings from Type 1 to OpenType format, and Type 1 fonts are becoming obsolete due to their limitations to 8-bit character sets.

With the development of X $\TeX$  and Lua $\TeX$  in recent years, new  $\TeX$  engines have become available which support Unicode input and OpenType output directly, without the need for a complicated setup of TFM font metrics or font map files.

At the same time, support for virtual fonts is being phased out in packages such as Con $\TeX$ t MkIV, making it difficult to continue to use Type 1 fonts with virtual fonts in those packages.

As we are entering this era of font technology, there is once again a need to develop replacements for existing fonts, this time providing OpenType replacements for Type 1 fonts.

With the development of the Latin Modern and  $\TeX$  Gyre fonts in recent years, replacements for the Computer Modern text fonts and several common PostScript fonts already exist, but the corresponding math fonts are still under development.

At the time of writing, choices of full-featured OpenType math fonts remain limited to Cambria Math (developed by Microsoft), Asana Math (derived from `pxfonts`) and XITS Math (derived from STIX fonts). Additional choices of OpenType math fonts are still unfinished, such as Neo Euler (derived from AMS Euler) and Latin Modern Math (derived from Computer Modern Math).

With the addition of Lucida Math as another choice of OpenType math fonts under development, we are about to reach the same level of font support in the OpenType era that was available in the early years of the PostScript era in the mid-1990s.

Nevertheless, the general trend continues also in the OpenType era in that there are few choices for math fonts and even fewer for complete font families.

## 2 History of Lucida font distributions

The Lucida family of fonts was developed by the Bigelow & Holmes foundry of Charles Bigelow and Kris Holmes in the mid-1980s [1]. At this time, Chuck Bigelow was on the faculty at Stanford, so he was well aware of the development of  $\TeX$  and the Computer Modern fonts by Don Knuth. A primary goal of Lucida was to create a typeface design which would digitize well, even at relatively low resolutions.

Another goal of Lucida was to provide a complete font family of matching designs for serif, sans-serif, and monospace fonts, which were later augmented by a number of fancy variants [2].

Yet another goal of Lucida was to provide an extensive character set, including Latin, Greek, symbols, and even dingbats, so that the fonts could be used for math typesetting as well. (The dingbats fonts were later distributed independently [3].)

The original versions of Lucida fonts from the mid-1980s are still being sold by some font vendors under the names of Lucida Serif and Lucida Math, but these versions were never really supported with a setup for use with  $\TeX$ .

The current versions of Lucida fonts were extended and revised for use with  $\TeX$  in the early 1990s in cooperation with Y&Y Inc., in particular its principal Berthold Horn for  $\TeX$ -specific adjustments. Y&Y sold the fonts under the names of Lucida Bright and Lucida New Math for many years, until the company was dissolved. The same font packages are now being supported and sold directly by TUG [4] and by  $\text{PC}\TeX$  Inc. [5].

Finally, other versions of Lucida fonts that exist are widely distributed as system fonts with operating systems or software development kits. These include the Lucida Console and Lucida Sans Unicode

fonts on MS Windows [6, 7], the Lucida Grande fonts on Apple Mac OS X [8], and a set of Lucida fonts distributed with Sun's Java Development Kits (JDK).

All of these fonts are in TrueType format and provide some level of Unicode coverage, some of them even including support for non-Latin scripts such as Greek, Cyrillic, Arabic, Hebrew, etc.

Unfortunately, most of these Lucida Unicode system fonts provide only single font instances or incomplete font families, so they are not really well suited for sophisticated typesetting.

As a result, users who want to use Lucida for typesetting with  $\TeX$  are essentially stuck with the Type 1 distribution of the early 1990s, providing only a limited 8-bit character set and requiring the use of virtual fonts to support accented languages, which is no longer up to the requirements of modern  $\TeX$  engines geared toward Unicode typesetting with OpenType font technology.

## 3 Scope of the Lucida Type 1 distribution

The current distribution of Lucida Type 1 fonts for use with  $\TeX$  was originally developed in the early 1990s by Bigelow & Holmes and Y&Y Inc. It is now being supported and sold directly by TUG. The TUG distribution consists of two font packages: Lucida Basic and Lucida Complete [9, 10].

The basic distribution provides three complete font families: Lucida Bright, Lucida Sans Typewriter, and Lucida New Math (Fig. 1).

The complete distribution adds the following font families: Lucida Sans, Lucida (Serif) Typewriter, Lucida Fax, Lucida Casual, as well as several fancy variants: Lucida Blackletter, Lucida Calligraphy, and Lucida Handwriting (Fig. 2).

Given the limitations of Type 1 technology, the fonts are based on an 8-bit character set.

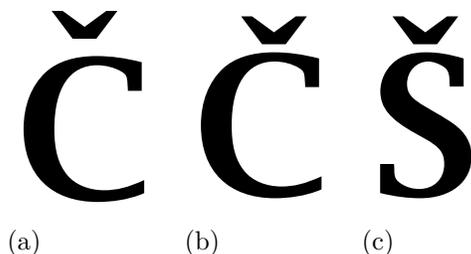
The recommended setup suggested by Y&Y Inc. was to use the so-called `TeXnANSI` encoding (LY1), which combines parts of the 7-bit old  $\TeX$  encoding (OT1) in the lower half with the Windows ANSI 1252 encoding in the upper half [11, 12].

An alternative setup suggested by the  $(\mathcal{L})\TeX$  community was to use the so-called `TeXBase1` (8r) encoding as a base font encoding for virtual fonts, implementing the 8-bit Cork text and text companion encodings (T1 and TS1).

Since the Cork encoding extends considerably beyond the scope of Windows ANSI 1252 (Latin 1), some of the accented letters could not be provided by glyphs from the fonts, but had to be substituted by constructions in the virtual fonts.

The quality of these constructed glyphs varies considerably, but it rarely matches the quality of





**Figure 3:** Comparison of the placement of accents: (a) constructed letter c-caron, using virtual fonts, (b) designed letter c-caron, (c) designed letter s-caron.

Moreover, these modern font distributions also support a much wider range of 8-bit font encodings, such as QX for Polish, CS for Czech and Slovak, L7X for Lithuanian, or even T5 for Vietnamese, besides EC for Cork and TeXnANSI.

The second problem arises from the use of virtual fonts to supply missing accented glyphs, and the associated problems of design quality.

An example of such a problem is illustrated in Fig. 3, comparing a constructed letter c-caron from a virtual font (as it used to be) with a designed letter c-caron from an OpenType font (as it should be).

In the constructed letter, the placement of the accent is done automatically, based on the glyph metrics (bounding box), so the accent is placed too high and it is centered on the geometric center of the glyph rather than the visual center.

If you compare the constructed letter c-caron with the designed letter s-caron (which happens to be available in the TeXnANSI encoding, despite not being part of ISO Latin 1), the difference in quality becomes very obvious. It's unquestionably desirable to have a more comprehensive set of properly designed accented letters for better language support.

Finally, a third problem consists in the requirement for virtual font support in TeX engines and device drivers to be able to provide substitutions for missing glyphs in the first place.

While the mainstream TeX distributions (such as TeX Live) have supported virtual fonts in programs such as dvips or pdftex for many years, support for virtual fonts was never universal, and it was notably absent in the device drivers of the commercial TeX distribution of Y&Y Inc.

Finally, in recent developments we are facing a situation that support for traditional virtual fonts (based on VF and TFM files) is being phased out in modern macro packages such as ConTeXt MkIV. While the LuaTeX engine still supports traditional virtual fonts, the font loader in ConTeXt MkIV now uses a completely different mechanism.

Karl Berry (TUG)	project coordination
Chuck Bigelow (B&H)	glyph design, coordination
Khaled Hosny	glyph assembly
Mojca Miklavec	testing of text fonts
Ulrik Vieth	testing of math fonts
Hans Hagen	technical advisory
Taco Hoekwater	technical advisory

**Table 1:** Team members of the Lucida OpenType project team and responsibilities.

## 5 Inception of the Lucida OpenType project

The idea of a project to create a Lucida OpenType font distribution was first conceived at last year's ConTeXt meeting in September 2010.

When a user asked how to set up Lucida for use with ConTeXt MkIV, Hans Hagen's answer was simply: "Don't use Lucida. It doesn't work!"

After a brief discussions, it was eventually concluded that something needed to be done about it, or else Lucida would soon become an obsolete and unsupported font family.

It was then suggested to hire Khaled Hosny as a developer to repackage and extend the existing Lucida Type 1 fonts into OpenType fonts and to seek support from TUG to fund and coordinate the development.

By October 2010, just a few weeks after the conference, Karl Berry had entered discussions between TUG and Bigelow & Holmes about the project, and by November 2010, the necessary legal agreement had been drafted and a project team was assembled, consisting of the team members listed in Table 1.

The agreed scope of the project was to develop OpenType versions of the Lucida basic distribution at first, which includes Lucida Bright, Lucida Math, and Lucida Sans Typewriter. Other family members, such as Lucida Sans or others may be added in a second phase of the project.

The goal for Lucida text font families was to develop OpenType fonts with good Unicode support for Latin languages, so these fonts will feature a significant number of accented Latin letters, but hardly any non-Latin scripts. In most cases, no new glyph designs will be required, just the assembly and placement of combining accents.

The goal for Lucida Math was to develop an OpenType math font with good Unicode support of math symbols and math alphabets. Besides the assembly of existing symbols from Lucida Type 1 fonts, a number of additional symbols may need to be designed, while most of the math alphabets can be taken from existing Lucida fonts.

LucidaBright	692 (955)
LucidaBright-Italic	395
LucidaBright-Demi	395 (527)
LucidaBright-DemiItalic	395
LucidaSansTypewriter	358
LucidaSansTypewriter-Oblique	358
LucidaSansTypewriter-Bold	358
LucidaSansTypewriter-BoldOblique	358

**Table 2:** Number of glyphs per font for the Lucida OpenType text fonts. (Numbers in brackets are the totals including small caps and oldstyle figures.)

## 6 Progress of the Lucida OpenType project

By November 2010, the project team was ready to start working, and by December 2010, the project was already well under way.

Bigelow & Holmes had supplied the designs of a number of additional glyphs for several Unicode blocks (mostly additional math symbols), and the first preliminary versions of OpenType fonts had been assembled for testing.

By January 2011, testing of the text fonts had started, while work on assembling combining accents for accented letters continued.

For the text fonts, testing mostly focused on checking the placement of combining accents and tracking the number of languages covered or the number of glyphs missing for each language.

Some statistics for the number of glyphs per font (as of May 2011) are given in Table 2 for each of the Lucida text fonts. Unsurprisingly, the regular version of Lucida Bright is the most complete one, followed by other Lucida Bright fonts.

Besides the basic ASCII and ISO Latin 1 blocks, which were already available in 8-bit Type 1 fonts, all of the Lucida Bright fonts include the complete Latin Extended-A block (U+0100 to U+017F), while only the regular Lucida Bright also includes some parts of Latin Extended-B (U+0180 to U+024F) and Latin Extended Additional (U+1E00 to U+1EFF).

Besides the more extensive glyph coverage, the regular version of Lucida Bright is also the most advanced with regard to feature support for combining marks, providing some support for multiple marks, as well as marks above and below.

For the Lucida Sans Typewriter fonts, the glyph coverage is somewhat smaller than for the Lucida Bright fonts. Most of the Latin Extended-A block is also available, but a few gaps remain, awaiting new designs from Bigelow & Holmes. Apart from that, the typewriter fonts also have fewer ligatures,

but those are unlikely to be used anyway.

Compared to the old Lucida Type 1 fonts, which typically had 252 glyphs, the number of 395 glyphs in the new Lucida OpenType fonts already presents a significant advantage, especially for users of Latin languages beyond Latin 1.

Compared to other versions of Lucida fonts with Unicode support, such as Lucida Sans Unicode (1779 glyphs) or even Lucida Grande (2826 glyphs), however, the scope of the new Lucida OpenType fonts is still pretty small, as it only includes support for Latin, but not for other scripts, such as Greek, Cyrillic, Arabic, Hebrew, etc.

By March 2011, testing of the math fonts had also started, while ongoing work on extending and improving the text fonts continued.

For the math fonts, testing mostly focused on typesetting a variety of sample math documents to check for missing symbols or alphabets.

In total the new Lucida Math OpenType font includes 2148 glyphs, of which 948 glyphs are from math alphabets (U+1D400 to U+1D7FF).

An overview of the available math alphabets in Lucida Math is shown in Fig. 4. As it turned out, most of the math alphabets in Unicode could be supplied from existing Lucida Type 1 fonts (including some fancy variants such as Lucida Calligraphy and Blackletter). Only a few alphabets remain missing, such as lowercase bold Script, upper- and lowercase bold Fraktur, and lowercase Blackboard Bold letters. In addition, some individual symbols are missing in just a few alphabets.

As a unique feature, Lucida Math provides two alternate versions of the math italic alphabet, but only one version can be assigned to Unicode slots, so the other one has to be relegated to slots in the private use area and accessed via font substitutions, if the `+ss01` feature is selected.

As for the coverage of math symbols, all of the existing symbols from Lucida Type 1 math fonts have been integrated into Lucida Math OpenType. In addition, Bigelow & Holmes have supplied new designs for some additional Unicode blocks of math symbols.

While there are still a few gaps left to be filled, most of the gaps are in lesser used alphabets, so they will not affect most documents. In our tests, we have successfully typeset a number of sample math and physics documents without encountering any missing symbols.

A very small sample of math typesetting with Lucida Math is shown in Figs. 5–6 (inspired by [15]).

We are confident that Lucida Math is about as good as other existing OpenType math fonts, such

LucidaNewMath-Roman	<code>\mathup</code>	ABCXYZ abcxyz	ABIEΨΩ αβγξψω	0123
LucidaNewMath-AltItalic	<code>\mathit(-)</code>	<i>ABCXYZ abcxyz</i>	<i>ABIEΨΩ αβγξψω</i>	
LucidaNewMath-Demi	<code>\mathbfup</code>	<b>ABCXYZ abcxyz</b>	<b>ABIEΨΩ αβγξψω</b>	<b>0123</b>
LucidaNewMath-AltDemiItalic	<code>\mathbfit(-)</code>	<b><i>ABCXYZ abcxyz</i></b>	<b><i>ABIEΨΩ αβγξψω</i></b>	
LucidaNewMath-Roman	<code>\mathup</code>	ABCXYZ abcxyz	ABIEΨΩ αβγξψω	0123
LucidaNewMath-Italic	<code>\mathit(+)</code>	<i>ABCXYZ abcxyz</i>	<i>ABIEΨΩ αβγξψω</i>	
LucidaNewMath-Demi	<code>\mathbfup</code>	<b>ABCXYZ abcxyz</b>	<b>ABIEΨΩ αβγξψω</b>	<b>0123</b>
LucidaNewMath-DemiItalic	<code>\mathbfit(+)</code>	<b><i>ABCXYZ abcxyz</i></b>	<b><i>ABIEΨΩ αβγξψω</i></b>	
LucidaSans	<code>\mathsfup</code>	ABCXYZ abcxyz	(not assigned)	0123
LucidaSans-Italic	<code>\mathsfit</code>	<i>ABCXYZ abcxyz</i>	(not assigned)	
LucidaSans-Demi	<code>\mathbfsfup</code>	<b>ABCXYZ abcxyz</b>	<b>ABIEΨΩ αβγξψω</b>	<b>0123</b>
LucidaSans-DemiItalic	<code>\mathbfsfit</code>	<b><i>ABCXYZ abcxyz</i></b>	<b><i>ABIEΨΩ αβγξψω</i></b>	
LucidaNewMathSymbol	<code>\mathcal</code>	<i>ABCXYZ</i>		
LucidaNewMathSymbol-Demi	<code>\mathbfcal</code>	<b><i>ABCXYZ</i></b>		
LucidaCalligraphy	<code>\mathscr</code>	<i>ABCXYZ abcxyz</i>		
—	<code>\mathbfscr</code>	<b><i>ABCXYZ</i></b> (missing)		
LucidaBlackletter	<code>\mathfrak</code>	<b><i>ABCXYZ abcxyz</i></b>		
—	<code>\mathbffrak</code>	(missing) (missing)		
LucidaNewMathArrows	<code>\mathbb</code>	<b><i>ABCXYZ</i></b> (missing)		

**Figure 4:** Overview of math alphabets in Lucida Math OpenType and where they were taken from. Note that switching between italic and alternate italic requires leaving math mode and reloading the font with different OpenType feature settings: (+) = fonts loaded with option `+ss01`, (-) = fonts loaded with option `-ss01`.

as Cambria Math or XITS Math. While Cambria Math is often used for comparison, as it was the very first OpenType math font, it also has some gaps in the math alphabets, and it may depend on the usage which ones are relevant.

## 7 Status of the Lucida OpenType project

As of April 2011, shortly before the presentation of the project at the EuroBachTeX 2011 conference, a set of preliminary versions of Lucida OpenType fonts has been completed. However, the project now faces an uncertain future.

For one reason, Khaled Hosny, our main developer, will be unavailable for some time due to being drafted for military service in Egypt.

For another reason, Bigelow & Holmes did not have enough time during the academic year to supply designs for missing glyphs, so even a number of trivial issues affecting only a few glyphs have remained unfinished so far.

As for the current status, the Lucida OpenType text and math fonts clearly represent a work in progress, but not yet a finished product.

For the text fonts, it would be desirable to reach a consistent level of glyph coverage in all fonts, including all of Latin Extended-A, and possibly Latin Extended-B or Latin Extended Additional.

Of course, supporting a certain number of Latin

Unicode blocks directly implies supporting a certain number of languages with Latin scripts.

In the case of Latin Extended-A and -B, this primarily implies support for European languages. In the case of Latin Extended Additional, this might even imply support for Vietnamese, although it is questionable if this will ever happen.

Besides a consistent level of glyph coverage, it would also be desirable to reach a consistent level of feature support for combining marks, including marks above, marks below, and multiple marks.

So far, only the regular version of Lucida Bright comes near this level (with some remaining gaps), while the other fonts include only Latin Extended-A (also with some remaining gaps).

For the math font, the existing coverage of math symbols and alphabets is already quite good, but it would also be desirable to close the remaining gaps in the alphabets, requiring some new designs for bold Script and bold Blackletter fonts.

Finally, once the basic set of Lucida OpenType fonts (Lucida Bright, Lucida Math, and Lucida Sans Typewriter) have been completed, there are other members of the Lucida complete set which remain to be done in a second phase, such as Lucida Sans and possibly some of the fancy variants.

Most likely, it will not be worth the effort to create a full set of accented letters for each of the

**Theorem 1 (Residue Theorem).** Let  $f$  be analytic in the region  $G$  except for the isolated singularities  $a_1, a_2, \dots, a_m$ . If  $\gamma$  is a closed rectifiable curve in  $G$  which does not pass through any of the points  $a_k$  and if  $\gamma \approx 0$  in  $G$  then

$$\frac{1}{2\pi i} \int_{\gamma} f = \sum_{k=1}^m n(\gamma; a_k) \operatorname{Res}(f; a_k).$$

**Theorem 2 (Maximum Modulus).** Let  $G$  be a bounded open set in  $\mathbb{C}$  and suppose that  $f$  is a continuous function on  $G^-$  which is analytic in  $G$ . Then

$$\max\{|f(z)| : z \in G^-\} = \max\{|f(z)| : z \in \partial G\}.$$

**Figure 5:** Sample math document typeset with Lucida Bright and Lucida Math OpenType using the default set of math italic (OpenType feature `-ss01`).

**Theorem 1 (Residue Theorem).** Let  $f$  be analytic in the region  $G$  except for the isolated singularities  $a_1, a_2, \dots, a_m$ . If  $\gamma$  is a closed rectifiable curve in  $G$  which does not pass through any of the points  $a_k$  and if  $\gamma \approx 0$  in  $G$  then

$$\frac{1}{2\pi i} \int_{\gamma} f = \sum_{k=1}^m n(\gamma; a_k) \operatorname{Res}(f; a_k).$$

**Theorem 2 (Maximum Modulus).** Let  $G$  be a bounded open set in  $\mathbb{C}$  and suppose that  $f$  is a continuous function on  $G^-$  which is analytic in  $G$ . Then

$$\max\{|f(z)| : z \in G^-\} = \max\{|f(z)| : z \in \partial G\}.$$

**Figure 6:** Sample math document typeset with Lucida Bright and Lucida Math OpenType using the alternate set of math italic (OpenType feature `+ss01`).

fancy variants, but it would certainly be useful to do so for the major variants such as Lucida Sans, and to provide a basic conversion of Type 1 fonts to OpenType for some of the other variants.

As for availability, the Lucida fonts will remain non-free commercial fonts with all rights held by Bigelow & Holmes, and licenses being sold by TUG. The members of the project team will be rewarded with a free license for the fonts, but will not get any proceeds from the sales.

## 8 Post-conference updates

The bulk of this article was written in May 2011 and represents the status as of EuroBach<sub>T</sub>E<sub>X</sub> 2011.

As of June 2011 the project has been regaining momentum, as Khaled Hosny is now temporarily back to work on the project during his spare time.

As one of the first steps, the glyph coverage of the Lucida Sans Typewriter fonts has been extended to the same level as the Lucida Bright fonts, now featuring 395 glyphs representing the complete Latin Extended-A Unicode block.

As another step, a very basic conversion of the Lucida Sans fonts from Type 1 to OpenType format has been done, so that a complete set of serif, sans-serif, and monospace fonts is now available.

While the glyph coverage of the converted fonts is limited to the same 250 glyphs, having the fonts available in OpenType format should make it easier to start extending these fonts as well.

Further steps are under discussion and could be directed either towards converting more fonts or towards extending the glyph and feature coverage of existing fonts (or a bit of both).

Finally, a preliminary version of a bold math font has also been assembled, which might be used in an all-bold context such as headings or theorems. For a start, only the available glyphs from demibold Lucida Math Type 1 fonts have been assembled, but ideally, such a bold math font should eventually cover a complete set of bold symbols and alphabets, including heavy versions of bold alphabets.

In any case, work on Lucida OpenType is now continuing and has been showing great progress in just a few days, so we are confident that something useful will eventually come out of this project.

## References

- [1] Charles Bigelow: Notes on Lucida designs, November 2005.  
<http://tug.org/store/lucida/designnotes.html>

- [2] Wikipedia article: Lucida  
<http://en.wikipedia.org/wiki/Lucida>
  - [3] Wikipedia article: Wingdings  
<http://en.wikipedia.org/wiki/Wingdings>
  - [4] T<sub>E</sub>X Users Group: Lucida and T<sub>E</sub>X  
<http://tug.org/store/lucida/>
  - [5] PCT<sub>E</sub>X Inc.: Lucida Fonts  
[http://www.pctex.com/Lucida\\_Fonts.html](http://www.pctex.com/Lucida_Fonts.html)
  - [6] Charles Bigelow and Kris Holmes:  
The design of a Unicode font, *Electronic Publishing*, 6(3), 289–305, September 1993.  
<http://cajun.cs.nott.ac.uk/wiley/journals/epobetan/pdf/volume6/issue3/bigelow.pdf>
  - [7] Wikipedia article: Lucida Sans Unicode  
[http://en.wikipedia.org/wiki/Lucida\\_Sans\\_Unicode](http://en.wikipedia.org/wiki/Lucida_Sans_Unicode)
  - [8] Wikipedia article: Lucida Grande  
[http://en.wikipedia.org/wiki/Lucida\\_Grande](http://en.wikipedia.org/wiki/Lucida_Grande)
  - [9] T<sub>E</sub>X Users Group: Lucida basic font set  
<http://tug.org/store/lucida/basic.html>
  - [10] T<sub>E</sub>X Users Group: Lucida complete font set  
<http://tug.org/store/lucida/complete.html>
  - [11] Y&Y Inc.: T<sub>E</sub>X'n ANSI (LY1) font encoding  
<http://tug.org/yandy/ly1.htm>
  - [12] Wikipedia article: Windows 1252 encoding  
<http://en.wikipedia.org/wiki/Windows-1252>
  - [13] CTAN: Lucida Font metrics  
<http://ctan.org/pkg/lucida/>
  - [14] Sebastian Rahtz and David Carlisle:  
The `lucidabr` package, November 2005.  
<http://ctan.org/pkg/psnfssx-luc/>
  - [15] Stephen G. Hartke: A survey of Free Math Fonts for T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X, May 2006.  
[http://ctan.org/tex-archive/info/Free\\_Math\\_Font\\_Survey/en/survey.pdf](http://ctan.org/tex-archive/info/Free_Math_Font_Survey/en/survey.pdf)
- ◇ Ulrik Vieth  
Stuttgart, Germany  
ulrik dot vieth (at) arcor dot de
  - ◇ Mojca Miklavec  
Sežana, Slovenia  
mojca dot miklavec dot lists (at) gmail dot com

**MFLua**

Luigi Scarso

**Abstract**

We present a new implementation of METAFONT which embeds a Lua interpreter. It is fully compatible with canonical METAFONT but it has some internal “sensors” — read-only callbacks — to collect data for use in possible post-processing. An example of post-processing that extracts the outlines of some glyphs is discussed.

**1 Introduction**

MFLua is an extension of METAFONT that embeds a Lua [3] interpreter. It doesn’t introduce any new primitives, so a METAFONT file can be used with MFLua without any modification to produce exactly the same result. The Lua interpreter inside MFLua doesn’t change the internal state of METAFONT in any way and it’s not reachable from inside METAFONT. This is a strict requirement: MFLua must be fully compatible at least with the current release of METAFONT (which is currently 2.718281).

The Lua interpreter is used to register the data coming from new “Lua sensors” which are, practically speaking, read-only callbacks, i.e. functions inserted into the Pascal WEB code that call external Lua scripts, which eventually do nothing. Some sensors store the same information available with the various tracing instructions, but others are placed where there are no tracing instructions; also, not all procedures with tracing instructions have a sensor. The goal is to collect as much data as possible about the outlines of a METAFONT picture — typically a glyph.

Important note: Although MFLua is able to process a full set of characters, it’s still alpha-quality code: just a bit more than proof-of-concept.

**2 The Lua sensors**

It’s well-known that Lua<sub>T<sub>E</sub>X</sub> embeds a Lua interpreter, and it’s relatively simple to read its source code to find where and how the interpreter is initialised; this is, moreover, a particular case of a call of a C function from a Pascal WEB function, which is possible thanks to the automatic translation from Pascal WEB to C (the Web2C translator) and it’s widely used in pdf<sub>T<sub>E</sub>X</sub> and in METAFONT too (Lua<sub>T<sub>E</sub>X</sub> is now implemented in CWEB).

**2.1 Initialization**

The first step is to initialise the Lua interpreter. This is done by inserting in `mf.web` the procedure

`mflua_begin_program` (without parameters) just after the `begin` of the main program; Web2C translates it to `mfluabeginprogram` (without “\_”) and then the compiler looks for the symbol among the available sources. By convention all sensors start with `mflua` prefix and they are declared in the header `mflua.h` and implemented in the file `mflua.c`; both the files are inside the `mflua` folder which also contains the source of a canonical Lua distribution. Hence, in `mflua.h` we have:

```
extern int mfluabeginprogram();
and mflua.c contains its implementation:
```

```
lua_State *Luas[];
int mfluabeginprogram()
{
    lua_State *L = luaL_newstate();
    luaL_openlibs(L);
    Luas[0] = L;
    /* execute Lua external "begin_program.lua" */
    const char* file = "begin_program.lua";
    int res = luaL_loadfile(L, file);
    if ( res==0 ) {
        res = lua_pcall(L, 0, 0, 0);
    }
    priv_lua_reporterrors(L, res);
    return 0;
}
```

As we can see, the C function creates a new Lua state `L`, saves it in a global variable, loads the standard libraries (i.e. `math`, `string`, etc.) and evaluates the external file `begin_program.lua`. This is a common pattern: the `mflua*` sensor calls an external script and evaluates it or its function; the return value is never used because it can potentially modify the state of the METAFONT process. In this way we can manage the sensor data without recompiling the program.

The script `begin_program.lua` is quite simple, just the “greetings” message:

```
print("mflua_begin_program says 'Hello world!'")
but other scripts are more complex; for example, the
sensor mfluaPRE_fill_envelope_rhs(rhs) has one
input rhs (of type halfword) and its implementation
calls the script do_add_to.lua that contains the
function PRE_fill_envelope_rhs(rhs):
```

```
int mfluaPREfillenvelopeperhs P1C (halfword, rhs)
{ lua_State *L = Luas[0];
  const char* file = "do_add_to.lua";
  int res = luaL_loadfile(L, file);
  if ( res==0 ){
    res = lua_pcall(L, 0, 0, 0);
    if ( res==0 ){
      /* function to be called */
      lua_getglobal(L, "PRE_fill_envelope_rhs");
      /* push 1st argument */
      lua_pushnumber(L, rhs);
```

```

/*do the call (1 arguments, 1 result)*/
res = lua_pcall(L, 1, 1, 0) ;
if ( res==0 ){ /* retrieve result */
    int z = 0;
    if (!lua_isnumber(L, -1)){
        fprintf(stderr,
"\n! Error:function 'PRE_fill_envelope_rhs'
must return a number\n",lua_tostring(L, -1));
        lua_pop(L, 1);/*pop returned value*/
        return z;
    }else{
        z = lua_tonumber(L, -1);
        lua_pop(L, 1);/*pop returned value*/
        return z;
    }
}
}
}
}
priv_lua_reporterrors(L, res);
return 0; }

```

Here is the related Lua function `PRE_fill_envelope_rhs(rhs)`. It's not important to understand the details now — suffice it to say that it stores the knots of an envelope:

```

function PRE_fill_envelope_rhs(rhs)
    print("PRE_fill_envelope_rhs")
    local knots, knots_list
    local index, char
    local chartable = mflua.chartable
    knots = _print_spec(rhs)
    index = (0+print_int(LUAGLOBALGET_char_code()))
    +(0+print_int(LUAGLOBALGET_char_ext()))*256
    char = chartable[index] or {}
    knots_list = char['knots'] or {}
    knots_list[#knots_list+1] = knots
    char['knots'] = knots_list
    chartable[index] = char
    return 0; end

```

As a general rule, every sensor has exactly one Lua function; the script is loaded and the function is evaluated each time the sensor is activated (therefore the script doesn't maintain state between two calls). Furthermore, a sensor that has at least one input must be registered in `texmf.defines`, so we have for example

```

@define procedure mfluaPREfillenveloperhs();
but not
@define procedure mfluabeginprogram(); .

```

## 2.2 Exporting WEB procedures via Web2C

The files `mflua.h` and `mflua.c` fully define the implementation of the sensors and also functions needed to read some of METAFONT's global data. For example, character numbers are stored in the global METAFONT variables `char_code` and `char_ext`, and Web2C translates them in C as components of the

global array `internal` with index `char_code` and `char_ext`, so that it's easy to read them in `mflua.c`:

```

static int
priv_mfweb_LUAGLOBALGET_char_code(lua_State *L)
{ integer char_code=18;
  integer p=
    roundunscaled(internal[char_code])%256;
  lua_pushnumber(L,p);
  return 1;
}

static int
priv_mfweb_LUAGLOBALGET_char_ext(lua_State *L)
{ integer char_ext=19;
  integer p=
    roundunscaled(internal [char_ext]);
  lua_pushnumber(L,p);
  return 1; }

```

Next, we register both functions in the file `mfluaini.lua` as, respectively, `LUAGLOBALGET_char_code` and `LUAGLOBALGET_char_ext` for the Lua interpreter, so every Lua function can use them:

```

int mfluainitialize()
{ lua_State *L = Luas[0];
  /* register lua functions */
  ...
  lua_pushcfunction(L,
    priv_mfweb_LUAGLOBALGET_char_code);
  lua_setglobal(L, "LUAGLOBALGET_char_code");
  lua_pushcfunction(L,
    priv_mfweb_LUAGLOBALGET_char_ext);
  lua_setglobal(L, "LUAGLOBALGET_char_ext");
  ...
  return 0; }

```

In this way we can make available any Pascal WEB macro, procedure, function, variable, etc.; for example, the `info` field of a memory word

```

/* @d info(#) == mem[#].hh.lh */
/* {the |info| field of a memory word} */
static int priv_mfweb_info(lua_State *L)
{ halfword p,q;
  p = (halfword) lua_tonumber(L,1);
  q = mem [p ].hhfield.v.LH ;
  lua_pushnumber(L,q);
  return 1; }

```

which becomes available for Lua as `info`:

```

int mfluainitialize()
{ lua_State *L = Luas[0];
  /* register lua functions */
  ...
  lua_pushcfunction(L, priv_mfweb_info);
  lua_setglobal(L, "info");
  ...
  return 0; }

```

Of course it's best to use a minimum set of sensors.

### 2.3 Direct translation of a WEB procedure

Pascal WEB and Lua are not so different and we can easily translate from one to another. For example, the WEB procedure `print_scaled`

```
@<Basic printing...@>=
procedure print_scaled(@!s:scaled);
{prints scaled real, rounded to five digits}
var @!delta:scaled;
{amount of allowable inaccuracy}
begin if s<0 then
  begin print_char("-"); negate(s);
    {print the sign, if negative}
  end;
print_int(s div unity);
{print the integer part}
s:=10*(s mod unity)+5;
if s<>5 then
  begin delta:=10; print_char(".");
  repeat if delta>unity then
    s:=s+@'10000-(delta div 2);
    {round the final digit}
    print_char("0"+(s div unity));
    s:=10*(s mod unity);
    delta:=delta*10;
  until s<=delta;
  end;
end;
```

can be translated to Lua as

```
function print_scaled(s)
  local delta
  local res = ''; local done
  if s== nil then
    print("\nWarning from print_scale
    in mflua.ini: s is nil");
    return res; end
  if s<0 then
    res = '-'; s = -s
  end
  res = res .. print_int(math.floor(s/unity))
  -- {print the integer part}
  s=10*(math.mod(s,unity))+5
  if s ~= 5 then
    delta=10; res = res .. '.'
    done = false
  while not done do
    if delta>unity then
      s=s+half_unit-(math.floor(delta/2))
      -- {round the final digit}
    end
    res = res .. math.floor(s/unity);
    s=10*math.mod(s,unity);
    delta=delta*10;
    if s<=delta then done = true end
  end;
  end
  return res
end
```

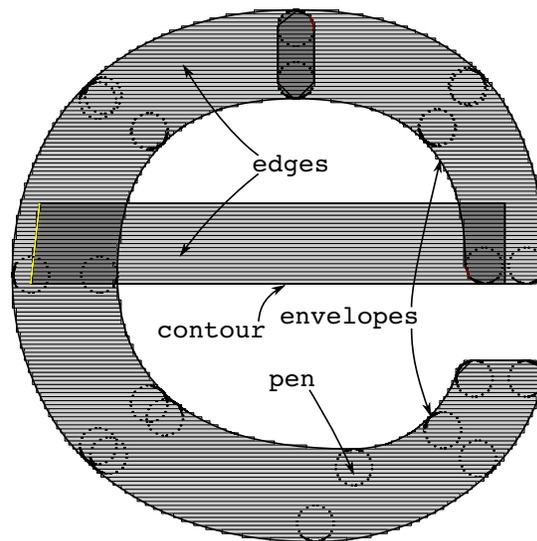
### 3 Collecting data

To properly draw the outline of a glyph we need the following information:

1. the edge structures, i.e. the pixels of the picture;
2. the paths from the filling of a contour;
3. the paths from the drawing of an envelope with a pen;
4. the pen used in drawing an envelope.

In fig. 1 we can see these components for the lower case 'e' of Concrete Roman at 5 point.

Figure 1: The components of a glyph.



To store the edge structures we put one sensor into the procedure `ship_out(c: eight_bits)` that outputs a character into `gf_file`:

```
procedure ship_out(@!c: eight_bits);
...
mflua_printedges(" (just shipped out)",
  true,x_off,y_off);
if internal[tracing_output]>0 then
  print_edges(" (just shipped out)",
    true,x_off,y_off);
end;
```

The Lua implementation is the function `print_edges(s,nuline,x_off,y_off)` in `print_edges.lua` and it is the direct translation of the WEB `print_edges`:

```
function print_edges(s,nuline,x_off,y_off)
  print("\n... Hello from print_edges! ...")
  local p,q,r -- for list traversal
  local n=0 -- row number
  local cur_edges = LUAGLOBALGET_cur_edges()
  local y = {}; local xr = {}; local xq = {}
  local f, start_row,
    end_row ,start_row_1, end_row_1
  local edge
```

```

local w,w_integer,row_weight,xoff
local chartable = mflua.chartable
local index; local char
p = knil(cur_edges)
n = n_max(cur_edges)-zero_field
while p ~= cur_edges do
  xq = {}; xr = {}
  q=unsorted(p); r=sorted(p)
  if(q>void)or(r~=sentinel) then
    while (q>void) do
      w, w_integer,xoff = print_weight(q,x_off)
      xq[#xq+1] = {xoff,w_integer}
    end
    while r ~= sentinel do
      w,w_integer,xoff = print_weight(r,x_off)
      xr[#xr+1]= {xoff,w_integer}
    end
    y[#y+1] = {print_int(n+y_off),xq,xr}
  end
  p=knill(p); n=decr(n);
end
-- local management of y, xq, xr
--f = mflua.print_specification.outfile1
index=(0+print_int(LUAGLOBALGET_char_code()))
  +(0+print_int(LUAGLOBALGET_char_ext()))*256
char = chartable[index] or {}
print("#xq=".. #xq)
for i,v in ipairs(y) do
  xq,xr = v[2],v[3]
  -- for j=1, #xq, 2 do end ??
  row_weight=0
  for j=1, #xr, 1 do
    local xb = xr[j][1]; local xwb = xr[j][2]
    row_weight=row_weight+xwb
    xr[j][3]=row_weight
  end
end
char['edges'] = char['edges'] or {}
char['edges'][#char['edges']+1]=
  {y,x_off,y_off}
...
return 0
end

```

As we already said, a Lua script is stateless during its lifetime, but this doesn't mean that we can't store global variables: it suffices to set up the global data by means of a sensor that is placed in the main program just before the sensors that need the global data. By convention, the global data are placed in the file `mflua.ini.lua`: they have the namespace `mflua` (as in `mflua.chartable` which collects the pixels) or the prefix `LUAGLOBAL` (as in `LUAGLOBALGET_char_code()` that we have seen previously). Also `mflua.ini.lua` hosts some functions like `print_int(n)` (print an integer in decimal form, directly translated from WEB to Lua) and aliases like `knill=info`.

The sensors for the contours and the envelope are more complicated. It's not easy to find the optimal point where to insert a sensor, and it's compulsory to have the book *The METAFONTbook* [2] at hand (and of course also [1]). In this case the starting point is the procedure `do_add_to` where METAFONT decides, based on the current pen, to fill a contour (`fill_spec`) or an envelope (`fill_envelope`); we can hence insert a couple of sensors before and after these two points:

```

procedure do_add_to:
if max_offset(cur_pen)=0 then
  begin mfluaPRE_fill_spec_rhs(rhs);
    fill_spec(rhs);
    mfluaPOST_fill_spec_rhs(rhs);
  end
else
  begin mfluaPRE_fill_envelope_rhs(rhs);
    fill_envelope(rhs);
    mfluaPOST_fill_envelope_rhs(rhs);
  end;
if lhs<>null then
  begin rev_turns:=true;
    lhs:=make_spec(lhs,max_offset(cur_pen),
      internal[tracing_specs]);
    rev_turns:=false;
    if max_offset(cur_pen)=0 then
      begin mfluaPRE_fill_spec_lhs(lhs);
        fill_spec(lhs);
        mfluaPOST_fill_spec_lhs(lhs);
      end
    else
      begin mfluaPRE_fill_envelope_lhs(lhs);
        fill_envelope(lhs);
        mfluaPOST_fill_envelope_lhs(lhs);
      end;
    end;
  ...
end;

```

Both `fill_spec` and `fill_envelope` have in turn another couple of sensors:

```

procedure fill_spec(h:pointer);
...
  mflua_PRE_move_to_edges(p);
  move_to_edges(m0,n0,m1,n1);
  mflua_POST_move_to_edges(p);
...
end
procedure fill_envelope(spec_head:pointer);
...
  mfluaPRE_offset_prep(p,h);
  {this may clobber node |q|, if it
    becomes ''dead''}
  offset_prep(p,h);
  mfluaPOST_offset_prep(p,h);
...
end

```

We will not show the Lua code here; we have followed the same strategy of the edge structures and stored the data in the global table `mflua.chartable`. The data are Bézier curves  $\{p, c_1, c_2, q, \text{offset}\}$  which corresponds to the METAFONT path `p .. controls c1 and c2 .. q` shifted by `offset`.

For each character `char = mflua.chartable[j]` we have available `char['edges']`, `char['contour']` and `char['envelope']` (the latter with its pen) for the post-processing.

#### 4 The outlines of the glyphs

Up to this point, things have been relatively easy because, after all, we have been following the completely commented Pascal WEB code. The post-processing phase is easy to explain but more heuristic.

Briefly, for each curve we check (using the table `char['edges']`) if it is on the frontier of the picture and cut the pieces that are inside or outside. The problems stem from the fact that, by cutting a path, we are left with pending (pendent, drooping) paths that possibly should be removed; also we must have a robust algorithm to compute the intersection between two Bézier curves.

If we put the sensor `mflua_end_program` just before the end of the program, we can process the data collected so far. The script `end_program.lua` executes the function `end_program()` that aims to extract the contour and append it as a MetaPost path to the file `envelope.tex`. We can describe the strategy as a sequence of three phases: preparation, compute the intersections, remove unwanted paths.

##### 4.1 Preparation

If we remove the pixels in fig. 1 we can see the contours, the envelopes and the pens (see fig. 2). Currently for a pen we will consider the polygonal closed path that joins the points.

The goal of this phase is to decide when a point of a path is inside the picture and then split the path to remove its subpaths that are inside the picture. The main tool is the de Casteljau algorithm (see, for example [4]): given a Bézier curve  $C = \{(p, c_1, c_2, q), t \in [0, 1]\}$ , place  $b_0 = p, b_1 = c_1, b_2 = c_2, b_3 = q$ , the de Casteljau algorithm is expressed by the recursive formula

$$\begin{cases} b_i^0 = b_i \\ b_i^j = (1-t)b_i^{j-1} + tb_{i+1}^{j-1}, \end{cases}$$

for  $j = 1, 2, 3$  and  $i = 0, \dots, 3 - j$ . For a fixed  $t = t_1$  we have

Figure 2: The components of a glyph, without pixels.

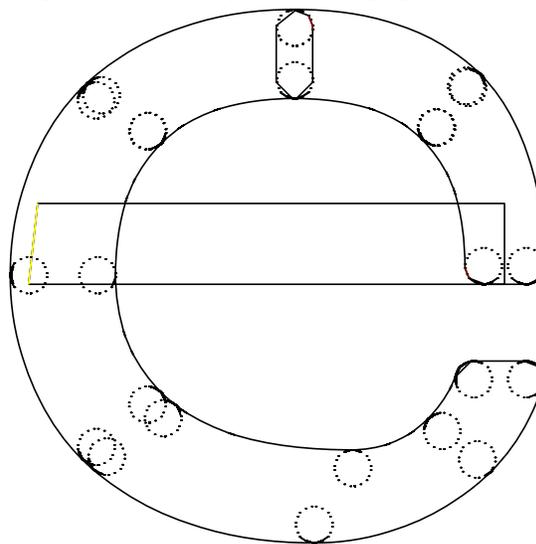
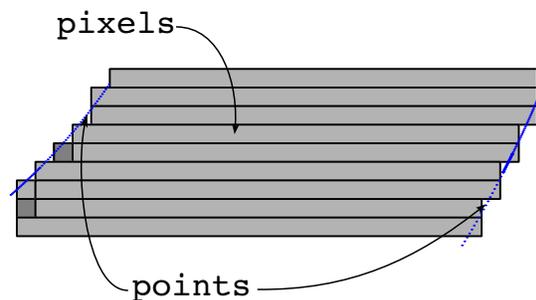


Figure 3: Points (very tiny) on the frontier and pixels.



$$\begin{matrix} b_0^0 & b_1^0 & b_2^0 & b_3^0 \\ b_0^1 & b_1^1 & b_2^1 & \\ b_0^2 & b_1^2 & & \\ b_0^3 & & & \end{matrix}$$

where  $b_0^3$  is the point on  $C$  at the time  $t_1$ ,

$$C_{\text{left}} = \{(b_0^0, b_0^1, b_0^2, b_0^3), t \in [0, t_1]\},$$

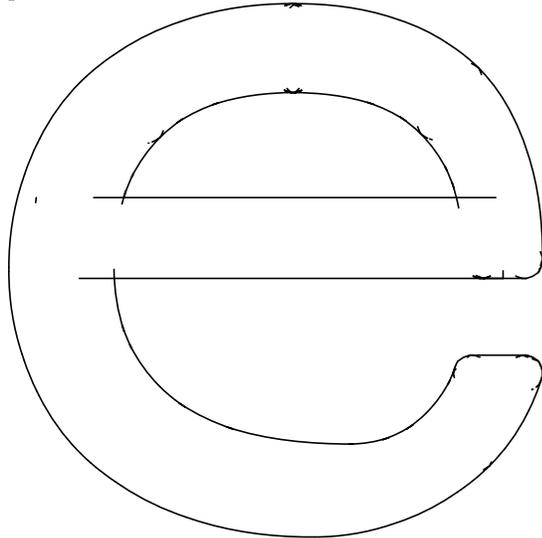
and

$$C_{\text{right}} = \{(b_0^3, b_1^2, b_2^1, b_3^0), t \in [t_1, 1]\}.$$

The Lua function `bez(p, c1, c2, q, t)` in `end_program.lua` is the immediate translation of the de Casteljau algorithm and returns `b30[1], b30[2], b00, b10, b20, b30, b21, b12, b03` where  $x = b30[1]$  and  $y = b30[2]$  are the coordinates of the point at time  $t$ .

The critical issue is to decide when a point is black and it's not on the frontier; as we can see in fig. 3, some points on the frontier are white and some points are black, so for each one we need to compute its weight and the weight of its closest neighbors, if all of them are black, then the point is black and

**Figure 4:** The components of a glyph, after the first phase.



inside the picture (otherwise it is on the frontier or outside).

Another problem is that we want a given path to have “good” intersections with other paths: if we are too strict we can erroneously mark a point as not internal—and hence we can lose an intersection—and if we are too tolerant we can have useless intersections (i.e. intersections that are internal) and the next phase is unnecessarily loaded.

These are the steps followed in this phase:

1. associate with each path a set of time intervals that describes when the subpath is not internal;
2. adjust each interval to ensure proper intersections;
3. split each path in  $C_{\text{left}}$  and  $C_{\text{right}}$  that is not completely internal.

In fig. 4 we can see the result: there are some small isolated paths that are internal, but we can easily remove them in the subsequent phases. Also note the effect of the non-linearity of a Bézier curve: we adjust the intervals with the same algorithm for both straight lines and semicircular lines—but the result cannot be the same.

## 4.2 Compute the intersections

Given that METAFONT can calculate the intersections between two paths, it’s natural to use its algorithm, but its translation in Lua or via Web2C is not cheap. It’s better to write, for each  $\text{path}_i$  and  $\text{path}_j$ , a simple METAFONT program like this one for  $i = 2$  and  $j = 1$ :

```
batchmode;
message "BEGIN i=2,j=1";
```

```
path p[];
p1:=(133.22758,62) ..
    controls (133.22758,62.6250003125)
    and (133.22758,63.250000800781)
    .. (133.22758,63.875001431885);
p2:=(28.40971260273,62) ..
    controls (63.349007932129,62)
    and (98.28829,62)
    .. (133.22758,62);
numeric t,u; (t,u) = p1 intersectiontimes p2;
show t,u;
message "" ;
```

After running MFLua on this, the log

```
This is METAFONT, Version 2.718281 [...]
**intersec.mf
(intersec.mf
BEGIN i=2,j=1
>> 0
>> 1
```

can be easily parsed with Lua.

The number of intersections can be quite large even if  $\text{path}_i \cap \text{path}_j = \text{path}_j \cap \text{path}_i$  and, if we have  $n$  paths, we compute only  $\frac{n(n-1)}{2}$  intersections. For example, the lower case letter ‘s’ of the Concrete Roman at 5 point has 207 paths, and on an Intel Core Duo CPU T7250 2GHz with 2 GByte, computing all the 21321 intersections took around 2 seconds—which was low enough to avoid re-implementing an intersection algorithm. There is an important point to understand here: we run MFLua inside another instance of MFLua by means of the Lua function `os.execute(command)`, hence we must carefully manage shared resources (i.e. intermediate files for output such as `envelope.tex`) by means of synchronization on the filesystem.

## 4.3 Remove unwanted paths

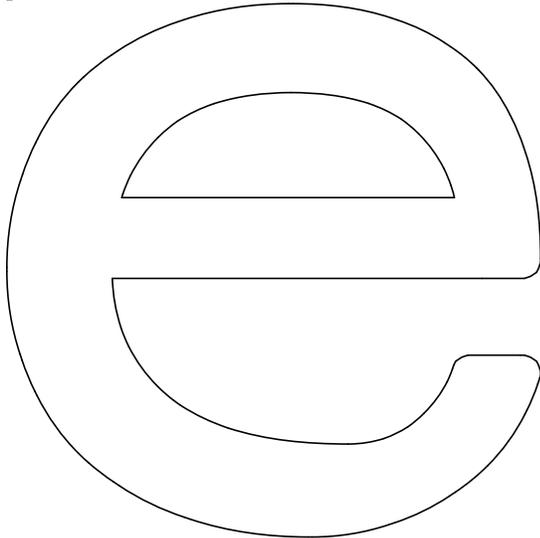
The last phase is the more heuristic one. The strategy is to gradually clean up the outlines by identifying a rule for the paths to be removed and implementing it with a Lua function. The common data structures are the set of paths `valid_curves`, the set of intersections for each path `matrixinters` and the set of pen paths `valid_curves_p_set`. Every time a curve is deleted these sets must be updated.

Here is a small example of the rules:

```
-- remove isolated paths
valid_curves, matrixinters =
_remove_isolate_path(valid_curves,matrixinters)

-- remove duplicate paths
valid_curves, matrixinters =
_remove_duplicate_path_I(valid_curves,
                        matrixinters)
```

**Figure 5:** The components of a glyph, after the last phase.



```
-- try to remove pen paths outside
-- the edge structure
valid_curves,matrix_inters =
  _open_pen_loop_0(valid_curves,
                   matrix_inters,
                   valid_curves_p_set,char)

-- try to remove duplicate pen paths
valid_curves,matrix_inters =
  _remove_duplicate_pen_path(valid_curves,
                             matrix_inters,
                             valid_curves_p_set)
```

Some rules are very specific, such as the following one, which takes care of a missing intersection for the letter ‘y’ (probably due to an erroneous set of time intervals):

```
-- a fix for an error found on ccr5 y
valid_curves,matrix_inters =
  _fix_intersection_bug(valid_curves,
                       matrix_inters)
```

and hence they are potentially useless for other glyphs. There are about twenty rules; after their incorporation the results are the outlines of fig. 5.

Figures 6, 7, 8 and 9 on the following page are a little gallery of results with these sets of rules.

## 5 Conclusions

MFLua shows that it’s possible to get the original outlines of a METAFONT glyph without advanced mathematical techniques and tracing algorithms. However, in attempting an automatic conversion of a METAFONT source into an OpenType font there are so many details to fix that it’s not opportune to focus on this for a next release. Here are some more immediate goals:

1. The sensors must go in a change file `mflua.ch` and not in `mf.web`.
2. MFLua should be buildable for Windows.
3. The function `end_program()` must be simplified; we need to test other METAFONT sources.
4. Some features remain to be implemented; for example, a better approximation for an elliptical pen (see fig. 8) and errors to fix as in fig. 9.
5. Perhaps the Lua scripts should use `kpathsea`.

The Lua code needs to be made more consistent for both variable names and the use of tables as arrays or hashes (some bugs resulting from the misunderstanding of indexes as integers rather than strings).

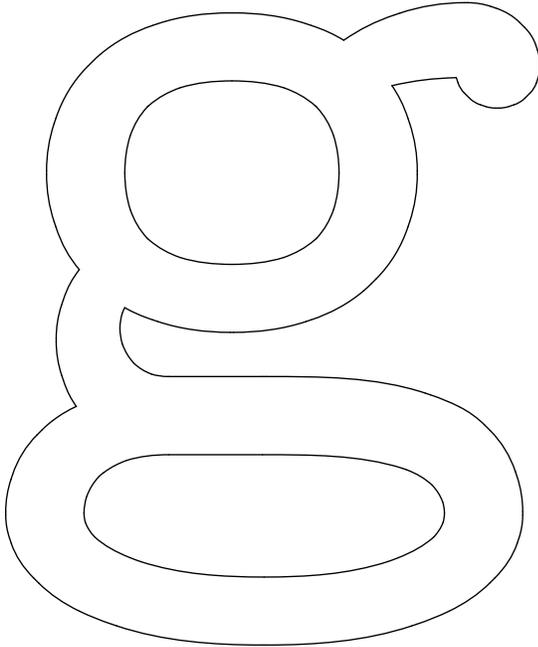
The source code will be available for the next (XIX<sup>th</sup>) BachTeX meeting in Bachotek, Poland.

## References

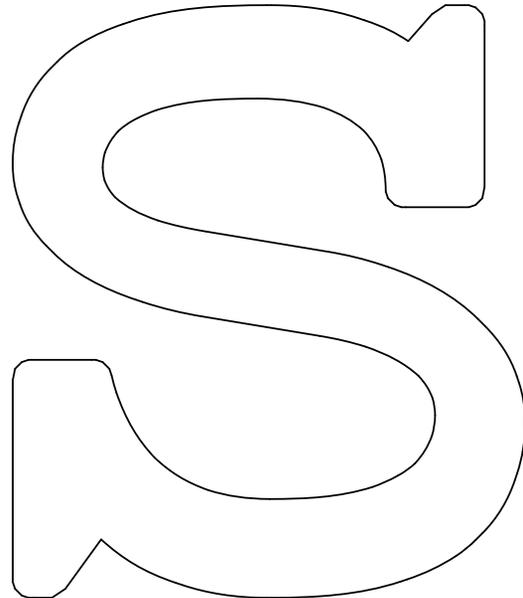
- [1] Donald E. Knuth, *Computers & Typesetting, Volume C: The METAFONTbook*. Reading, Massachusetts: Addison-Wesley, 1986. xii+361pp. ISBN 0-201-13445-4
- [2] Donald E. Knuth, *Computers & Typesetting, Volume D: METAFONT: The Program*. Reading, Massachusetts: Addison-Wesley, 1986. xviii+566pp. ISBN 0-201-13438-1
- [3] R. Ierusalimschy, *Programming in Lua*, 2nd ed. Lua.org, March 2006. Paperback, 328pp. ISBN 13 9788590379829 <http://www.inf.puc-rio.br/~roberto/pil2>.
- [4] D. Marsh, *Applied Geometry for Computer Graphics and CAD*, 2nd ed. Springer Undergraduate Mathematics Series, 2005. xvi+352pp. ISBN 978-1-85233-801-5

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

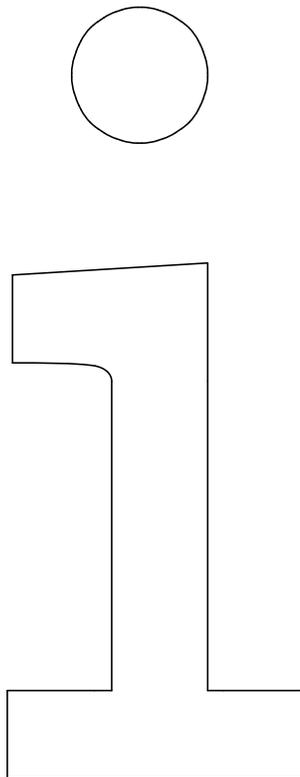
**Figure 6:** The ‘g’ of Concrete Roman at 5 point.



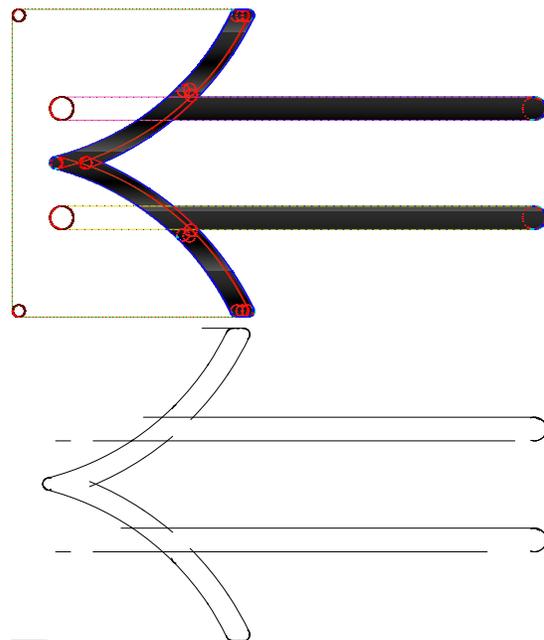
**Figure 8:** The ‘s’ of Concrete Roman at 5 point. Note the approximations of the polygonal pen of upper and lower barb.



**Figure 7:** The ‘i’ of Concrete Roman at 5 point.



**Figure 9:** The ‘Double leftward arrow’ of Computer Modern Math Symbols 10 point. An error of the time intervals breaks the contours.



## Macro interfaces and the *getoptk* package

Michael Le Barbier Grünewald

### 1 Introduction

We present the *getoptk* macro package for the *plain* format. It eases the definition of macros whose interface is similar to the one used by  $\TeX$  primitives such as `\vrule` or `\hbox`. We discuss some characteristics of interface styles and a short classification of these before describing our package. The implementation of the `\readblanks` macro, a variant of the primitive `\ignorespaces` triggering a callback, seems to us especially interesting.

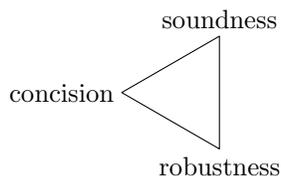
The *getoptk* macro package is similar to *xkeyval*, in that it allows optional arguments to be specified as a dictionary. However, it avoids the introduction of a new syntactic construction for the concrete form of the dictionary. Instead, it tries to imitate the convention used by `\vrule` and similar  $\TeX$  primitives.

### 2 Literate programming

We use Norman Ramsey's NOWEB [5] literate programming tool to present our code. A file is split up in *chunks*, each of which is given an identifier that we will always write in italics; for example, *Definition of getoptk*. In this text, chunk names that are not also file names are capitalised; although this is not conventional English syntax, it helps recognising a chunk name as such in the text.

### 3 Characteristics of interfaces

The main characteristics of macro interfaces are organised around the three ideas of *concision*, *robustness* and *soundness*, which in turn are the three vertices of the following *tension triangle*:



The idea of concision expresses itself in interfaces encouraging a terse and short way to type in a manuscript. Robustness is the ability of an interface to perform well in a wide range of contexts, especially nested calls. Sound interfaces mix well in the manuscript and do not break its homogeneity. They are easy to memorise and help in having a nice looking manuscript.

These three ideas are distinct: The `\proclaim` macro in *plain T $\TeX$*  and the modal behaviour of the *letter* example format described in *The T $\TeX$ book* [3] both put an emphasis on concision but break ro-

bustness and soundness. The `verbatim` environment in  $\LaTeX$  is sound and concise but not robust. If robustness is needed in a `verbatim` typesetting job, we may rely on the usual markup and named characters to input our data: we get a sound and robust answer to our problem, but this lacks concision.

*Remarks:*

1. Lexical analysis techniques can be used to build concise macro interfaces, often at the price of robustness. This loss is easily circumvented by cleanly separating the lexical analysis part and the processing part of the macro job.
2. Concise interfaces help in producing a text that is easy to maintain. The maintainability burden caused by the lack of concision of an interface can sometimes be ameliorated by a third party tool — *e.g.*, the code pretty-printer included in the literate programming tool WEAVE [4] — used to automatically generate parts of the text using a non-concise macro set.
3. Sound interfaces make life easier for third party software processing  $\TeX$  manuscripts.

### 4 Bestiary

Let us quickly review different styles of familiar interfaces and means to define macros using them. Note that many macros do not use purely one of the styles of interface described below, but rather a combination of them.

#### 4.1 Simple

In the *simple* interface style, the control sequence is followed by its arguments, each one being either a token or a group. Macros using the simple interface style are defined by the `\def` primitive:

```
\def\example#1#2{%
  This replacement text uses #1 and #2.
}
```

#### 4.2 Delimited

Macros using the *delimited* interface style take advantage of the ability of `\def` to be given somewhat arbitrary argument delimiters. This feature lets the macro usage blend smoothly in the surrounding text. A popular example is the `\proclaim` macro defined in *plain*:

```
\proclaim Theorem 1. {\TeX} has a powerful
  macro capability.\par
```

This macro has two delimited arguments, one starting right after the `\proclaim` control sequence and running to the first dot on the line, the second starting after the point and ending with the current paragraph, signalled by a double carriage return or an

explicit `\par` as in the example above. If `\proclaim` were designed to use the simple interface style, the previous usage example would have look like this:

```
\proclaim{Theorem 1}{\TeX} has a powerful
macro capability.}
```

### 4.3 Register

The *register* interface style relies on registers and control sequences instead of formal arguments to pass informations to the macro. With a hypothetical implementation of `\proclaim` using this interface style, the previous example could be:

```
\def\proclaimlabel{Theorem 1}
\def\proclaimtext{\TeX} has a powerful
macro capability.}
\proclaim
```

This use of *global variables*, as it is often referred to in classical programming languages, usually breaks the ability of a macro to support nested calls. This is not always a problem in  $\TeX$ , where modifications of registers can be made local to a group. There is a realm where this use of global variables is often the rule: machine level programming. Indeed, many BIOS or OS functions on PCs are serviced through software interruptions. In the typical case, the registers of the machine are assigned values corresponding to the parameters of the call, and the interruption is then triggered.

Some interactions with the typesetting engine  $\TeX$  are achieved through the use of dedicated registers. Using a register style for the interface of a macro may give the user a feeling he is interacting with  $\TeX$  as a machine. This style may be appropriate for font selection schemes and other “system services”. The main macro of our package partially uses this interface style.

There is no special provision needed to define a macro using such an interface, though some packages, including *getoptk*, provide the user with specialised macros used to set the values of the registers.

### 4.4 Keyword

The  $\TeX$  primitives `\hrule`, `\vrule`, `\hbox`, `\vbox` and `\vtop` use a special interface style that we call the *keyword* interface style. A typical call to the `\hrule` primitive is:

```
\hrule width 12pt depth 2pt height 10pt
```

Each parameter to the call is introduced by a keyword, then comes the actual parameter associated to the keyword. Keywords have no fixed order and it is possible to repeat the same keyword more than once or to omit some or all of them. It is a very flexible

way to pass arguments to macros, similar to *labels* in the OCaml programming language. Unluckily, there is no facility in  $\TeX$  itself or in *plain* to define macros using this interface style. The second part of this paper presents such a facility. A close variant of this style is the *keyval* style discussed hereafter and whose popularity among  $\LaTeX$  hackers is increasing.

### 4.5 Starred

Macros having a *starred* variant are well known to  $\LaTeX$  users. Structure domain related macros, such as `\chapter` or `\section`, usually have a starred variant whose behaviour is similar to the original version but does not produce an entry in the table of contents of the document or receive a section number. The use of a macro using this interface style and the simple interface style is illustrated by the following line of code:

```
\section*{Introduction}
```

Starred variants of macros are supported by the preferred  $\LaTeX$  methods for creating new macros. Any macro defined by `\newcommand` can use the pseudo-predicate `\@ifstar` to check for itself being called with a star or not.

### 4.6 Bracket

A common feature found in interfaces to macros defined in  $\LaTeX$  is the use of a bracketed optional argument. We call this the *bracket* style interface. The `\cite` macro defined by  $\LaTeX$  uses this interface style and the simple one, as illustrated by:

```
\cite[Theorem~1]{TEXBOOK}
```

The definition of macros using this style of interface is supported in  $\LaTeX$  by `\newcommand`, where the `\cite` command used above could have been (but was not) defined by

```
\newcommand{\cite}[] [1]{...}
```

### 4.7 Keyval

The *keyval* interface style is named after a popular  $\LaTeX$  package *keyval* by David Carlisle [2] and its successor *xkeyval* by Hendri Adriaens [1].

Macros using this interface style allow options of the form `key=value`; a sample use is:

```
\mybox[text=red,left=5pt]{some text}
```

It is easy to define macros using this interface style with the *xkeyval* package, which is available to *plain* and  $\LaTeX$  users. This interface is probably unsound in a *plain*  $\TeX$  document but may fit well in a  $\LaTeX$  document, since the notation it uses is reminiscent of the one used for package arguments.

## 5 Presentation of the package

We now describe the interface and the implementation of the *getoptk* package. Our goal is to provide users of *getoptk* with an easy way to define macros using the *keyword interface style* described above. This style is a very flexible way to pass arguments to macros and already used by T<sub>E</sub>X primitives. The use of this style therefore favours *soundness* of the interface.

Rather than presenting a formal specification of our macros, let us take a look at an example of utilisation and use that as a basis to discuss the features and the operation of the package.

### 5.1 Usage example

We show how to use *getoptk* to define `\example`, a macro having a mandatory argument and accepting optional arguments in the *keyword interface style*.

```
<example.tex> ≡
\input getoptk
\catcode'\@=11
<Dictionary definition>
<Main definition>
<Usage>
<Usage equivalence>
\bye
```

We need first to read the *getoptk* package. In the code chunk *Dictionary definition* we require the creation of a new optional argument dictionary that we fill with bindings between keywords and behaviours. In the following, we refer to a dictionary of this kind as a *behaviour dictionary*. We then define the `\example` command itself where the magic happens in *Main definition* and add a few examples in *Usage* and their replacement text after the call to `\getoptk` in *Usage equivalence*.

Note that we use private names containing `@` in this example, thus the example starts with the familiar `\catcode` mantra.

We require the creation of a fresh behaviour dictionary — a data structure represented by a control sequence — with `\newgetoptkdictionary` to bind keywords to behaviours. In this example, the binding operations are performed by the commands

```
\defgetoptkflag,
\defgetoptkbracket and
\defgetoptkcount.
```

The bindings are written in the dictionary associated with `example` because it is the last one created. (It is possible to choose another dictionary with `\setgetoptkdictionary`.) The binding macros mix the register interface style and the simple one. This is convenient because this avoids repeating

the argument `example` each time a binding macro is called and there is no plausible usage scenario involving nasty nested calls.

```
<Dictionary definition> ≡
\newgetoptkdictionary{example}
\defgetoptkflag{alpha}{(alpha)}
\defgetoptkflag{beta}{(beta)}
\defgetoptkcount{gamma}{(gamma #1)}
\defgetoptkbracket{delta}{%
\ifgetoptkbracket
(delta "#1")%
\else
(delta)%
\fi}
```

Each binding macro has two arguments: a *keyword*, that consists of a sequence of catcode 11 tokens, and a *behaviour*, a valid replacement text for a macro. The binding macro arranges things so that each occurrence of the keyword seen in the call to `\example` triggers the evaluation of the corresponding behaviour. Before we give more details on this triggering mechanism and the semantics of the binding, let us look at the definition of `\example`:

```
<Main definition> ≡
\def\example{%
\setgetoptkdictionary{example}%
\getoptk\example@M}
\def\example@M#1#2{%
\par\noindent[{\tt #1}][#2]}
```

We see that the definition of `\example` is basically a call to `\example@M`, supervised by `\getoptk`. The task of `\getoptk` is to look for keywords on the input stream and aggregate the corresponding behaviours and arguments. The resulting aggregate is then given as the first argument to `\example@M`. Before we pass control to `\getoptk`, we first use `\setgetoptkdictionary` to activate the behaviour dictionary defined above.

Returning to the above *Dictionary definition*, the two calls to `\defgetoptkflag` bind the keywords `alpha` and `beta` with the behaviours `(alpha)` and `(beta)`. These are saved as the replacement texts of `\getoptk@behaviour@example@alpha` and `\getoptk@behaviour@example@beta`. Given this, `\getoptk` arranges things so that the sequence:

```
<Usage> ≡
\example beta alpha {omega}
```

expands to:

```
<Usage equivalence> ≡
\example@M{%
\getoptk@behaviour@example@beta
\getoptk@behaviour@example@alpha
}{omega}
```

The call to `\defgetoptkcount` binds `gamma` to the behaviour (`gamma #1`) but also notes that the `gamma` keyword must be followed by an integer — a valid right-hand-side for `count` registers. This integer will replace the formal paragraph `#1` when behaviours are triggered.

The last binding of our example is performed by `\defgetoptkbracket`, that defines a keyword admitting an optional bracketed argument. As illustrated by our example, the behaviour uses the predicate `\ifgetoptkbracket` to test for the presence of an optional argument. This is a true predicate created by the `\newif` command. The sequence

```
<Usage> +≡
  \example gamma 2 delta [10] {omega}
```

then expands to

```
<Usage equivalence> +≡
  \example@M{%
    \getoptk@behaviour@example@gamma{2}%
    \getoptkbrackettrue
    \getoptk@behaviour@example@delta{10}%
  }{omega}
```

The `\getoptk` command is generous in accepting white space. In the following example, both calls to `\example` are expanded the same way.

```
\example delta [2]gamma10beta{omega}
\example delta [2] gamma
  10 beta {omega}
```

The `getoptk` package provides more binding macros, reading dimensions or tokens, and it is also possible to create new ones (6.6).

## 5.2 Criticism of the interface

We criticise the interface of a macro using `\getoptk` to get its optional arguments, in view of the three characteristics we isolated in the introduction:

**soundness** holds, because the interface mimics the behaviour of some  $\TeX$  primitives;

**concision** is as respected as it can, but the interface to a macro admitting a large number of optional arguments cannot be that concise;

**robustness** seems to hold, and it is also possible to circumvent the direct use of `\getoptk` and directly construct the resulting call, as demonstrated by the *Usage equivalence* chunks above.

## 6 Implementation

We dive here into the deepest part of the job.

### 6.1 Overview

There are three important parts in the implementation. The *Definition of getoptk* and the elaboration

of dedicated *Lexical analysis procedures* will almost entirely capture our attention. It is also useful to define macros manipulating *Behaviour dictionaries*: the techniques used there are very similar to those used in list processing [3, p. 378] and we will not give many details. In these three parts, there are a few short macro definitions that may have a general usefulness, we gather them in *Ancillary definitions*.

```
<getoptk.tex> ≡
  \catcode'\@=11
  <Ancillary definitions>
  <Lexical analysis procedures>
  <Definition of getoptk>
  <Behaviour dictionaries>
  \catcode'\@=12
```

The very special nature of  $\TeX$  programs forbids the use of literate programming techniques to describe the flow of the procedure. We use instead an imperative pseudo-code notation, where  $d$  stands for the active behaviour dictionary and  $c$  is the argument given to `\getoptk`, the callback taking control of the execution flow after `\getoptk` completes its task.

---

### Algorithm 1 Workflow of *getoptk*

---

```
x ← ∅ {accumulator}
f ← true {loop flag}
while f do
  k ← <incoming tokens>
5: if k is bound in d then
  b ← behaviour of k
  a ← argument of k
  stack b and a on x
else
10: f ← false
  apply c to x
  process tokens in k
end if
end while
```

---

The work needed to implement this simple procedure falls in three categories. First, we have to manipulate data structures. The easiest way to do this is to use registers to store arguments and output of procedures manipulating the data structure. There is no special difficulty in this lengthy task. Second, structured programming in  $\TeX$  is always an involved task, since the decision parts of the code have to put the bits whose processing they require on the stream of incoming tokens. We end up with many short macros mutually calling themselves. Again, there is no real difficulty here but rather a code organisation problem. Third, there are a few tricks in the lexical analysis techniques used (6.7).

## 6.2 Ancillary definitions

We use many short macros whose definition can be found in *The T<sub>E</sub>Xbook*, such as `\gobble`:

```
⟨Ancillary definitions⟩ ≡
  \def\gobble#1{}
  ⟨More ancillary definitions⟩
```

We also need `\cslet`, `\elet` and `\csdef` but omit the definition of these classical macros here. Instead we proceed to the definition of `\tokscat` used later in the text to concatenate two token registers into a third, as in

```
\tokscat\toks0 &\toks2\to{\toks0}
```

Note the space after the first occurrence of the character 0, it is mandatory to put a space there if you do not use a named token register.

```
⟨Ancillary definitions⟩ +≡
  \def\toksloadcsexpansion#1\to#2{%
    #2=\expandafter{#1}}
  \def\tokscat#1&#2\to#3{%
    \beginnext
    \edef\tokscat@a{\the#1\the#2}%
    \toks2={#3}%
    \toksloadcsexpansion\tokscat@a
    \to{\toks4}%
    \edef\next{\the\toks2={\the\toks4}}%
    \endnext}
```

This control sequence is similar to `\concatenate` [3, p. 378] concatenating two lists.

## 6.3 Description of behaviour dictionaries

A behaviour dictionary is a list of triples represented like this:

```
\\{\{keyword}\}{\parser}\{behaviour}\}
```

We already discussed the *keyword* and *behaviour* fields (5.1) but there is a new feature. The *parser* field contains a control sequence whose job is to read the argument associated with *keyword*, removing its tokens from the input stream and storing them in a dedicated token register:

```
⟨Definition of getoptk⟩ ≡
  \newtoks\getoptkargument
```

Once the parser has completed its task, it gives control back to *getoptk* by calling `\getoptkcallback`.

The `\getoptk` macro requires that a valid behaviour dictionary be stored in `\getoptkdictionary` before it is called. The `\setgetoptkdictionary` macro can be used for this; it is defined in *Behaviour dictionaries*, together with macros used in *Dictionary definition* from the usage example (5.1).

## 6.4 Definition of entry and exit blocks

The main piece of code is divided into many small macros, whose names consist of a common prefix *getoptk* followed by the private namespace character `@` and a capital letter. This notation is inspired by assembly languages providing local labels (usually denoted by a single digit). Using this notation puts the emphasis on all these macros being private pieces of a larger entity. The letter is sometimes chosen according to the function of the code (continue, end or exit, loop, main, predicate) but most of the time, letters are simply used in sequence, from A to Z. Small letters are used for private variables. Here is the, somewhat deceiving, definition of `\getoptk`:

```
⟨Definition of getoptk⟩ +≡
  \def\getoptk#1{%
    \beginnext
    \toks0={#1}%
    \toks2={}%
    \toks4={}%
    \toks6={}%
    \getoptkargument={}%
    \getoptk@L}
```

The argument of `\getoptk` is a callback, it is saved in `\toks0`, that corresponds to *c* in Algorithm 1. The content of `\getoptkargument` and some scratch registers are erased. The register `\toks2` plays the role of the accumulator *x*. The first token of the replacement text is `\beginnext`, which has not yet been defined. As its name suggests, it has a companion macro `\endnext`:

```
⟨Ancillary definitions⟩ +≡
  \def\beginnext{%
    \begingroup
    \let\next\undefined}
  \def\endnext{%
    \expandafter\endgroup\next}
```

This kind of construction is familiar to T<sub>E</sub>X programmers using `\edef` constructs: it allows the easy opening of a group inside which we are allowed to play all kinds of register-based games and finally use `\edef` to compute an appropriate replacement text for `\next`. The exit block of our procedure is:

```
⟨Definition of getoptk⟩ +≡
  \def\getoptk@E{%
    \edef\next{%
      \the\toks0{\the\toks2}%
      \the\toks4}%
    \endnext}
```

We already know that `\toks0` holds the callback registered by the user who called `\getoptk`, and `\toks2` the material gathered so far by the whole procedure.

Register `\toks4` corresponds to  $k$  in Algorithm 1 and contains tokens that were removed from the input stream but failed to compare with a keyword bound in the active behaviour dictionary. Please take a look again at the first example of usage discussed above (5.1):

```
\example beta alpha {\omega}
```

When the `\getoptk` procedure completes it ultimately calls `\getoptk@E`. Right after the evaluation of `\edef` the replacement text of `\next` is

```
\example@M{%
  \getoptk@behaviour@example@beta
  \getoptk@behaviour@example@alpha}
```

which `\expandafter` puts again in the stream of incoming tokens, therefore replacing the original sequence `\example beta alpha`.

### 6.5 Definition of the main loop

We read the incoming tokens that may stand for a keyword. For this, we use two custom lexical analysis procedures (6.7): `\readblanks` that discards blanks on the input stream and `\readletters` that gathers tokens with `catcode = 11` in a register.

```
<Definition of getoptk> +≡
\def\getoptk@L{%
  \readblanks\then\getoptk@A\done}
\def\getoptk@A{%
  \readletters\to\toks4\then
  \getoptk@B
  \done}
```

We now look for the keyword stored in `\toks4` in the dictionary `\getoptkdictionary`, using the classical list scanning technique described in *The T<sub>E</sub>Xbook* [3, p. 378].

```
<Definition of getoptk> +≡
\def\getoptk@B{%
  \let\getoptk@N\getoptk@E
  \let\getoptk@S
  \getoptkdictionary
  \getoptk@N}
```

The scanning macro `\getoptk@S` first unpacks its argument into a triple

```
\{\{<keyword>\}\{<parser>\}\{<behaviour>\}\}
```

The real work happens in `\getoptk@T`, which sets the value of the `\getoptk@N` callback to a value requiring the lecture of an argument to *keyword* with *parser*. Two values for the *parser* field have a special meaning: an empty value means no argument, while `[]` means a bracketed optional argument.

```
<Definition of getoptk> +≡
\def\getoptk@S#1{\getoptk@T#1}
```

```
\def\getoptk@T#1#2#3{%
  \edef\getoptk@a{\the\toks4}%
  \def\getoptk@b{#1}%
  \def\getoptk@p{#2}%
  \ifx\getoptk@a\getoptk@b
    \let\getoptk@gobble
    \toks6={}%
    \toks8={#3}%
  \def\getoptk@N{#2}%
  \def\getoptk@a{}%
  \ifx\getoptk@p\getoptk@a
    \let\getoptk@N\getoptkcallback
  \fi
  \def\getoptk@a{[]}%
  \ifx\getoptk@p\getoptk@a
    \let\getoptk@N\getoptk@O
  \fi
  \fi}
```

Each parser must end with a call to the callback `\getoptkcallback` that is in charge of aggregating behaviours and their arguments in the accumulator `\toks2`. It relies on `\tokscat` to concatenate two token registers. The `\getoptk@O` parser uses the register `\toks6` to communicate the position of `\ifgetoptkbracket` to `\getoptkcallback`. The old value of `\getoptk@p` defined in `\getoptk@T` is used to recognise the case when we did not have to gather an argument. Ultimately, we branch to the main loop `\getoptk@L` again.

```
<Definition of getoptk> +≡
\def\getoptkcallback{%
  \tokscat\toks2 &\toks6\to{\toks2}%
  \tokscat\toks2 &\toks8\to{\toks2}%
  \def\getoptk@a{}%
  \ifx\getoptk@p\getoptk@a
    \toks6={}%
  \else
    \edef\getoptk@N{%
      \toks6={%
        {\the\getoptkargument}%
      }%
    }%
  \getoptk@N
  \fi
  \tokscat\toks2 &\toks6\to{\toks2}%
  \getoptk@L}
```

### 6.6 Definition of parsers

The final step is the definition of parsers. Here is the one associated with keywords admitting an optional bracketed argument.

```
<Definition of getoptk> +≡
\newif\ifgetoptkbracket
```

```

\def\getoptk@0{%
  \readblanks\then
  \futurelet\getoptk@t\getoptk@P
  \done}
\def\getoptk@P{%
  \ifx\getoptk@t[%
    \toks6={\getoptkbrackettrue}%
    \let\getoptk@N\getoptk@Q
  \else
    \toks6={\getoptkbracketfalse}%
    \let\getoptk@N\getoptkcallback
  \fi
\getoptk@N}
\def\getoptk@Q[#1]{%
  \getoptkargument={#1}%
  \getoptkcallback}

```

We next define a meta-parser and use it to define a parser for integers. This meta-parser gives access to internal  $\TeX$  parsers associated with the various types of registers: to parse a value that is a valid right-hand-side for a *dimen* register, it must be provided a scratch *dimen* register, and so on. The value of this register is modified within a group, so any register is suitable as an argument for the meta-parser.

*(Definition of getoptk)*  $\equiv$

```

\def\getoptkmetaparser#1{%
  \def\getoptkmetaparser@r{#1}%
  \afterassignment\getoptkmetaparser@A
  #1}
\def\getoptkmetaparser@A{%
  \beginnext
  \toks2=\expandafter{%
    \getoptkmetaparser@r
  }%
  \edef\next{%
    \noexpand
    \getoptkmetaparser@B{\the\toks2}%
  }%
  \endnext}
\def\getoptkmetaparser@B#1{%
  \edef\getoptk@N{%
    \getoptkargument={\the#1}%
  }%
  \getoptk@N
  \getoptkcallback}
\def\getoptkcountparser{%
  \getoptkmetaparser{\count0 }}

```

*(Definition of getoptk)*  $\equiv$

*(More parsers)*

We similarly define

```

\getoptkdimenparser,
\getoptkskipparser and
\getoptktoksparser

```

in *More parsers*, but omit these details.

It is possible to define new parsers by following the pattern of `\getoptcallback`: use a `\beginnext/\endnext` pair to read the *argument* and then arrange for `\next` to expand to

```
\getoptkargument={argument}\getoptkcallback.
```

## 6.7 Lexical analysis procedures

We define two macros performing a simple task related to lexical analysis. The first one, `\readblanks`, has the seemingly simple task of discarding blank tokens on the input stream and triggering a callback when it finds the first non-blank token. The second one, `\readletters`, gathers in a register the largest prefix of catcode 11 tokens found in the input stream, and triggers a callback.

The  $\TeX$  primitive `\ignorespaces` does not support any callback. Thus we have to implement a macro of our own achieving this effect. It is not as easy as it seems, though at a high level, the task looks straightforward. We denote our callback by *c*:

---

### Algorithm 2 Reading white space

---

```

f ← true
while f = true do
  t ← incoming token
  if t is a space or a newline token then
    discard t
  else
    f ← false
  end if
end while
trigger c

```

---

We use `\futurelet` to scan incoming tokens and therefore need to bind a space token and a newline token to some control sequences. We can then apply `\ifx` to compare the incoming token and these control sequences. Binding these tokens to control sequences cannot be done with a simple `\let` as they would then be overlooked by  $\TeX$ . But a clever use of `\futurelet` will do:

*(Lexical analysis procedures)*  $\equiv$

```

\begingroup
\catcode'\* = 13
\def*#1{}
\global\futurelet\spacetoken*^^20\relax
\global\futurelet\newlinetoken*^^0a\relax
\endgroup

```

The main loop uses `\futurelet` to get an incoming token and test it in turn against `\spacetoken`, `\newlinetoken`, `\par` and `\input` to make the decision of exiting the loop with `\readblanks@E`, discarding a blank with `\readblanks@S` or a paragraph with

`\readblanks@I`, or expanding an `\input` command with `\readblanks@X`.

```

⟨Lexical analysis procedures⟩ +=
\def\readblanks\then#1\done{%
  \beginnext
  \def\next{#1}%
  \readblanks@L}
\def\readblanks@L{%
  \futurelet\readblanks@t\readblanks@A}
\def\readblanks@A{%
  \let\readblanks@N\readblanks@E
  \ifx\readblanks@t\spacetoken
    \let\readblanks@N\readblanks@S
  \fi
  \ifx\readblanks@t\newlinetoken
    \let\readblanks@N\readblanks@S
  \fi
  \ifx\readblanks@t\par
    \let\readblanks@N\readblanks@I
  \fi
  \ifx\readblanks@t\input
    \let\readblanks@N\readblanks@X
  \fi
  \readblanks@N}
\def\readblanks@E{\endnext}

```

The actual discard of a space token requires a small trick. An easy way to discard a general token is to use a macro ignoring its argument: this will not work here, because space tokens are ignored by  $\TeX$  as it searches the input stream for a macro argument. An assignment to a counter register will consume a space token following it: the space we want to get rid of then marks the end of a numeric constant and is discarded. We use `\afterassignment` to regain control after this.

```

⟨Lexical analysis procedures⟩ +=
\long\def\readblanks@S{%
  \afterassignment\readblanks@L
  \count0=0}

```

The two last choices, *ignore* and *expand*, are readily implemented:

```

⟨Lexical analysis procedures⟩ +=
\def\readblanks@I#1{%
  \readblanks@L}
\def\readblanks@X{%
  \expandafter\readblanks@L}

```

```

⟨Lexical analysis procedures⟩ +=
⟨Definition of readletters⟩

```

Our second analysis procedure `\readletters` gathers tokens with catcode 11 in a register and triggers a callback. It is much like `\readblanks`, a `\futurelet`-based loop. We will not reproduce it here.

## 7 Conclusion

We've surveyed macro interface styles and implemented the interface used by `\vrule`, etc., for plain  $\TeX$ . We hope this will be of use to other macro writers.

◇ Michael Le Barbier Grünewald  
 Hausdorff Center for Mathematics  
 Villa Maria Endenicher Allee 62  
 D 53 115 Bonn  
 Germany  
 michi (at) mpim-bonn dot mpg dot de

## References

- [1] Hendri Adriaens, *The xkeyval package*. 2008. <http://mirror.ctan.org/macros/latex/contrib/xkeyval>
- [2] David Carlisle, *The keyval package*. 1999. <http://mirror.ctan.org/macros/latex/required/graphics>
- [3] Donald E. Knuth, *The  $\TeX$ book*. Addison Wesley, Massachusetts. Corrected edition, 1996.
- [4] Donald E. Knuth, *The Web System of Structured Documentation*. 1983.
- [5] Norman Ramsey, *Noweb: A Simple, Extensible Tool for Literate Programming*. <http://www.cs.tufts.edu/~nr/noweb>

---

## The `cals` package: Multipage tables with decorations

Oleg Parashchenko

### Abstract

Tables are one of the most complicated parts of any typesetting or publishing system, and  $\LaTeX$  is no exception. There are a number of packages related to tables, but so far the following goal has been unreachable: to automatically typeset huge, complex and attractive tables.

The new package `cals` makes this possible.

### 1 Introduction

I use  $\TeX$  as an alternative to XSL-FO [2] for publishing XML as PDF. The customers do not care about  $\LaTeX$  restrictions and guidelines (for example, “never use vertical rules” from `booktabs` [4]); they demand their specified layout. I failed to implement their complex requirements for tables using existing  $\LaTeX$  packages and decided to write my own. The name “`cals`” comes from “`CALS Table Model`” [1], a standard for table markup in XML.

The key features are:

- huge tables
- spanned cells
- decorations
- automatic typesetting

Different table packages implement different approaches to break a table across pages; see the  $\TeX$  FAQ [8], “Tables longer than a single page”. The `cals` package typesets the current row in memory, checks if the rest of the page is enough for the row, forces a page break if required, and finally flushes the row. This way, only a bit of memory is required, and therefore tables can be long. As a downside, the widths of columns need to be provided by the user.

The  $\TeX$  code for tables is supposed to be generated automatically, therefore the syntax is not traditional and maybe not convenient for manual coding. Instead of dividing a row into cells using `&`, each cell is introduced by a named command.

The implementation of decorations is unique throughout  $\TeX$ , to my knowledge. Table rules are:

- style-driven and
- context-sensitive.

A stylesheet defines how a typical table looks. The user need only give cells; the decorations do or do not appear automatically. The width of a border depends on its location; it is different for the table frame and for the header separator line.

The `cals` package has many features, but some unusual requirements might not be supported. If

you want to make changes or just look at the implementation, the source code and support files are available at <http://github.com/olpa/tex/> in the directory `cals`.

### 2 User’s guide

This section

- provides a summary of the `cals` commands,
- shows how to use the commands, and
- suggests compatibility strategies.

A complete document `demo.tex` (`demo.pdf` [6], also on CTAN) from the package documentation contains examples of:

- a simple table,
- decoration control,
- cell spanning, and
- a multipage table inside a `multicols` environment inside a table.

#### 2.1 Summary

First, an overview of `cals` commands. Details and examples follow.

Table elements:

```
\thead, \tfoot
\tbody{\penalty-10000}
\lastrule
```

Alignment:

```
\alignL, \alignC, \alignR
\vfill
```

Padding (lengths):

```
\cals@setpadding{Ag}
\cals@paddingL, \cals@paddingT
\cals@paddingR, \cals@paddingB
\cals@setcellprevdepth{A1}
\cals@paddingD
```

Color:

```
\cals@bgcolor
```

Rules (macros as lengths):

```
\cals@cs@width, \cals@rs@width
\cals@framecs@width, \cals@framers@width
\cals@bodyrs@width
\cals@borderL, \cals@borderT
\cals@borderR, \cals@borderB
```

Hooks:

```
\cals@AtBeginCell, \cals@AtEndCell
```

Spanning:

```
\nullcell
\spancontent
```

## 2.2 Simple tables

Sample code:

```
\par
\begin{calstable}
\colwidths{{50pt}{100pt}}
\brow \cell{a} \cell{b} \erow
\brow \cell{c} \cell{d} \erow
\end{calstable}
```

a	b
c	d

Basics:

- Tables are created with the environment `calstable`.
- Column widths must be specified explicitly.
- Each row is marked by the `\brow... \erow` pair.
- Individual cells are specified by the command `\cell`.

And more specifics:

- Tables must start in vertical mode.
- Cells are `vboxes`, i.e.,  $\TeX$  uses restricted vertical mode to typeset the content.
- Changes inside `\cell{...}` are local.
- The macros `\cals@AtBeginCell` and `\cals@AtEndCell` are called at the boundaries of a cell group.
- The pair `\brow... \erow` does *not* make an implicit group. All changes are active till the end of the table.

## 2.3 Multipage tables

`Cals` tables are split over pages automatically. Such tables benefit from repeatable headers and footers, specified by the commands `\thead` and `\tfoot`.

```
\begin{calstable}
\colwidths{{50pt}{100pt}}
%
\thead{\bfseries\selectfont
\brow \cell{col1} \cell{col2} \erow
\mdseries\selectfont}
\tfoot{\lastrule\nointerlineskip
\textit{\strut Some table caption
(not implemented: PartKofN)}\par}
%
\brow \cell{r1,col1} \cell{r1,col2} \erow
\brow \cell{r2,col1} \cell{r2,col2} \erow
\brow \cell{r3,col1} \cell{r3,col2} \erow
\tbreak{Manual table break!\strut\par}
\brow \cell{r4,col1} \cell{r4,col2} \erow
\brow \cell{r5,col1} \cell{r5,col2} \erow
...1000 rows...
\end{calstable}
```

col1	col2
r1,col1	r1,col2
r2,col1	r2,col2
r3,col1	r3,col2

*Some table caption (not implemented: PartKofN)*

Manual table break!

col1	col2
r4,col1	r4,col2
r5,col1	r5,col2

... 1000 rows ...

*Some table caption (not implemented: PartKofN)*

Comments:

- `\thead` and `\tfoot` must be given before the table body.
- Small distraction: text is bold in the header and is reset before the body starts.
- `\thead` and `\tfoot` can contain any vertical material. In such a case, `\tfoot` should use the command `\lastrule` where the table ends, so the code decorates the table correctly.
- I'd like to implement "Part  $K$  of  $N$ " functionality, but can't say when it will happen.

As long as the current row plus the footer fits on the rest of the page, there is no page break. Otherwise, `cals` emits the footer, page break, the header, and only then the current row.

A manual table break can be made using the command `\tbreak{<smth>}`, where `<smth>` is what to emit between the footer and the next header. Most likely, it is `\vfill\break`.

## 2.4 Alignment

To left, center, or right-align the content of a cell, use `\alignL`, `\alignC` or `\alignR`, respectively. The default is left-alignment. To vertically align a cell to the middle or bottom, add `\vfil` or `\vfill` before the cell content.

```
\begin{calstable}
\colwidths{{60pt}{60pt}{60pt}}
\def\cals@framers@width{0.4pt}
\def\cals@framecs@width{0.4pt}
%
\brow
\alignR \cell{\vfill right, bottom}
\alignC \cell{\vfil center, middle}
\alignL \cell{left, top}
\ht\cals@current@row=50pt
\erow
\end{calstable}
```

	center, middle	left, top
right, bottom		

For demonstration purposes, the example sets the height of rows. This is an undocumented and unplanned feature. You should not rely on it.

## 2.5 Padding

Padding depends on the current font and is calculated when a table starts. If you change a font inside a table, it is a good idea to update the padding:

```
\cals@setpadding{Ag}
\cals@setcellprevdepth{Al}
```

The first command sets the left, top, right and bottom padding: `\cals@paddingL`, `\cals@paddingT`, `\cals@paddingR`, and `\cals@paddingB`, respectively. The value is half of the height of the argument. The second command sets the length `\cals@paddingD`, which helps to align baselines in a row. More details on the ‘Ag’ and ‘Al’ are discussed later.

```
\fontsize{20pt}{22pt}\selectfont
...
\begin{calstable}
\colwidths{{70pt}{70pt}{70pt}}
%
\fontsize{10pt}{12pt}\selectfont
\brow
  \cell{Padding} \cell{is too} \cell{big}
\erow
%
\cals@setpadding{Ag}
\cals@setcellprevdepth{Al}
\brow
  \cell{This} \cell{padding}
  \cell{is better}
\erow
%
\setlength{\cals@paddingT}{0pt}
\setlength{\cals@paddingB}{0pt}
\brow
  \cell{Zero padding} \cell{aaaaaa}
  \setlength{\cals@paddingD}{-10000pt}
  \cell{aaaaaa}
\erow
\end{calstable}
```

Padding	is too	big
This	padding	is better
Zero padding	aaaaaa	aaaaaa

In this example, we make the table font smaller than the document font. In the first row, the padding is too big. Then the padding is updated, and the second row looks better. In the third row, the top and bottom padding are set to zero. However, the second cell has additional space at top to align the baseline. To omit this, the length `\cals@paddingD` is disabled in the third cell.

## 2.6 Colors and rules

Specifying a color is straightforward: if the macro `\cals@bgcolor` is non-empty, its value is the name of the cell color.

The width of a cell border (rule) depends on the context:

- The usual borders get widths from the macros `\cals@cs@width` and `\cals@rs@width`.
- The table frame uses `\cals@framecs@width` and `\cals@framers@width`.
- The separation between the table body and its header or footer is `\cals@bodyrs@width`.

The default settings are correspondingly 0.4pt (the usual line for usual cells), 0pt (table frame is absent) and 1.2pt (header and footer are delimited by a thick line). All the borders are “phantoms” and do not affect layout.

Border types are further divided into subtypes: `cs` means “column separation” (left and right borders), and `rs` means “row separation” (top and bottom borders).

Finally, there are overrides. If any of the macros `\cals@borderL`, `\cals@borderT`, `\cals@borderR`, or `\cals@borderB` are defined, they specify the width of the left, top, right or bottom border, ignoring the cell’s context. By default, these macros are assigned `\relax` and are thus inactive.

```
\begin{calstable}
\colwidths{{60pt}{60pt}{60pt}}
\def\cals@cs@width{1pt}
\def\cals@rs@width{0pt}
\def\cals@framers@width{2pt}
\def\cals@framecs@width{1pt}
% background swap
\def\c{\ifx\cals@bgcolor\empty
  \def\cals@bgcolor{lightgray}
  \else \def\cals@bgcolor{} \fi}
%
\brow \c\cell{A}
  \c\cell{B} \c\cell{C} \erow
%
\brow \c\cell{D}
  \def\cals@borderL{3pt}
  \def\cals@borderT{4pt}
```

```

\def\cals@borderR{5pt}
\def\cals@borderB{6pt}
\c\cell{E}
\let\cals@borderL=\relax
\let\cals@borderT=\relax
\let\cals@borderR=\relax
\let\cals@borderB=\relax
\c\cell{F} \erow
%
\brow \c\cell{G}
  \c\cell{H} \c\cell{I} \erow
\end{calstable}

```

A	B	C
D	E	F
G	H	I

In this example, the macro `\c` alternately sets and disables a color of a cell.

## 2.7 Spanned cells

To define a spanning area, use the `\nullcell` command for each component cell. The argument of this command specifies the location of the cell: `l` if on the left edge, `t` on the top, `r` on the right and `b` on the bottom. To typeset the spanning area, use the command `\spancontent`, which should be given immediately after right-bottom component cell.

The following table illustrates the words, providing an example how to typeset three spanned areas of different shapes.

```

\let\nc=\nullcell
\let\sc=\spancontent

```

<code>\nc{ltr}</code>	<code>\nc{lrb}</code>	<code>\nc{tbr}</code>	<code>\nc{tbr}</code> <code>\sc{...}</code>
<code>\nc{lr}</code>	<code>\nc{lt}</code>	<code>\nc{t}</code>	<code>\nc{tr}</code>
<code>\nc{lr}</code>	<code>\nc{l}</code>	<code>\nc{}</code>	<code>\nc{r}</code>
<code>\nc{lbr}</code> <code>\sc{...}</code>	<code>\nc{lrb}</code>	<code>\nc{b}</code>	<code>\nc{br}</code> <code>\sc{...}</code>

As an example, here is a “spiral” in a  $3 \times 3$  table.

```

\begin{calstable}
\colwidths{{40pt}{40pt}{40pt}}
\def\cals@frameecs@width{0.4pt}
\def\cals@framers@width{0.4pt}
\brow
  \nullcell{ltr}
  \nullcell{lrb}
  \nullcell{tbr}\spancontent{b3, c3}
  \ht\cals@current@row=40pt
\erow
\brow
  \nullcell{lbr}\spancontent{a2, a3}

```

```

\cell{b2}
\nullcell{ltr}
\ht\cals@current@row=40pt
\erow
\brow
  \nullcell{lrb}
  \nullcell{tbr}\spancontent{a1, b1}
  \nullcell{blr}\spancontent{c1, c2}
  \ht\cals@current@row=40pt
\erow
\end{calstable}

```

a2, a3	b3, c3	
	b2	c1, c2
a1, b1		

## 2.8 User-level tricks

There are a few compatibility issues and out-of-design uses. So far, they are:

- pdfsync compatibility,
- multicols compatibility,
- inter-row page breaks.

### 2.8.1 pdfsync support

The package `pdfsync` seems obsolete, but is still in use. It registers `\every-hooks` and inserts synchronization markers. The `cals` package does not expect such interference and fails. The solution is:

- disable `pdfsync` inside a table,
- temporarily enable it inside a cell.

Sample code:

```

\makeatletter
\let\oldcalstable=\calstable
\def\calstable{\oldcalstable\pdfsyncstop}
\def\cals@AtBeginCell{\pdfsyncstart}

```

We use `\def` instead of `\let` for `\cals@AtBeginCell` because `\pdfsyncstart` and `\pdfsyncend` do not exist in the preamble; they are defined during execution of `\begin{document}`.

### 2.8.2 multicols compatibility

If a cell contains a `multicols` environment, the content is not padded. This is a side effect of the technical implementation:

- padding is implemented using `\leftskip`,
- `multicols` issues boxes in vertical mode. In this case,  $\TeX$  ignores `\leftskip`.

The solution is easy and quite unexpected: we pretend that the cell is a list item:

```
\cell{
\makeatletter
\@totalleftmargin=\cals@paddingL\relax
\begin{multicols}{2}
... now ok ...
\end{multicols}
}
```

### 2.8.3 Inter-row page breaks

All the typesetting systems I have seen break long tables between rows. But if rows are very tall, it may be useful to break even within a row, to make full use of the page height.

The `cals` package can be extended to support inter-row breaks. A proof of concept for a simplified case (no spanned cells, no vertical alignment) is published in `comp.text.tex` [7].

A complete solution is not presently available. The hardest part is to code a generic `\vsplit` — dividing a box and its internal sub-boxes into two parts, “before” and “after”. Contributions are welcome.

## 3 Technical details



For advanced customization and adding new functionality, one needs to understand the internals of the `cals` package. The rest of the article requires advanced  $\TeX$  and  $\LaTeX$  knowledge.

### 3.1 Padding and alignment of cells

In short, a cell is a vbox.

#### 3.1.1 Horizontal

Horizontal dimensions are applied indirectly when the cell content switches  $\TeX$  to the restricted horizontal mode. Before typesetting the content, the parameters are set:

- `\hsize` = width of the cell
- `\leftskip` = left padding
- `\rightskip` = right padding

If a mode switch does not occur (for example, the content is wrapped by a box), then  $\TeX$  does not use `\leftskip` and `\rightskip`, and you get no left and right padding. This is not a bug, this is a feature. If the automatic width of a box is incorrect, `cals` handles this case and forces the right value (`\wd\boxX=YY`).

Horizontal alignment is also implemented using `\leftskip` and `\rightskip`. The command `\alignC` adds `plusfill` to both skips, `\alignR` only to the right one, and `\alignL` drops the plus/minus components.

#### 3.1.2 Vertical

Before closing the box, `cals` adds:

```
\vfil \vskip\cals@paddingB
```

This code aligns the content top and sets the bottom padding.

The start of a cell is more complicated:

```
\vskip -\langle rowspan\_compensation \rangle
\vskip\cals@paddingT
\vskip-\parskip
\prevdepth=\cals@paddingD
```

In addition to the top padding (`\cals@paddingT`), there are a number of adjustments. The first is conditional and happens only if `cals` typesets a row-spanned cell. In this case, the negative skip increases the visual height of the cell, so visually the cell starts on the first row of spanning.

On switching from a vertical to a horizontal mode,  $\TeX$  adds `\parskip` glue, but we do not need this at the beginning of a cell; therefore we annihilate it. Finally, there is `\baselineskip` glue, implicitly set by `\prevdepth`. The value is calculated in such a way, that the distance between the top border and the top of the letters “Al” is exactly `\cals@paddingT`.

Cell content is packed vertically twice. First, as a normal vbox. Second, after the final height of its row is known, the cell is unboxed and put into a vbox of the target height.

### 3.2 Decorations

A table row and its decorations are separated. To illustrate the explanations, I use the second row from the following sample table (normal border is 1pt, first column right border 2pt, second column right border 3pt, right table frame 4pt, thick horizontal border 5pt, columns are 60pt, 70pt, 80pt):

a3	b3	c3
a2	b2	c2
a1	b1	c1

At some point after `\erow`, when `\cals@issue@row` is called, the following boxes and macros are set:

- `\cals@current@row`
- `\cals@current@cs`
- `\cals@current@rs@above` and `\cals@current@rs@below`
- `\cals@last@rs@below`

The row content resides in `\cals@current@row`:

```
a2          b2          c2
```

Here is an annotated dump:

```
> \box\cals@current@row=
\hbox(15.72218+0.0)x210.0 % Second row
```

```

.\vbox(15.72218+0.0)x60.0 % The cell "a2"
..\glue 4.38887           % \cals@paddingT
..\glue 0.0 plus -1.0    % -\parskip
..\glue(\parskip) 0.0 plus 1.0
..\glue(\baselineskip) 0.5
..\hbox(6.44444+0.0)x60.0, glue set 41fil
...\glue(\leftskip) 4.388 % \cals@paddingL
...\hbox(0.0+0.0)x0.0
...\OT1/cmr/m/n/10 a
...\OT1/cmr/m/n/10 2
...\penalty 10000
...\glue(\parfillskip) 0.0 plus 1.0fil
...\glue(\rightskip) 4.38 % \cals@paddingR
..\glue 0.0 plus 1.0fil % \vfil
..\glue 4.38887         % \cals@paddingB
.\vbox(15.72218+0.0)x70.0 % The cell "b2"
..  ...b2 here...
.\vbox(15.72218+0.0)x80.0 % The cell "c2"
..  ...c2 here...

```

This dump illustrates also the structure of a cell, with all the glue items, as described in the previous section.

The width of a border is defined by its location, unless the user explicitly sets the width. If the user sets different widths for a common border of adjoined cells, the greater value wins. (This is related to why `cals` does not support border colors or styles: I have no idea what to do if one cell wants, for example, green border and another red.)

Background color and vertical borders are in the box `\cals@current@cs`:



Horizontal rules are not yet typeset. Instead, they are described by the macros `\cals@current@rs@above` (row separation above the row) and the analogous `\cals@current@rs@below`, as follows:

```

> \cals@current@rs@above=macro:
->{\{60pt\}{1pt}{2pt}\relax }
   {\{70pt\}{2pt}{3pt}{5pt} }
   {\{80pt\}{3pt}{4pt}\relax }.
> \cals@current@rs@below=macro:
->{\{60pt\}{1pt}{2pt}\relax }
   {\{70pt\}{2pt}{3pt}\relax }
   {\{80pt\}{3pt}{4pt}\relax }.

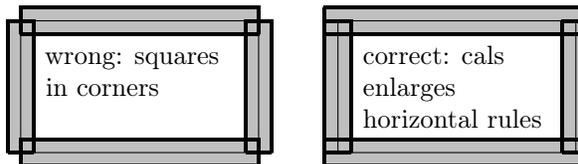
```

The macros consists of  $N$  groups of four, where  $N$  is the number of columns. Each record contains:

- the length of the rule fragment (the width of the column)
- the width of the left border
- the width of the right border
- the width of the rule or `\relax`

Unless the user manually sets the width using `\cals@borderT` or `\cals@borderB`, the last field contains `\relax`. It means “as yet unknown”, and the code will decide on the width later, when the location of the rule is clear.

The borders should be a bit longer then cell dimensions, in order to create a closed frame instead of leaving empty squares in the corners.



The procedure of typesetting a rule combines three components: 1) the default width; 2) the preceding row bottom rule description; 3) the following row top rule description. The code tries to produce as large a rule as possible. Instead of emitting a rule fragment immediately, it remembers the length and width. If the next fragment is the same width, the recorded length is extended. If not, then the pending rule is typeset, and the new fragment is remembered.

Let’s trace a simplified version of the algorithm for the separation between the first and the second rows in the sample table. The border width is 1pt, the value of `\cals@current@rs@above` is shown above, and `\cals@last@rs@below` is the same as `\cals@current@rs@below`.

- a3 border. Length 60pt, width 1pt.  
→ Remember: L=60pt, W=1pt.
- a2 border. Left 60pt, length 60pt, width 1pt.  
→ Nothing changed.
- b3 border. Length 70pt, width 1pt.  
→ Increase: L=130pt, W=1pt.
- b2 border. Left 70pt, length 70pt, width 5pt.  
→ Emit rule 130x1pt. Emit left skip 70pt.  
Remember: L=70pt, W=5pt.
- c3 border. Length 80pt, width 1pt.  
→ Emit rule 70x5pt. Remember: L=80pt, W=1pt.
- c2 border. Left 80pt, length 80pt, width 1pt.  
→ Nothing changed.
- End of the row. → Emit rule 80x1pt.

The real procedure is a bit more complicated because it takes into account the corrections for the vertical borders. Still, the real result is very similar to the simplified version:

```

\glue -0.5 % column 1, left border
\rule(0.5+0.5)x132.0 % columns 1 and 2
\glue -1.5 % column 2, right border
\glue -70.0 % back
\glue -1.0 % column 2, left border
\rule(2.5+2.5)x72.5 % column 2

```

```

\glue -1.5    % column 2, right border
\glue -1.5    % column 3, left border
\rule(0.5+0.5)x83.5    % column 3
\glue -2.0    % column 3, right border

```

The first rule is for the first and the second column, the second for a thick border in the second column, and the third rule for the third column. Note that the border in the second column consists of two overlapping rules. Such intersection is not nice, but it happens only when the width is changing, which should not happen often. In the usual case, when the width is constant, the code emits only one rule.

### 3.3 Spanning

I made several attempts before the current implementation of spanning was developed. In my opinion, the interface balances clarity for users and ease and maintenance of coding.

The command `\nullcell` performs two main tasks:

- calculate the dimensions of the spanned area, `\cals@span@width` and `\cals@span@height`,
- handle decorations.

Calculation of the width of a spanned area starts when the left-bottom `\nullcell` appears (the argument contains both `l` and `b`). The width is updated while the bottom `\nullcells` continue to appear (the argument contains `b` without `l`).

The height calculation is similar: start in the left-top (both `l` and `t`) `\nullcell`, update in a left (`l` without `t`) one. However, there is a complication in that several spanning areas can interfere. To allow several spans at once, each right (`r` without `t`) `\nullcell` adds the span height to the end of the queue `\cals@spanq@heights`, and each left (`l` without `t`) `\nullcell` takes the saved height from the beginning of the queue. With several active spans, the queue works like a cyclic buffer. The values rise from the end to the beginning, and magically the first value is always the saved height for the coming span.

The decorations of a spanned area are composed from independent parts. More precisely, the command `\spancontent` does not produce decorations at all. Instead, each `\nullcell` works like a usual `\cell` after some tuning. First, it temporarily disables all the borders. Then it looks at the location and restores the corresponding areas. For example, if the `\nullcell` is on the left edge (the argument contains `l`), the settings for the left border are restored. After the decorations are produced, all the border settings are reverted to their original values.

### 3.4 Multipage

Compared to the other parts of `cals`, the multipage functionality was very easy to code. On the other hand, it required understanding of how `TeX` works underneath, which is not my strength. Therefore, I expect bugs in multipage functionality, especially in:

- detecting if a table break is required (the macro `\cals@ifbreak`),
- executing a break within a table (the macro `\cals@issue@break` and the end of the macro `\cals@row@dispatch@nospan`)

Initially I tried to implement multipage tables in an output procedure, which, if a page break occurred within a table, added a footer before the break and a header after the break. It more or less worked, but it could not support decorations: the thickness of a row separator depends on its context. In the output procedure, it is very hard (if possible at all) to remove the old in-body separation and insert a table frame rule. My conclusion is that the main code should know if a table break is expected before typesetting a row, and create the break if required.

The following heuristic seems good: does the current row plus the footer fit into the rest of the page? If it does not, a break is required. There are also a few special cases:

- a break is forced if the user defined the macro `\cals@tbreak@tokens` (using `\tbreak`).
- no break in the header, in the footer, after the header or after the first row of a table chunk.

After a row is finished and its decorations are prepared, `cals` runs the macro `\cals@row@dispatch`. Its main parameters are:

- the bottom decoration of the previous row (given in `\cals@last@rs@below`) and its context (in `\cals@last@context`); details in the package documentation.
- the current row (`\cals@current@row`), its context (`\cals@current@context`), and decorations (`\cals@current@cs`, `\cals@current@rs@above`, `\cals@current@rs@below`).

Depending on whether there is an active rowspan, there are two different modes of work. First, when a cell spans over rows, the package must avoid a table break between. This is implemented by wrapping the row as a `vbox`. The row dispatcher emits the row not to the page, but appends it to a temporary box. After the last spanned row is collected, the collection becomes `\cals@current@row` and the collected decorations constitute `\cals@current@row@cs`. Then the dispatcher switches to the normal mode.

In the normal mode, usually it is enough to output the decorations and the row. But if a table break is required, the code saves the current row, typesets the footer, the break, the header and only then emits the saved row.

#### 4 T<sub>E</sub>X tricks and traps

The code in the `cals` package contains a few tricks, which can be reused in other tools. There were also a few unexpected problems, which I'd like to discuss here.

Maybe it is time to start collecting “T<sub>E</sub>X design patterns”, as with other programming languages [9].

##### 4.1 Actions after an implicit parameter

A straightforward definition of a command `\cell` could be:

```
\newcommand\cell[1]{%
  ...actions before...
  #1%
  ...actions after... }
```

I disliked this approach because then the macro must collect a potentially big argument, which might degrade performance. Instead, an `aftergroup`-trick is used to inject post-actions to the token stream. Pseudo-code:

```
\def\cals@cell@end{...actions after...}
\def\cell{...actions before...
  \bgroup\aftergroup\cals@cell@end
  \let\next=% eat '}' of the argument
}%{ Implicit argument follows }
```

This trick is described by Victor Eijkhout in “T<sub>E</sub>X by Topic” [3], section “12.3.4 `\aftergroup`”.

Initially, it was perhaps a premature optimization, but later useful side-effects appeared:

- The changes inside `\cell` are local due to wrapping in a group.
- Verbatim and other special content is supported inside `\cell`. This would be impossible when passing the content as a parameter.

##### 4.2 `\newcommand` for documentation, `\def` for definition

The problem with implicit arguments (as with the `\cell` command in the previous section) is that such macros confuse the readers of the code. It is very easy to overlook that a macro requires more parameters than expected.

To help the reader, `cals` defines a macro twice, first as a prototype and then as the real thing:

```
\newcommand\cell[1]{ }
\def\cell{...macro code...}
```

For me, it is an useful eye-catcher.

##### 4.3 Nested conditions

The following code does not work:

```
\let\next=\iftrue
...
\if...\next...\fi...\fi
```

The idea is to pre-calculate some condition and use it later. But instead of what the programmer wants—the first `\if` matching the outer `\fi` and `\next` (assigned to `\iftrue` here) matching the inner `\fi`—T<sub>E</sub>X matches `\if` with the inner `\fi`, and the outer `\fi` causes an error.

As a workaround, `cals` uses a variant of `\iftrue` and `\iffalse` that drops a following token:

```
\def\cals@iftrue#1{\iftrue}
\def\cals@iffalse#1{\iffalse}
```

The nested condition now looks as:

```
\let\next=\cals@iftrue
...
\if...\next\iftrue...\fi...\fi
```

The token `\iftrue` (or any other if-token) after `\next`:

- repairs the if-fi balance,
- is ignored when the code is executed.

##### 4.4 The trap of brace delimiting

When a list of macro parameters ends with “#{”, it means that the last parameter is delimited by a curly brace:

```
\def\mybox#1#\hbox#1}
\mybox to 40pt{some text}
```

expands to:

```
\hbox to 40pt{some text}
```

In this code fragment, the macro parameter #1 is ‘to 40pt’. The surprise is that we can’t make it explicit:

```
\mybox{to 40pt}{some text}
```

expands to:

```
\hbox{to 40pt}{some text}
```

The macro parameter is now empty and the box content is “to 40pt”. The text “some text” is typeset outside the box. In retrospect and in this simplified example, such expansion is obvious. In the real package, however, I fell into the trap several times.

##### 4.5 The trap of dropped curly braces

To use a macro as a list data type, it is convenient to put list items in groups. For example, a list with items “aaa”, “bbb” and “ccc” can be defined as:

```
\def\list{{aaa}{bbb}{ccc}}
```

A straightforward definition of list de-construction on the first element and the rest could be:

```
% wrong
\def\decons@helper#1#2\relax{%
  \def\first{#1}%
  \def\rest{#2}}
\def\decons#1{%
  \expandafter\decons@helper#1\relax}
```

The code works in most cases:

```
\decons\list
\show\first
\show\rest
=>
> \first=macro:
->aaa.
> \rest=macro:
->{bbb}{ccc}.
```

Unfortunately, two-element lists are de-constructed incorrectly, losing the braces:

```
\def\listII{{aaa}{bbb}}
\decons\listII
\show\first
\show\rest
=>
> \first=macro:
->aaa.
> \rest=macro:
->bbb.
```

The wrong value of `\rest` is `'bbb'`; the correct result would be `'{bbb}'`. The problem is the second parameter of the macro `\decons@helper`. If the value is `'{{bbb}{ccc}}...{xxx}'` (several items) or empty, the parameter is not changed. But if the value is `'{bbb}'` (exactly one item),  $\TeX$  interprets the curly braces as a parameter delimiter and drops them. Again, it is not a bug, but an unexpected side effect of  $\TeX$ 's rules.

#### 4.6 Unit testing

The complexity of the `cals` code is above my  $\TeX$  skills. Fortunately, automated tests helped to keep the code under control.

There are a number of tools for testing  $\TeX$  code (for example, `qstest` [5]), but I decided that it would be faster to write my own framework instead of mastering an existing one. So far, I think this was a good decision.

I wrote my tool in Python. It is straightforward. For each test:

- It takes the code fragment, assembles a complete document and compiles it.

- It extracts  $\LaTeX$  messages and the output of `\show`-commands from the log file and asserts that the result is equal to the known master.
- If the test comes with PNG images, then the tool asserts that the generated PDF, after conversion to PNG, is equal to the master images.

The tests for `cals` and the tool itself are located in the directory `test` of the `cals` package. To run the tests, execute:

```
$ export TEXINPUTS='pwd'/'../cals:
$ python support/run_tests.py
```

To run a subset of the tests, simple filtering is supported. For example, to run only the tests with the names containing “cell”, execute:

```
$ python support/run_tests.py cell
```

The testing framework is generic and can be used in other projects as well. If there is positive feedback and use by other developers, the tool will be extracted to a separate CTAN package.

◊ Oleg Parashchenko  
bitplant.de GmbH, Fabrikstr. 15  
89520 Heidenheim, Germany  
olpa (at) uucode dot com  
<http://uucode.com/>

#### References

- [1] OASIS. TM 9502:1995 – CALS table model DTD. <http://www.oasis-open.org/specs/a502.htm>, 2001.
- [2] W3C. Extensible stylesheet language (XSL), version 1.0. W3C recommendation 15 Oct 2001. <http://www.w3.org/TR/2001/REC-xsl-20011015/>, 2001.
- [3] Victor Eijkhout. *TEX by Topic: A TEXnician's Reference*. Addison-Wesley, 1992.
- [4] Simon Fear. Publication quality tables in  $\LaTeX$ . <http://mirror.ctan.org/macros/latex/contrib/booktabs/booktabs.pdf>, 2005.
- [5] David Kastrup. `qstest`, a  $\LaTeX$  package for unit tests. *TUGboat*, 29(1):193–198, 2008.
- [6] Oleg Parashchenko. CALS tables demo. <http://mirror.ctan.org/macros/latex/contrib/cals/examples/demo.pdf>, 2011.
- [7] Oleg Parashchenko. Re: Multi-page table with inter-row page breaks. `comp.text.tex`, <http://groups.google.com/group/comp.text.tex/msg/8141d45708fe85c2>, 6 Dec 2010.
- [8] UK TUG.  $\TeX$  frequently asked questions. <http://www.tex.ac.uk/faq>, 2011.
- [9] Wikipedia. Design pattern (computer science). [http://en.wikipedia.org/wiki/Design\\_pattern\\_\(computer\\_science\)](http://en.wikipedia.org/wiki/Design_pattern_(computer_science)), 2011.

## Glisterings

Peter Wilson

What e're this youth of fire weares fair,  
Rosy fingers, radiant hair,  
Glowing cheeks, and glistening wings,  
All those fair and flagrant things,  
But before all, that fiery Dart  
Had fill'd the Hand of this great Heart.

---

*The Flaming Heart*, RICHARD CRASHAW

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

Corrections, suggestions, and contributions will always be welcome.

The web, then, or the pattern; a web at once sensuous and logical, an elegant and pregnant texture: that is style, that is the foundation of the art of literature.

---

*The Art of Writing*,  
ROBERT LOUIS STEVENSON

## 1 Ornaments

One of the many free fonts available for use with L<sup>A</sup>T<sub>E</sub>X is Web-O-Mints, a Type 1 font, from the Galapagos Design Group, with L<sup>A</sup>T<sub>E</sub>X support provided by Maurizio Loreti [1]. Both of these are available from CTAN. The font consists of a set of printers' ornaments and flowers as displayed in the font table in Table 1.

Before getting in to how you might use the font, one simple application is generating a pattern like this, composed from the Web-O-Mints glyphs accessed as 'T', 'J', 'K', and 'L'.



You could use this, or something similar, to separate writings on different topics.



To make life simpler I've defined a few macros which I'll be using a lot in this column. These first two are from the samples that come with the L<sup>A</sup>T<sub>E</sub>X support for Web-O-Mints.

```
\newcommand*\wb}[2]{%
  \fontsize{#1}{#2}\usefont{U}{webo}{x1}{n}}
\newcommand*\wbc}[3]{%
  \vspace*{#1}\begin{center}
  \wb{#2}{#2}#3
  \end{center}\vspace*{#1}}
\newlength\wsp\setlength\wsp{1ex}
```

Peter Wilson

The first makes Web-O-Mints the current font with the given size and `\baselineskip`. The second sets up a `center` environment around the third argument with Web-O-Mints as the font with the size and `\baselineskip` equal. The first argument is space before and after the environment. For example, for the previous glyph I used:

```
\wbc\wsp{24pt}{4}
```

which centered the glyph accessed as '4', size 24pt, with vertical space of `\wsp` (which has been set to `1ex`) before and after.

For ease of seeing what is happening I am using a much larger glyph size than one would normally.

I will be assembling some glyphs to make more elaborate patterns. The next set of macros move or rotate their argument.

```
\newcommand*\upit}[2]{\raisebox{#1}{#2}}
\newcommand*\rotpi}[1]{\rotatebox{180}{#1}}
\newcommand*\rotrt}[1]{\rotatebox{90}{#1}}
\newcommand*\rotlft}[1]{\rotatebox{-90}{#1}}
```

You can use them on this glyph  (accessed as the character '3') like so

```
\wbc\wsp{24pt}{%
  \upit{26pt}{\rotlft{3}}% rotate left & lift
  3% normal position
  \llap{% overlap
    \upit{36pt}{\rotpi{3}}% rotate & lift
    \upit{10pt}{\rotrt{3}}% rotate right & lift
  }
}
```

to produce this rather charming device:



If you haven't come across `\llap` or its companion `\rlap`, these are T<sub>E</sub>X macros that lets its argument overlap its surroundings. More precisely, `\llap` places its argument at the left of the macro but without taking up any space; `\rlap` is similar but puts its argument at the right, again without taking up any space. Knuth's example is typesetting a  $\neq$  (neq) symbol by using either `\rlap{/}=` or `\llap{/=}` to create the neq symbol. On the other hand, the `\kern` command moves the next character rightwards (positive length) or leftwards (negative length) the given amount, but the character takes up its normal space.

I do find it difficult to tell just from looking at a single glyph what a group of them will look like. For example:



Table 1: Glyphs in the Web-O-Mints font

32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
 48	 49	 50	 51	 52	 53	 54	 55
 56	 57	58	59	60	61	62	63
64	 65	 66	 67	 68	 69	 70	 71
 72	 73	 74	 75	 76	 77	 78	 79
 80	 81	 82	 83	 84	 85	 86	 87
 88	 89	 90	 91	92	 93	94	95
96	 97	 98	 99	 100	 101	 102	 103
 104	 105	 106	 107	 108	 109	 110	 111
 112	 113	 114	 115	 116	 117	 118	 119
 120	 121	 122	123	124	125	126	127

doesn't look like much to me, but when put together with another member of the family by using `\wbc{\wsp}{24pt}{[] [] []}` the appearance is rather different.



Below is the set of four glyphs that I will be using for the first sets of patterns, specified by: `\wbc{\wsp}{24}{opqn}`



Note that these are in their natural relationship with each other. We can make simple chains like: `\wbc{\wsp}{24pt}{ppqpqp}`



Which, although attractive, isn't all that exciting. What I'm going to do is to 'attach' the single horns to the double ones and use these as the basis for a more complex pattern.

The simpler part is to move (and rotate) two single horns to join with the right hand double horn. First move the single horn to join the double with `\wbc{\wsp}{24pt}{q\kern-14pt\upit{-19pt}{n}}`



Then rotate and move a second horn, overlapping the construction we already have

```
\wbc{\wsp}{24pt}{q\kern-14pt\upit{-19pt}{n}%
\llap{\upit{35pt}{\rotpi{o}}}}
```



Doing the other double horn is slightly more complicated because of the order of the operations:

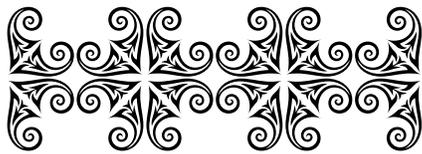
```
\wbc{\wsp}{24pt}{\upit{-19pt}{o}%
\llap{\upit{35pt}{\rotpi{n}}}\kern-14pt p}
```



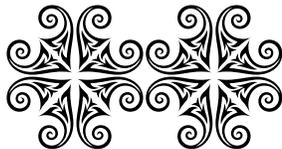
Defining a macro for each assembly will let us mix and match.

```
\newcommand*\{qno}{q\kern-14pt\upit{-19pt}{n}%
\llap{\upit{35pt}{\rotpi{o}}}}
\newcommand*\{onp}{\upit{-19pt}{o}%
\llap{\upit{35pt}{\rotpi{n}}}\kern-14pt p}
```

Now,  
`\wbc{\wsp}{24pt}{\onp\qno\onp\qno\onp\qno}`  
 produces



and  
`\wbc{\wsp}{24pt}{\qno\onp\qno\onp}`  
 displays



I had to experiment to decide on the various distances to move things to create the `\onp` and `\qno` assemblies. These distances would have to be changed if something other than 24pt was used as the font size. However, it is always possible to use `\scalebox` from the `graphicx` package to appropriately size a pattern. This gives a half-size result compared with the previous ones.

`\wbc{\wsp}{24}{\scalebox{0.5}{\qno\onp}}`



Except for the simplest scheme of just putting the glyphs in a row, experimentation will nearly always be required to obtain sympathetic relationships among the elements of the pattern. They don't have to be mathematically exact but must look good to the eye.

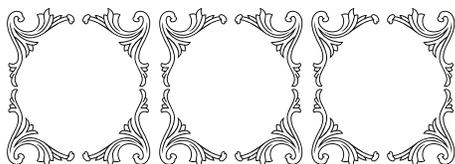
Moving on, here is another set of four glyphs, which would normally be used at the corners of a page, that can be combined in interesting ways.

`\wbc{\wsp}{24pt}{E F G H}`



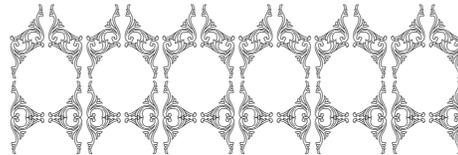
One simple way is just using two lines, which reminds me of a row of gilt mirrors.

`\wbc{\wsp}{24pt}{EFEFEF\HGHHGH}`



We can add some further decorative elements, reducing the size at the same time:

`\wbc{\wsp}{12pt}{HGHHGHGHGHGH\[-3pt]`  
`EFEFEFEFEFEF\HGHHGHGHGHGH\[-3pt]`  
`EFEFEFEFEFEF}`



and add more and more if desired. On the other hand, joining the four elements in a different manner can lead to something even fancier.

In the days of lead type, a sort (a single character) was a rectangular bar of lead with the glyph in relief on the end that was to be inked and printed. There was no way of stretching or shrinking a piece of type, and neither was there any way of getting one piece of type to overlap another, except by printing twice, once with one sort and then with the second sort. With digital fonts these restrictions no longer apply. In the next example, the bounding boxes of the glyphs overlap, even if the glyphs themselves do not.

`\newcommand*{\k1}{\kern-24pt}`  
`\newcommand*{\ks}{\kern-4pt}`  
`\newcommand*{\mir}{E\k1 H\ks G\k1 F}`  
`\wbc{\wsp}{24pt}{\scalebox{0.75}{%`  
`{\mir\ks\mir\ks\mir\ks\mir}}}`



Again, you can add to this basic scheme, embellishing it as much as you think bearable. For instance, by inserting another set of glyphs inside a mirror, although this may get complicated. The experimentally determined code below appears to produce a reasonable result.

First, here is a macro for producing a more vertically oriented version of the four leaves that I showed earlier.

`\newcommand*{\leaves}{\upit{32pt}{\rotlft{3}}%`  
`\kern-6pt3%`  
`\kern-9pt\upit{17pt}{\rotrt{3}}%`  
`\kern-30pt\upit{49pt}{\rotpi{3}}}`  
`\wbc{\wsp}{24pt}{\leaves}`



And secondly, this is the code for combining the mirror and leaves so that they will be centered:

```
\wbc{\wsp}{24pt}{\mbox{}\kern-24ptEF%
\kern-43pt{\upit{-22pt}%
{\scalebox{0.6}{\leaves}}}%
\[-18pt]\mbox{}\kern0ptGH}
```



Moving on to a more organic feel, represented above by `\wbc{\wsp}{16pt}{m/m/m/m/m/m}`, many of the traditional ornaments were based on vines, although whether this was due to a proclivity towards wine by the typesetters I couldn't say.

This is a typical scheme, mixing grapes and leaves.

```
\wbc{\wsp}{15pt}{cedafbcedced}
```



And here is another with major emphasis on the grapes:

```
\wbc{\wsp}{15pt}{ghghghghghgh}
```



And a third, back to the grapes plus leaves:

```
\wbc{\wsp}{15pt}{fgfgfgfgfgfg}
```



This one, you may have noticed, forms the top and bottom of the frame around page 204. The sides are also formed from similar grapes and leaves.

The method I used for framing was to create the frame as a zero-sized picture and add that to the page footer; I could just as well have added it to the header but the footer was easier for me in this case, because I wanted to keep the *TUGboat* header. You would normally use the facilities provided by the `fancyhdr` package [2] or the `memoir` class [3] for this, both of which should be available on your system.

Here's the code for the picture of the frame.

```
%% draws a (page) frame
\newcommand*{\goddfoot}{\begin{picture}(0,0)
```

```
\wb{10pt}{10pt}
\put(-29,-20){% change these to move the frame
\begin{picture}(0,0)
\multiput(0,0)(25,0){21}{fg} % bottom

\multiput(-5,5)(0,24){31}{i} % left
\multiput(-5,23)(0,24){31}{\rotpi{k}}

\multiput(520,5)(0,24){31}{j} % right
\multiput(520,23)(0,24){31}{\rotpi{1}}

\multiput(0,742)(25,0){21}{fg} % top
\end{picture}}
\end{picture}}
\let\gevenfoot\goddfoot
```

For the example on the previous page, I created a new pagestyle using the *TUGboat* headers and adding my feet (this is code that is normally hidden in a class or package).

```
\makeatletter
%% new 'glistner' pagestyle
\newcommand*{\ps@glistner}{%
\def\@evenfoot{\gevenfoot}
\def\@oddfoot{\goddfoot}}
\makeatother
```

Zero-sized pictures have other uses as well. I put one containing a gray checkerboard pattern, composed from the Web-O-Mints 'z' and '7' glyphs, at the start of the previous paragraph to form a textured background. You may love or hate this particular result but the technique can be useful.

May you have many happy hours designing your own ornaments and embellishments, but consider that maybe simple and few are better than elaborate and many; it all depends on the feel you are trying to convey.

## References

- [1] Maurizio Loreti. `webomints`, 2002. [mirror.ctan.org/latex/macros/contrib/webomints](http://mirror.ctan.org/latex/macros/contrib/webomints).
- [2] Piet van Oostrum. Page layout in L<sup>A</sup>T<sub>E</sub>X, 2005. [mirror.ctan.org/latex/macros/contrib/fancyhdr](http://mirror.ctan.org/latex/macros/contrib/fancyhdr).
- [3] Peter Wilson. The memoir class for configurable typesetting, 2011. [mirror.ctan.org/latex/macros/contrib/memoir](http://mirror.ctan.org/latex/macros/contrib/memoir).

◇ Peter Wilson  
20 Newfield Ave.  
Kenilworth CV8 2AU, UK  
herries dot press (at)  
earthlink dot net

---

## Merciadri packages: An overview

Luca Merciadri

### 1 Introduction

I have written four L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> packages: `bigints`, `dashundergaps`, `plantslabels`, and `turnthepage`. Because

- ◇ there are so many submitted packages, and
- ◇ T<sub>E</sub>X information sources are so heavily used that articles rapidly go to the archives,

their respective releases could have gone unnoticed by the majority of the community. I here propose a brief summary about what they do, and how they could be useful for your work. This idea came to my mind when reading Nicola Talbot's article [7]. I don't have the delusion that they will be useful to everybody all the time, but, except `plantslabels`, it's reasonable to think they might come in handy at least, *i.e.* for a T<sub>E</sub>X author to want to write a bigger integral than the one which is proposed, to dot or dash some text, or to indicate that the page needs to be turned.

I shall now describe them in alphabetical order, though I wrote fewer packages than Nicola!

The reader might notice that the text here is essentially extracted from their respective manuals: [2, 3, 4], where more detailed information and a bibliography of each is available. I am open to any suggestions, or remarks, concerning my packages.

After their respective descriptions, we will take a look at another tip: how to use a brace so that some elements of a given matrix are selected. We will here present the case where the brace is above the matrix.

### 2 The `bigints` package

The `bigints` package (v1.1) helps you to write *big integrals* when needed. When making a report for a course, during the 2009–2010 academic year, I realized that there was no satisfactory implementation of 'big integrals' in L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>. (If there are, please let me know.) My idea, during the report, was to write a big integral in front of an  $n \times 1$  ( $n$  rows, 1 column) matrix, to signify the integration of every element on the  $n$  rows. (For the doubtful, I asked mathematicians, and, yes, this is sometimes written like this, though it is not very standard.) I also thought about writing the integral in front of a big expression, which might be different from a matrix.

#### 2.1 Example

Consider, for example, a rocket which is propelled in space thanks to motors, giving a thrust by mass

unity ( $\alpha > 0$ ) which is supposed constant. Using Mechanics' laws, the time which is necessary to go from Earth's surface ( $r = R$ ) to an orbit of height  $2R$  ( $r = 3R$ ) is given by

$$T = \int_R^{3R} \frac{dr}{\sqrt{2\alpha(r-R) + 2\mu\left(\frac{1}{r} - \frac{1}{R}\right)}}, \quad (1)$$

where  $\mu$  is a positive constant, associated to the gravitational force.

In Equation (1), the integral sign is too small. Consider now

$$T = \int_R^{3R} \frac{dr}{\sqrt{2\alpha(r-R) + 2\mu\left(\frac{1}{r} - \frac{1}{R}\right)}}, \quad \text{or}$$

$$T = \int_R^{3R} \frac{dr}{\sqrt{2\alpha(r-R) + 2\mu\left(\frac{1}{r} - \frac{1}{R}\right)}}.$$

They look reasonably better, simply because the integral sign's height is related to the integrand's height. (For skeptics: even without `displaystyle` in front of the opening parenthesis, the integral sign of the original expression is still too small.)

Creating integral signs which are adapted to their argument (the integrand) was the idea for the package, and this is what gave rise to this package.

### 2.2 Available commands

The available commands and their output are shown in Table 1.

Put briefly, I defined `\bigint`, `\bigints`, and so on, with their respective o-counterparts (for line integrals along a closed curve, for example). The only rule to keep in mind is that the more you add 's' to the integral command, the smaller the integral sign is. To use these functions, you simply need to load the `bigints` package.

### 3 The `dashundergaps` package

The `dashundergaps` package (v1.2) helps you to use a pattern or patterns from this list:

- `\dashing`,
- `\dotting`,
- `\underlining`

for a word which can be either hidden, or not.

This can be useful in these situations:

1. You are writing a document for which you need to dash or (and) to dot text,

Command	Std. output	Command output
<code>\bigint,</code> <code>\bigoint</code>	$\int, \oint$	$\int, \oint$
<code>\bigints,</code> <code>\bigoints</code>	$\int, \oint$	$\int, \oint$
<code>\bigintss,</code> <code>\bigointss</code>	$\int, \oint$	$\int, \oint$
<code>\bigintsss,</code> <code>\bigointsss</code>	$\int, \oint$	$\int, \oint$
<code>\bigintssss,</code> <code>\bigointssss</code>	$\int, \oint$	$\int, \oint$

**Table 1:** Commands in the `bigints` package.

- You want to write a test for which students have to “fill in the gaps”, and you want to choose when to print the answers.

### 3.1 Examples

Here is an example of sentence dashing.

```
\documentclass[10pt]{article}
\usepackage[dash]{dashundergaps}
\begin{document}
\dashuline{This is a dashed sentence}
\end{document}
```

gives

This is a dashed sentence
---------------------------

Dotting is done in the same way.

And an example of dotted gaps for a student version (notice that gaps are always numbered):

```
\documentclass[10pt]{article}
\usepackage[dot, phantomtext]{dashundergaps}
\begin{document}
In Artificial Intelligence, ‘RL’ means
‘Reinforcement \gap{Learning}.’
\end{document}
```

results in

In Artificial Intelligence, “RL” means “Reinforcement ..... (1).”
---

Despite its rather ugly appearance, several people asked me how to achieve something like

Head A	Head B	Head C
.....		
Col 1	Col 2	Col 3
Col 1	Col 2	Col 3

For this, use:

```
\begin{tabular}{lll}
\hline
Head A & Head B & Head C\\
\multicolumn{2}{l}{\dotuline{\hfill}}\\
Col 1 & Col 2 & Col 3\\
Col 1 & Col 2 & Col 3\\
\hline
\end{tabular}
```

where you want it to appear.

### 3.2 Available commands

Option(s)	Consequence
<code>\gap{text}</code>	<code>\dashuline{text}</code>
<code>\dash (only)</code>	<code>\dotuline{text}</code>
<code>\dot (only)</code>	<code>\text</code>
<code>dash, dot</code>	<code>\text</code>
<code>phantomtext (only)</code>	<code>\text</code>
<code>phantomtext, dash</code>	<code>\text</code>
<code>phantomtext, dot</code>	<code>\text</code>
<code>phantomtext, dash, dot</code>	<code>\text</code>
<code>phantomtext, teachernotes</code>	<code>\text</code>
<code>phantomtext, dash, teachernotes</code>	<code>\text</code>
<code>phantomtext, dot, teachernotes</code>	<code>\text</code>
<code>phantomtext, dash, dot, teachernotes</code>	<code>\text</code>

**Table 2:** Possible calls of the `dashundergaps` package.

Without any option, the package will not do anything useful. Consequently, one of the following options should be specified:

- **dash**: will dash `text` if used with the command `\dashuline{text}` where you want “text” to be dashed (*i.e.* somewhere in the `document` environment).
- **dot**: will dot `text` if used with the command `\dotuline{text}` where you want “text” to be dotted.
- **phantomtext**: helps in writing a pattern at the place of the text. This pattern can be
  - dashing, if used with **dash** option;
  - dotting, if used with **dot** option;
  - underlining, if used with (**dash and dot**) options *or* with neither **dash** nor **dot**;
  - the text itself, if used with the **teachernotes** option.
- **teachernotes**: see the last above.
- **displaynbgaps**: produces, at the end of your document (in the center of the page), a summary of the number of gaps, like this:

**GAPS:** *x*.

All the commands (their order is immaterial) of `dashundergaps.sty` are given in Table 2 *except* the use of `displaynbgaps`, which can trivially be used iff `phantomtext` is used. Here, “×” means “not applicable”.

### 3.3 Sectioning and dashundergaps

#### 3.3.1 Numbering

Some users would like to have dashed or dotted section numbers. This can be done with, for example:

```
\usepackage[dash,dot]{dashundergaps}
\usepackage[calwidth,pagestyles,...]{titlesec}
...
\titleformat{\section}
  {\normalfont\Huge\bfseries}
  {\dashuline{\thesection}}{1em}{}
\titleformat{\subsection}
  {\normalfont\LARGE\bfseries}
  {\dotuline{\thesubsection}}{1em}{}

```

It is possible, and will work. For example, here, sections and subsections will have their numbering respectively dashed and dotted.

#### 3.3.2 Titles

However, this approach using `titlesec` does not work for the section titles. To do this, the present solution is to use, in each section command, code like this:

```
\section{\protect\dashuline{My section}}
```

At present, the `\protect` is required.

## 4 The plantslabels package

I have a somewhat wide-ranging collection of carnivorous plants. Once objects are categorized (be they plants, or anything else), it is useful to distinguish them easily. This is easy between plants whose characteristics are radically different, as is normally the case with distant species (as defined in biology). But once one has many objects sharing the same characteristics (which, here, occurs more frequently for plants belonging to the same species), it becomes more difficult not to mix up two plants. Or some plants belonging to the same species could have different reactions towards natural elements such as cold, etc., and it is thus interesting to distinguish them, which leads to the idea of labelling. This is what motivated me to write this package, which aims at making labels for plants, as its name suggests.

### 4.1 Example

Let’s say that you have two kinds of plants that you want to label: “Myplant1” and “Myplant2.” One habitually lives in the desert, and the other lives in tropical regions. You have, say, 2 specimens of the first, and 4 of the second. You can invoke, assuming `cactus.eps` is your image for the first one, that you have no image for the second one, and that they respect the conditions mentioned below:

```
\plant{1}{1}{2}{Myplant1}{5}{EUR}
  {$-10\to +50$}{Peat moss, sand, %
perlite}{cactus.eps}
\plant{2}{2}{4}{Myplant2}{10}{EUR}
  {$20\to +40$}{Peat moss, fertilizer}{}

```

This will create  $2+4 = 6$  labels (2 of the first, 4 of the second). The two labels are represented (without the captions, images and a slightly smaller length for the label) in Figure 1.

<b>Name</b>	<i>Myplant1</i>
<b>Price</b>	5 EUR
<b>Temperature</b>	−10 → +50
<b>Substratum</b>	Peat moss, sand, perlite

(a) Label of *Myplant1*.

<b>Name</b>	<i>Myplant2</i>
<b>Price</b>	10 EUR
<b>Temperature</b>	20 → +40
<b>Substratum</b>	Peat moss, fertilizer

(b) Label of *Myplant2*.

**Figure 1:** The two kinds of labels produced.



## 6 Matrices with borders

The code in this section comes from [1]; I thought it was valuable enough to describe. For pedagogical reasons, one may want matrices with borders, like this:

$$\left( \begin{array}{r|ccc} a & \overbrace{b \dots c}^{n \text{ times}} \\ \hline d & & & \\ \vdots & & & A \\ e & & & \end{array} \right)$$

This can be achieved with a variety of approaches. First, here is what produced the above example:

```
\usepackage{multirow}
\makeatletter
\def\Biggg#1{{\hbox{\$left#1\ vbox to32\p@{ }
\right.\n@space$}}}
\newdimen\bracketwidth
\settoheight{\bracketwidth}{\Biggg{ }
\makeatother
\left[ \begin{array}{r@{}r@{\hspace{\arraycolsep}}rcc%
c@{\hspace{\arraycolsep}}c@{ }1}
& & & \multicolumn{3}{c}{\vspace{-.5em}
\overbrace{\hphantom{b \hspace{2\arraycolsep}
\cdots
\hspace{2\arraycolsep} c}}
~{n \mbox{\scriptsize\ times}}}}\
\multirow{5}{\bracketwidth}[3pt]{\Biggg{ }
& & a & \multirow{5}{1pt}[3pt]{\vrule height52pt}
& b & \cdots & c &
\multirow{5}{\bracketwidth}[3pt]{\Biggg}}\
\cline{2-7}
& & d\
& & \vdots & & A\
& & e
\end{array} \right]
```

This is close to unreadable for me. A more readable solution:

```
\left[ \vbox{%
\hskip2.8em\overbrace{\hphantom{b %
\hspace{2\arraycolsep} \cdots
\hspace{2\arraycolsep} c}}~{n%
\mbox{\scriptsize\ times}}$
\vskip-.25em
\left(
\begin{array}{r|ccc}
a & b & \cdots & c \\
\hline
d \\
\vdots & & & A \\
e
\end{array} \right)
\end{array} \right]
```

In another approach, one could define:

```
\def\moverbrace#1#2{%
\newdimen\moverbracewd
```

```
\settowidth\moverbracewd{#1}%
\addtolength\moverbracewd{-2\arraycolsep}
\ vbox to 1.6ex{\hsize=\moverbracewd
\centering\vss
\overbrace{#1}^{#2}}}
```

and then use it, for example like this:

```
\left[ \left(
\begin{array}{r|c}
a & \moverbrace{b \hspace{2\arraycolsep}
\cdots \hspace{2\arraycolsep} c}
{n \mbox{\scriptsize\ times}} \\
\hline
d \\
\vdots & A \\
e
\end{array} \right) \right]
```

In all the examples, `\mbox` could evidently be replaced by a `\text` equivalent. Please tell me if you know a simpler way to achieve this.

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

## References

- [1] Glad Deschrijver. *T<sub>E</sub>X Tricks*, 2011. <http://users.ugent.be/~gdschrij/LaTeX/textricks.html>.
- [2] Luca Merciadri. The `bigints` package, 2010. <http://mirror.ctan.org/macros/latex/contrib/bigints>.
- [3] Luca Merciadri. The `dashundergaps` package, 2010. <http://mirror.ctan.org/macros/latex/contrib/dashundergaps>.
- [4] Luca Merciadri. The `plantslabels` package, 2010. <http://mirror.ctan.org/macros/latex/contrib/plantslabels>.
- [5] Luca Merciadri, Marc Van Dongen, and Martin Münch. The `turnthepage` package, 2011. <http://mirror.ctan.org/macros/latex/contrib/turnthepage>.
- [6] Philipp Stephani and Luca Merciadri. New on CTAN: `turnthepage`, `comp.text.tex`, 2010.
- [7] Nicola Talbot. Talbot packages: An overview. *TUGboat*, 31:65–67, 2010.
- [8] Marc Van Dongen and Luca Merciadri. Turn the Page, `comp.text.tex`, 2011.

---

## ConTeXt basics for users: Paper setup

Aditya Mahajan

### 1 Basic setup

#### 1.1 Setting paper size

Plain TeX and L<sup>A</sup>T<sub>E</sub>X were primarily developed in the US. So, they default to letter paper, which is the standard paper size in the US. ConTeXt was developed in the Netherlands. So, it defaults to A4 paper, which is the standard paper size in Europe (and almost everywhere else in the world).

Changing the paper size is easy; for letter size:<sup>1</sup>

```
\setuppapersize[letter]
```

Similarly, to get A4 paper, use:

```
\setuppapersize[A4]
```

#### 1.2 Predefined paper sizes

Both `A4` and `letter` are predefined paper sizes. ConTeXt predefines many other commonly used paper sizes. These include:

- `letter`, `ledger`, `tabloid`, `legal`, `folio`, and `executive` sizes from the North American paper standard;
- sizes `A0–A10`, `B0–B10`, and `C0–C10` from the A, B, and C series of the ISO-216 standard;
- sizes `RA0–RA4` and `SRA0–SRA4` from the RA and SRA series of the ISO-217 paper standard;
- sizes `C6/C5`, `DL`, and `E4` from the ISO-269 standard envelope sizes;
- `envelope 9–envelope 14` sizes from the American postal standard;
- sizes `G5` and `E5` from the Swedish SIS-014711 standard. These are used for Swedish theses;
- size `CD` for CD covers;
- size `S3–S6`, `S8`, `SM`, and `SW` for screen sizes. These sizes are useful for presentations. `S3–S6` and `S8` have an aspect ratio of 4 : 3. `S3` is 300pt wide, `S4` is 400pt wide, and so on. `S6` is almost as wide as a `A4` paper. `SM` and `SW` are for medium and wide screens; they have the same height as `S6`;
- a few more paper sizes, which I will not mention here. See `page-lay.mki(i|v)` for details.

#### 1.3 Defining new paper sizes

The predefined paper sizes in ConTeXt cannot fit all needs. To define a new paper size, use

```
\definepapersize[exotic]
    [width=50mm, height=100mm]
```

which defines a paper that is 50mm wide and 100mm high; the name of this paper is *exotic* (we could have used any other word). All predefined paper sizes are defined using `\definepapersize`. For example, `A4` paper is defined as:

```
\definepapersize [A4] [width=210mm,height=297mm]
```

Use this new paper size like any of the predefined paper sizes. For example, to set the paper size to 50mm x 100mm paper, use

```
\setuppapersize[exotic]
```

#### 1.4 Orientation

Most of the popular paper sizes default to a portrait orientation. To get landscape orientation, use

```
\setuppapersize[letter,landscape]
```

### 2 Changing paper setup mid-document

Normally, the paper size is set up once—in the environment file—and doesn't need to be changed later. But, occasionally, changing paper size mid-document is needed; for example, to insert a table or a figure in landscape mode. There are two ways to change the paper size mid-document. To illustrate those, let us first define two paper sizes for convenience:

```
\setuppapersize[main] [A4]
```

```
\setuppapersize[extra] [A4,landscape]
```

One way to change the document size mid-document is simply to call `\setuppapersize`.

```
\starttext
% ...
% text with main paper size
% ...
\page \setuppapersize[extra]
% ...
% pages in landscape mode
% ...
\page \setuppapersize[main]
% ...
% back to main paper size
% ...
\stoptext
```

The `\page` before `\setuppapersize` is necessary as `\setuppapersize` changes the size of the current page.

---

<sup>1</sup> The syntax used here only works with ConTeXt versions newer than February 2011. Before that, you had to use

```
\setuppapersize[letter][letter]
```

to get letter sized paper. You may wonder why we need to repeat the paper size twice. In most cases, these are the same. You only need to use different arguments if you want to print on a bigger paper and trim it later (see the section on print size for details).

Often times, a different paper size is needed only for one page. Rather than manually switching the paper size back and forth using `\setuppapersize`, a convenient alternative is to use `\adaptpapersize`, which automatically reverts back to the existing paper size after *one* page. For example:

```
\setuppapersize[main]
\starttext
Page 1. Portrait \page
Page 2. Portrait \page
\adaptpapersize[extra]
Page 3. Landscape \page
Page 4. Portrait
\stoptext
```

As with `\setuppapersize`, always use an explicit `\page` before `\adaptpapersize`.

Successfully printing a document with such mixed paper sizes requires setting appropriate options in the document viewer and/or printer controller.

### 3 Setting print size

Occasionally you may want to print on a larger paper than the actual page size. This could be because you want to print to the edge of the page — so you print on large paper and crop later — or because the page size that you are using is not standard.

For example, suppose you want to print an A5 page on a A4 paper (and crop later). For that, you need to specify that the paper size is A5 but the *print paper* size is A4. This information is specified with the two-argument version of `\setuppapersize`:

```
\setuppapersize[A5][A4]
```

#### 3.1 Changing page location

By default, this places the A5 page on the top left corner of the A4 paper. To place the A5 page in the middle of the A4 paper use:

```
\setuppapersize[A5][A4]
\setuplayout[location={middle,middle}]
```

Other possible values for `location` are:

- `{top,left}`, `{top,middle}`, `{top,right}`,
- `{middle,right}`, `{middle,left}`,
- `{bottom,left}`, `{bottom,middle}`, and `{bottom,right}`.

Since `{middle, middle}` is the most commonly used value, it has a shortcut: `location=middle`.

If you use `{*,left}` or `{*,right}` and print double-sided, then also add `duplex` as an option; for example `location={duplex,top,left}`. This en-

sures that the page paper is moved appropriately on even pages.

### 3.2 Crop marks

To get crop marks (also called cut marks):

```
\setuplayout[marking=on]
```

By default, the page numbers are also included with the crop marks. To get additional information like job name, current date and time along with the crop marks, use

```
\setuplayout[marking=text]
```

If you want just the crop marks, and no other text, use

```
\setuplayout[marking=empty]
```

### 3.3 Defining page and print size combinations

It is convenient to define paper-size/print-paper-size combination for later reuse. These are also defined using `\definepapersize`. For example, suppose you want to define two paper-size/print-paper-size combinations: A4 paper on A4 print paper for normal work flow, and A4 paper on A3 print paper for the final proofs. For that, use the following:

```
\definepapersize[regular][A4][A4]
\definepapersize[proof][A4][A3]
```

You can then combine these paper sizes with modes:<sup>2</sup>

```
\setuppapersize[regular]
\doifmode{proof}{\setuppapersize[proof]}
```

Then, when you compile the document in the normal manner, you will get A4 paper on A4 print paper; if you compile the document with `-mode=proof`, then you will get a A4 paper on A3 print paper.

## 4 Conclusion

Paper setup is one of the most basic requirements for creating your own style. In this article, I explained the basics of paper setup, deliberately leaving out more advanced setups that are described in the *Page Design* chapter<sup>3</sup> of the new ConTeXt manual.

◇ Aditya Mahajan  
adityam (at) ieee (dot) org

<sup>2</sup> See the previous article on *Conditional Processing* in this series, a slightly modified version of which is available on the ConTeXt wiki: <http://contextgarden.net/Modes>

<sup>3</sup> <http://context.aanhet.net/svn/contextman/context-reference/en/co-pagedesign.pdf>

## Experiences with notes, references, and bibliographies

David Walden

Notes, references, and bibliographies, and the possible interactions among them, collectively are a complicated topic (at least in American English writing), and one that I struggle with on all but the shortest and simplest writing projects. In this paper I illustrate and discuss some of my approaches and struggles. I don't claim any particular expertise—just lots of experiences.

In this paper, I will use the word *notes* to mean strictly discursive material (that is, auxiliary discussion of the topic that would confuse the thread of the discussion if included in the main text), and I will use the word *references* to mean strictly references to (or acknowledgments of) sources and additional documentation on the topic.

Referenced items can appear in footnotes, endnotes, or bibliographies, typically using reference markers in the main text that are either some form of author-year notation or are a unique (to the book or chapter) numeric or alpha-numeric sequence. Notes can appear in footnotes, endnotes, or in reference lists, typically cited from the main text with a sequence number.

In the various style manuals I have consulted, references listed in notes have been allowed a more relaxed format (for example, as in footnote 3) than references listed in bibliographies; for example:

Ackoff, Russell. 1981. *Creating the Corporate Future*. New York: John Wiley & Sons.

### 1 Most simply

Most simply, one's footnotes can be inserted with `\footnote{...footnote text...}`, using L<sup>A</sup>T<sub>E</sub>X's default of the footnote getting automatically numbered and being placed on the bottom of the page where the `\footnote` command appeared.<sup>1</sup>

The most basic approach to literature citations, then,<sup>2</sup> is probably to put them in regular footnotes with manual formatting of the book or article information, as below.<sup>3</sup> (By the way, the style manual cited in that footnote has 111 pages about how to deal with references, notes, and bibliographies.) However, one doesn't want to repeat the full reference if one needs to cite it again somewhere in the same document. This can be handled by putting a label in the first footnote comment, e.g.,

<sup>1</sup> This is an example of a footnote.

<sup>2</sup> Avoiding learning anything new about L<sup>A</sup>T<sub>E</sub>X.

<sup>3</sup> *The Chicago Manual of Style*, thirteenth edition, The University of Chicago Press, Chicago, IL, 1982.

```
\footnote{\label{foot:CMS}\textit{The
Chicago Manual of Style}...}
```

and then using the code

```
\textsuperscript{\ref{foot:CMS}}
```

as the footnote marker, as I have done here.<sup>3</sup>

## 2 My first book in L<sup>A</sup>T<sub>E</sub>X

My first significant project with L<sup>A</sup>T<sub>E</sub>X was in writing and composing a 760-page book.

### 2.1 Using BIB<sub>T</sub>E<sub>X</sub>

This book has 315 items in the references section at the end of the book, and it had a notes section at the end of the Preface, the end of each of its 29 chapters, and at the end of the afterword.

With so many references, it was time to learn about BIB<sub>T</sub>E<sub>X</sub> (<http://www.ctan.org/pkg/bibtex>) which I mostly did using the then current edition of Kopka and Daly's *Guide to L<sup>A</sup>T<sub>E</sub>X*.

I don't remember exactly how I did this, as after I provided a typeset copy of the manuscript to the publisher, the publisher retypeset it using QuarkExpress, as explained in my TUGboat paper about this experience (<http://tug.org/TUGboat/tb24-2/tb77walden.pdf>). However, it was something like the following.

I created a `biblio.bib` file as follows (except with 315 items in the file instead of 3):

```
\begin{thebibliography}{ABCDEFGF}
@article{Abell93,
author="Thomas E. Abell and Dawn Dougherty Fitzgerald",
title="{HP}'s Quality Management System:
{CEO} Roundtable Report",
journal="The Center for Quality of Management Journal",
volume={2}, number={3}, pages={3--4},
month={Summer}, year={1993} }
```

```
@book{Ackoff81,
author="Russell Ackoff",
title="Creating the Corporate Future",
publisher="John Wiley & Sons",
address="New York", year=1981 }
```

```
@book{Akao90,
author="Yoji Akao",
title="Quality Function Deployment",
address="Cambridge, MA",
publisher="Productivity Press", year=1990 }
\end{thebibliography}
```

I found it convenient to put the items in the `biblio.bib` file in alphabetic order basically by first author's last name and year of publication (abbreviated in the BIB<sub>T</sub>E<sub>X</sub> key field with a two digit year—I wasn't referencing any items where the early 1900s could be mixed up with the early 2000s).

I cited the references as in this example file:

```

\documentclass{book}
\begin{document}
TEST
\cite{Ackoff81} \cite{Abell93} \cite{Akao90}
\renewcommand*{\bibname}{References}
\bibliographystyle{plain}
\bibliography{biblio}
\end{document}

```

One uses  $\text{\LaTeX}$  with a  $\text{\LaTeX}$  document named  $X$  by compiling  $X.tex$ , then giving the command  $\text{\texttt{bibtex X}}$ , and then compiling  $X.tex$  again. This resulted in the following test output:

TEST [2] [1] [3]

and a references section at the end of the book such as the following:

### References

- [1] Thomas E. Abell and Dawn Dougherty Fitzgerald. HP's quality management system: CEO roundtable report. *The Center for Quality of Management Journal*, 2(3):34, Summer 1993.
- [2] Russell Ackoff. *Creating the Corporate Future*. John Wiley & Sons, New York, 1981.
- [3] Yoji Akao. *Quality Function Deployment*. Productivity Press, Cambridge, MA, 1990.

Because the plain  $\text{\LaTeX}$  style orders bibliography entries alphabetically by the first author's last name, the 315-item bibliography became a useful resource in its own right.

See also Appendix A.

## 2.2 End-of-chapter notes

The publishing world for management books doesn't like footnotes — they look too scholarly to be popular. In fact, the non-fiction world in the U.S. increasingly shies away from using note markers in the main text. In many books there is a set of notes at the end of the book with a section for each chapter and the individual notes labeled with a page number, a brief quote of text from that page, and then notes about the quoted words. An example follows:<sup>4,5</sup>

<sup>4</sup> Taken from Ian Ayres, *Super Crunchers*, Bantam Books, New York, 2007, p. 224.

<sup>5</sup> To try to accomplish this endnote style, I tweaked a few lines of `endnotes.sty` to make the `endnote` command have two arguments, where the first argument is text that goes inline in the main text and is also shown in the endnotes, the second argument is the endnote itself, and the correct page number gets passed along to the endnote. But mine is not a robust solution: it was done by trial and error without so much understanding of `endnotes.sty`, and without any generality or cleanup of parts of `endnotes.sty` that were no longer needed. It would be nice if someone would implement a *real* package to do this.

## CHAPTER 2

*Page 46: Fisher proposes randomization:* Ronald Fisher, *Statistical methods for Research Workers* (1925); Ronald Fisher, *The Design of Experiments* (1935).

In any case, the editors of my book would not stand for having notes on pages of the main text. We compromised on end-of-chapter notes with numeric note markers in the main text of the chapters, although I had been drafting the book using footnotes.

I handled this by using the `endnotes` package (`endnotes.sty`) to which I made a few tiny changes in formatting (renaming the resulting changed file `r-endnotes.sty` (for revised endnotes). I initiated use of this capability in the usual way, that is,

```
\usepackage{r-endnotes}
```

(I had not yet learned about `\RequirePackage`.)

I next added the following code to my style file for this book that was processed as part of the  $\text{\LaTeX}$  preamble, and I put a `\dumpendnotes` command at the end of the file for each chapter. The `\theendnotes` command in the following definition is the piece of `endnotes.sty` that actually includes the saved-up endnotes at that point in the  $\text{\LaTeX}$  output.

```

\renewcommand{\footnote}{\endnote}
\newcommand{\dumpendnotes}{%
  \medskip
  \begingroup
    \setlength{\parindent}{0pt}
    \setlength{\parskip}{1ex}
    \renewcommand{\notesize}{\normalsize}
    \theendnotes
  \endgroup
  \setcounter{endnote}{0}
}

```

With endnote sections for each chapter and a separate bibliography, the notes often also cited items in the bibliography, e.g., note 2 (for chapter 22) on page 465 of that book says:

2. The idea of push-and-pull, introduced in the figure, are derived from the ideas of [189].

where 189 is the number of a bibliography item.

## 3 Incremental development

Even when I know I am going to be using notes at the ends of chapters or at the end of the book, I usually do my original drafting with the notes at the bottom of pages so that I can see the text referencing the note and the note at the same time. Thus, in recent years I often do not use the actual `\footnote` and `\endnote` commands in the main text. Rather, I create definitions such as

```
\def\EN#1{\footnote{#1}}
```

and use `\EN{note text}` in the main text so that I can make the decision later to keep using footnotes or switch to having endnotes.

Sometimes I have used both footnotes and endnotes in a document, for instance, footnotes for notes and endnotes for references, or the reverse. In that case, I would define both `\EN` and `\FN` to be footnotes while drafting, and then switch one type to be endnotes when nearing completion of the document. Sometimes I also have not been sure if I was going to use `BIBTEX` and the `\cite` command or put the references in footnotes or endnotes. Thus, I typically also do not use `\cite` in the main text but instead use `\CT` which later can be defined to do whatever I decide I want it to do.

Often while developing a book or other complex document I go back and forth several times switching the definitions of `\FN`, `\EN`, and `\CT`. Sometimes the final decision comes based on the publisher's book or journal style despite my preference.

## 4 My second book

I self-published the second book I composed using `LATEX` (*Breakthrough Management*, co-authored with Shoji Shiba). Because the book was self-published, I got to control all the design decisions, for better or for worse.

### 4.1 Different approach to a bibliography

By the time I started this book, I had fallen in love with the convention for `BIBTEX` keys, e.g., `Ackoff81`, that I had used in the previous book. I have never liked the conventional author-year approach to citing references, i.e., [`Ackoff`, 1981], despite its widespread use and the fact that many scholarly journals require it and the *Chicago Manual of Style* strongly recommends it. Numbers alone, e.g., [189], don't give any information to the reader. It is nice to know without actually going to the bibliography which author (at least first author) is being cited and in what year; sometimes that alone is enough for the reader to recall the document being cited.

Thus, I adopted my own convention for labeling bibliography items and manually formatted the bibliography entries. This bibliography was much smaller, the editor I hired had her own ideas about proper formatting for bibliography entries, and it was easier not to use `BIBTEX`. The beginning of the bibliography file had the following definition to format the bibliography entries and entries such as the example for `Ackoff81` shown below.

```
\def\ref#1#2{\vskip 4pt
  \vbox{\noindent\small
    \hangindent = 1pc \textbf{#1.}}
```

```
\hspace{.05in}#2}}
% using the MLA standard
```

```
\ref{Ackoff81}{Ackoff, Russell L.
  \textit{Creating the Corporate
  Future: Plan or Be Planned For}.
  New York: Wiley, 1981.}
```

which resulted in bibliography entries such as the following:

**Ackoff81.** Ackoff, Russell L. *Creating the Corporate Future: Plan or Be Planned For*. New York: Wiley, 1981.

There was a problem with using this notation. The book was typeset using the Minion set of fonts which includes old style numbers which made a zero in the year part of the notation the same size as a small letter o.

### 4.2 End-of-book notes

By the time I started this book, I also had become convinced that notes should either be at the bottom of pages where they are easy to see, or they should all be at the end of the book where they are relatively easy to find. The reader should not have to hunt for the notes at the end of every chapter.

In this approach, the command at the end of each chapter only reset the note counter to 0 and didn't actually dump any notes. Thus, all the notes from all the chapters are saved up with note numbers repeating from the beginning of each chapter.

Once again I modified `endnotes.sty`. In addition to some small formatting changes to the endnotes themselves, I tweaked the `\theendnotes` macro in `endnotes.sty` (and gave the style my own private name). With my change, when endnotes were dumped the only time at the end of the book, a chapter counter was started at 0 and then incremented by 1 each time a new note 1 went by as part of the endnote dump. The change also prints the text "Chapter" and the appropriate chapter number between the last note of one chapter and the first note of the new chapter.

## 5 My third book

I also self-published the third book I composed, *Visionary Leaders in Manufacturing*, again co-authored with Shoji Shiba, this time using the book option of the `memoir` class. In this case I decided to use the style I must use when writing and editing for the *IEEE Annals of the History of Computing*. This journal's style has all references and notes in one numeric sequence at the end of the article (or book, in this case) in a section called "Notes and references".

I could have just used one long sequence of numbered endnotes. However, then I would have to

do something special<sup>6</sup> not to have duplicate entries in the notes-and-references list when a particular book or article was cited more than once.

Consequently, I decided to use  $\text{BIB}\text{T}\text{E}\text{X}$  and  $\text{\backslash bibliographystyle\{unsrt\}}$  (a numeric list in order of use) for both the the notes and references, as in the following examples from my `biblio.bib` file:

```
@book{Shiba2003,
  author="Shoji Shiba and David Walden",
  title="Breakthrough Management: Principles,
    Skills and Patterns for
    Transformational Leadership",
  year=2006,
  publisher="Confederation of Indian Industry",
  address="New Delhi" }
@misc{ch1a,
  note="Although I have not used quote marks
    here, these paragraphs are copied or
    derived from the CII website\cite{CIIurl}
    and the preface of several VLFM documents;
    and I have kept their spelling in these
    near quotes." }
@misc{CIIurl,
  note="\url{www.cii.in}" }
```

This resulted in the following sorts of entries in the combined notes-and-references list, which I just labeled “References”.

## References

1. Shoji Shiba and David Walden. *Breakthrough Management: Principles, Skills and Patterns for Transformational Leadership*. Confederation of Indian Industry, New Delhi, 2006.
2. `www.cii.in`
3. Although I have not used quote marks here, these paragraphs are copied or derived from the CII website<sup>2</sup> and the preface of several VLFM documents; and I have kept their spelling in these near quotes.

## 6 Final note

Sometimes I wish that I could just pick one useful style for notes and references and use it over and over. However, different situations keep coming up, and sometimes I am too lazy to use something powerful like  $\text{BIB}\text{T}\text{E}\text{X}$  even when it is appropriate to the situation. Fortunately, the set of experiences and approaches I have described here seems to be a good foundation for cobbling together additional approaches as I decide they are needed for particular

<sup>6</sup> E.g., create a label in the endnote for the first instance of the citation and then manually insert that note number as a superscript at the point of the second instance of the citation.

situations. (In general I avoid seeking out and learning new packages as long as I can tweak something I already know to do what I need.)

Still, there are other packages and options to use. For example, in the latest document I wrote before this one, I used  $\text{\usepackage[para]\{footmisc\}}$  to put multiple footnotes on the same line at the bottom of the page. Also, I have recently heard about the `biblatex` package for  $\text{L}\text{A}\text{T}\text{E}\text{X}$  and the `biber` program which I understand are somehow alternatives to using  $\text{BIB}\text{T}\text{E}\text{X}$ ; maybe I *will* try something new for my next big project.

## Acknowledgments

I already mentioned Kopka and Daly’s *Guide to L<sup>A</sup>T<sub>E</sub>X* as my basic source of  $\text{BIB}\text{T}\text{E}\text{X}$  information. However, probably my main source of information when struggling to accomplish some new approach is the `comp.text.tex` discussion group; many people answered my questions or had answered prior questions in which I found answers (Boris Veytsman may have been the person who suggested the technique I mentioned in Section 5). I learned the technique in Appendix A from a file Frank Mittelbach sent to me.

Thank you to Karl Berry and Barbara Beeton for their editorial work on this note.

## Appendix A: $\text{BIB}\text{T}\text{E}\text{X}$ without a separate file

If you have only a few references in a document and don’t want to bother creating a separate `.bib` file, you can use the following technique.

Begin the file (before  $\text{\documentclass}$ ) with a `filecontents` environment; for example:

```
\begin{filecontents}{\jobname.bib}
@article{Abell93,
  author="Thomas E. Abell and Dawn Dougherty Fitzgerald",
  title="{HP}'s Quality Management System:
    {CEO} Roundtable Report",
  journal="The Center for Quality of Management Journal",
  volume={2}, number={3},
  month={Summer}, year={1993}, pages={3--4} }
...
```

Then put a command such as

```
\usepackage[numbers]{natbib}
```

in the preamble. Finally, at the end of the file put commands such as:

```
\bibliography{\jobname}
\bibliographystyle{plainnat}
```

Then you can cite the bibliographic entries in the normal way, e.g.,  $\text{\cite{Abell93}}$ .

◇ David Walden  
<http://www.walden-family.com>

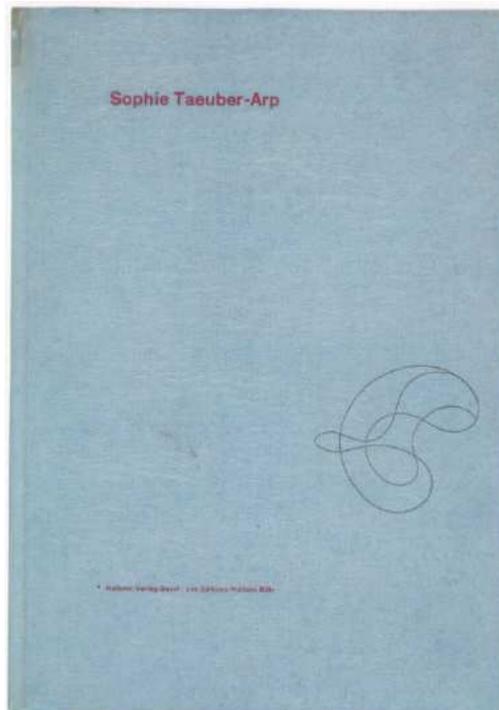
## Sixty years of book design at St. Gallen, Switzerland

Paul Shaw

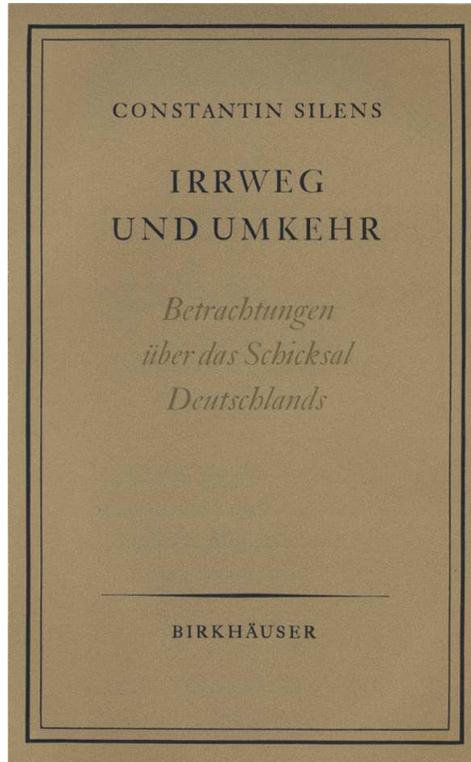
Editor's note: The worlds of T<sub>E</sub>Xnicians and book designers sometimes overlap. T<sub>E</sub>X and friends are a beautiful set of programs, but they are means to an end: the creation of beautiful texts, both in print and on screen. Thus a magazine about visual culture and design, *Print* (<http://www.printmag.com>) may be of interest to TUG members. As an introduction to this magazine we reproduce here, with permission, an article from its online newsletter *Imprint* (<http://imprint.printmag.com>). The online version contains many additional links.

Fifty-five years ago Swiss design was at a crossroads. In “Über Typographie” (*Schweizer Graphische Mitteilungen*, April 1946) Max Bill urged Swiss designers to follow “‘asymmetric’ or organically formed typography”, to reject “the conventional text-image of axial symmetry” and the retreat into historicism that it represented. Jan Tschichold’s rebuttal, “Glaube und Wirklichkeit” (*SGM*, June 1946), repeated his assertion that “The New Typography has not yet been superseded, but it has proved itself to be suitable only for advertising and jobbing. For the book, and particularly for literature, it is completely unsuitable.” He defended the need to design some books “in the manner of traditional typography” while allowing that others might be more suitably done in Bill’s “functional” typography. The Swiss design that won worldwide recognition in the decades following the Second World War followed the path laid down by Bill rather than that of Tschichold. “Swiss” design became the source of “the International Typographic Style” associated with sans serif type—usually Helvetica, a face that ironically most Swiss designers rejected—and grids. (Both essays have been translated into English and published, accompanied by “Über moderne Typographie”, an essay by Paul Renner attempting to mediate the conflict, and contextual essays by Christopher Burke and Robin Kinross in *Typography Papers* 4 (2000), pp. 57–90. Sadly, the issue is out of print.)

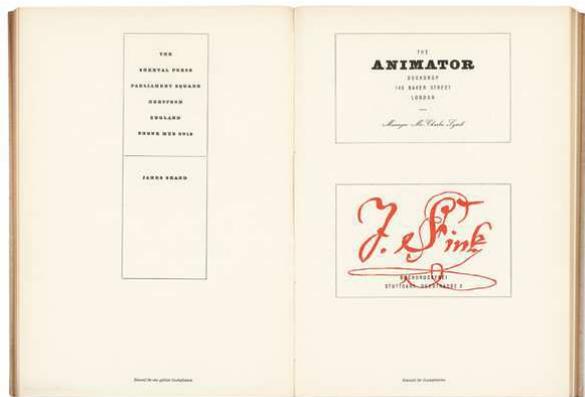
Tschichold’s plea for “the right to work in the way that I find best”, whether “newly revived traditional typography” or “functional typography”, although ignored by the graphic design community, took root among book designers, especially those in St. Gallen, a town fifty miles east of Zurich. It is an appropriate place since St. Gallen is renowned for



*Sophie Taeuber-Arp* by Georg Schmidt (Basel: Holbein-Verlag, 1948). Binding, title page spread and two double page spreads. Design by Max Bill.



*Irrweg und Umkehr* by Constantin Silens (Zurich: Birkhäuser Verlag, 1945). Jacket. Design by Jan Tschichold.



*Typo-Graphik: Studien und Versuche* by Imre Reiner (St. Gallen: Zollikofer AG, 1943). Spread designed by Imre Reiner.



*Typo-Graphik: Studien und Versuche* by Imre Reiner (St. Gallen: Zollikofer AG, 1943). Spread designed by Imre Reiner.

the Abbey Library of the Benedictine monastery of St. Gall (established 747) which houses one of the most important collections of early Medieval manuscripts in the world. It is also the home of Zollikofer, a printer founded in 1789 that became a publisher as well, first of newspapers in 1803, then of the trade journal *Schweizer Graphische Mitteilungen* in 1884, and of books in 1943. (Zollikofer Druck AG is now part of Ringier AG, an international media company.)

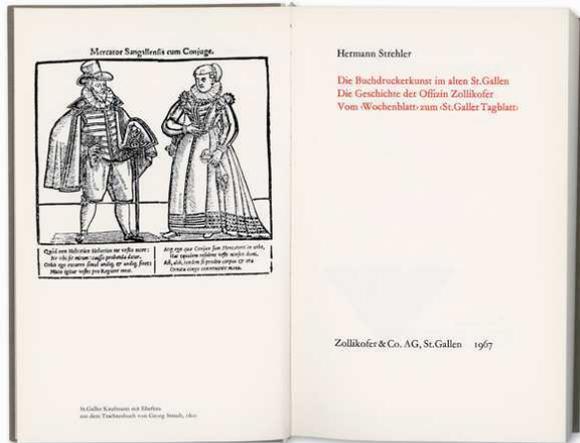
The first book from Zollikofer Verlag was *Typo-Graphik: studien und versuche* by Imre Reiner (1900–1987), a Hungarian-born type designer, book designer and illustrator. Appropriately, it is the first book on display in the exhibition *60 Years of Book Design at St. Gallen, Switzerland* at the AIGA National Design Center in New York City. Reiner studied at the Kunstgewerbeschule (Am Weissenhof) in Stuttgart under F.H.E. Schneider, a charismatic teacher who followed a middle path between the modernism of the Bauhaus and *die neue typographie* and the traditional

book design of contemporaries such as E.R. Weiss and F.H. Ehmcke. This was similar to the attitude that Rudolf Hostettler (1919–1981), the progenitor of the “St. Gallen” style, adopted in the wake of the Bill/Tschichold debate.

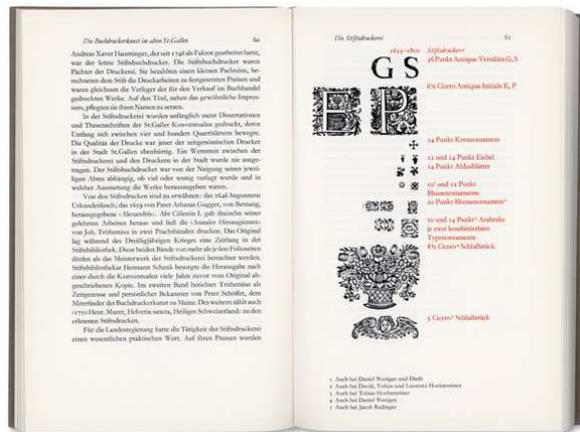
Hostettler, the designer and editor of *SGM* at the time of the debate, began his career in 1943 at Zollikofer as a layout artist for the journal as well as for the newspaper *St. Galler Tagblatt*. In 1951, when *SGM* merged with *Typographische Monatsblätter* (TM) and *Revue Suisse de l’Imprimerie* (RSI), he became the chief editor, a position he held until his death. From 1952 on TM became one of the primary outlets for the new Swiss style of design, the magazine that showcased the work of Robert Bächler, Emil Ruder, Karl Gerstner and others. At the same time Hostettler, with Hermann Strehler, established an imprint for *SGM* which published books about the graphic arts in a new traditional style. Three of the books were written by Hostettler himself: *The Printer’s Terms* (1949), *Type: A Selection of Types* (1949) and *Klassierung der Schriften* (1964). Hostettler was thus a man in the middle in the Swiss typographic wars, someone who was able to remain on good terms with protagonists on both sides.

The AIGA exhibition includes all of these books by Hostettler along with Strehler’s *Die Buchdruckerkunst im alten St. Gallen* (1967) which he designed. With the exception of squarish *Type*, these books typify the St. Gallen style. Jost Hochuli, at the event organized by AIGA NY on June 16 to celebrate the exhibition, objected to the idea that there is a “St. Gallen style” yet there is no doubt that the books on display (as well as others by Hochuli himself) share some common design attributes and a similar design sensibility. Following Tschichold, the St. Gallen attitude is that books are for reading above all else. Thus, content comes before design. Depending upon the content, the design may be symmetrical (traditional) or it may be asymmetrical or it may be a combination of both. The typeface may be serifed or sans serif or the two may be used together. Above all, as Hochuli has reiterated on several occasions, the designer must not follow dogma. This open approach is the legacy of Hostettler that Hochuli has passed down in his work, his teaching and his writings. The books in the AIGA exhibition tend to mix symmetry with asymmetry and often to combine serif and sans serif types.

The emphasis on books for reading influences their physical appearance. Here again, Tschichold is a guiding spirit. He emphasized the handiness of a book, decrying formats that were too wide, too large or square:



*Die Buchdruckerkunst im alten St. Gallen* by Hermann Strehler (St. Gallen: Zollikofer, St. Galler Tagblatt, 1967). Title page designed by Rudolf Hostettler.



*Die Buchdruckerkunst im alten St. Gallen* by Hermann Strehler (St. Gallen: Zollikofer, St. Galler Tagblatt, 1967). Title page designed by Rudolf Hostettler.

There are three arguments that speak against books whose format approaches the equal-sided rectangle. The first is simply handiness. It is difficult for an unsupported hand to master a square book—even more difficult than to handle the ugly A5 format. The second argument concerns storage. If these books are wider than 24 cm (9 1/2 in.) they must be put down flat. Yet books should be capable of being stood upright on a shelf so that they can be found quickly and used. For the final argument, I have to make a little detour. It is the hinges on either side of the spine that hold the inner book, the book block, in position. If the inner book is heavy—regrettably often the case—then the face of the book will drop, touch the shelf and begin to collect dust, a



*Schriftenverzeichnis* (St. Gallen: Zollikofer, St. Galler Tagblatt, 1989). Spread designed by Max Koller.

thing the edges of the cover are supposed to prevent. The longer the spine of the book relative to its width, the better the inner book will remain in position.

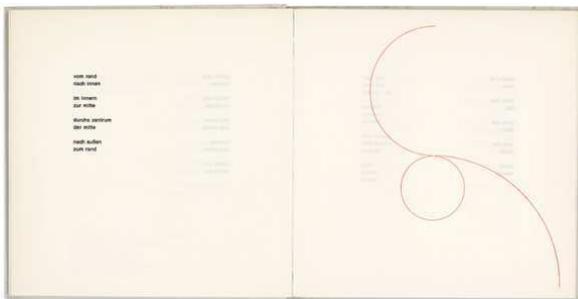
From “On Books that Are Too Wide, Too Large, or Square” (1975), reprinted in *The Form of the Book: Essays on the Morality of Good Design* (Vancouver and Pt. Roberts, Washington: Hartley & Marks, 1991), p. 167

In the same essay Tschichold also argued for books that were not heavy, singling out art paper (the older term for coated paper) as something to be avoided whenever possible. In the AIGA exhibition there is one book in landscape format, one in a near-square format and only two that are true squares. These books look out of place amidst the others, many of which are strikingly narrow. For instance, *Klassierung der Schriften* is 115 × 180 mm, *Die Buchdruckerkunst in alten St. Gallen* is 160 × 250 mm, *Schriftenverzeichnis* (1989) is 162 × 297 mm, *Schreibwerkstatt St. Gallen* (1986/1987) is 134 × 225 mm and the Thomas Mann books (2002) are 125 × 205 mm. (For American readers, the standard 8 1/2 × 11 book is equal to 216 × 279 mm.) These books are not only small but they are light. With one possible exception none are printed on coated paper. (The exhibition includes copies of many of the books on display under glass that visitors can handle.)

The two square books are from Tschudy-Verlag, founded by Henry Tschudy (1882–1961) as a spin off of his printing company H. Tschudy & Co. (*Punkt, Cicero und Kaviar* (1982), a book celebrating Tschudy’s life was written and designed by Hochuli on



*33 konstellationen* by eugen gomringer (St. Gallen: Tschudy-Verlag, 1960). Title page spread. Design by Max Bill. Set in Neue Haas Grotesk.



*33 konstellationen* by eugen gomringer (St. Gallen: Tschudy-Verlag, 1960). Double page spread. Design by Max Bill. Set in Neue Haas Grotesk.

the occasion of the centennial of his birth.) They are part of a 44-book series edited by Hans Rudolf Hilty between 1959 and 1964 called *Reihe Quadrat-Bücher*. No. 11 in the series is *33 konstellationen* (1960) by Eugen Gomringer, the Concrete poet, designed and illustrated by Max Bill. It is set in Neue Haas Grotesk (which later that year was renamed Helvetica by D. Stempel AG). No. 15/16 is *Die dritte Generation: 42 Junge Schweizer kunstler...* (1960) by Hilty was designed by Hochuli's teacher Willy Baus. (For Hochuli's tribute to him see *Typotron* 6. It is set in Akzidenz Grotesk and uses a self-evident grid. Unfortunately, not enough pages are on display to fully appreciate the grid.

These books are from St. Gall but they are far from the St. Gall "style". Equally far though are



*Die dritte Generation: 42 junge Schweizer Künstler* (St. Gallen: Tschudy-Verlag, 1960). Title page spread. Design by Willi Baus.



*Ein ABC um alte Segelschiffe* by Albert Saner (St. Gallen: Tschudy-Verlag, 1958). Spread with woodcuts by Albert Saner. Design by Willi Baus.

two older books by Baus for Tschudy-Verlag: *Von der Kunst und vom Künstler* by Josef Weinheber (1954) and *Ein ABC um alte Segelschiffe* by Albert Saner (1958). Both are set in typefaces by Baus' mentor Rudolf Koch, Marathon for the former and the blackletter Jessenschrift for the latter. (Marathon, one of Koch's least known faces, was singled out for opprobrium by Tschichold in "Glaube und Wirklichkeit.") They are beautiful books—the balance of the text block in Jessenschrift with the woodcut illustrations of ships by Saner in *Ein ABC* is outstanding—despite looking like strays from an Arts & Crafts exhibition.

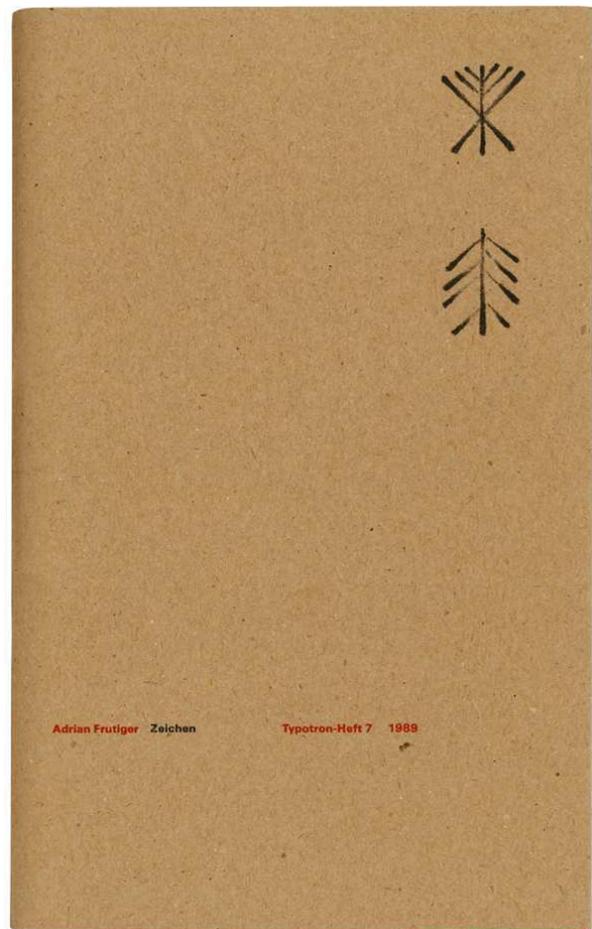
The St. Gallen style of Hostettler, Hochuli and their students tends to be less self-conscious as well as undogmatic. Typefaces are chosen for their readability, rather than their trendiness. Most are seriffed faces in a classical manner, but not wholly of the past. Along with Bembo, Stempel Garamond and Sabon, there is Trump Medieval, Scala, Trinite, Collis and Rialto. (The latter three are fonts rarely seen in the United States.) The preferred sans serif is Univers,

but there is also Franklin Gothic, Futura and Scala Sans — and two instances of Helvetica, a face that Hochuli has studiously avoided in his book designs. Text blocks tend to have indented paragraphs — as Tschichold argued for — rather than the skipped lines typical of Bill and his followers. Grids are present but rarely overtly so. Despite the small size of most of these books, they have generous margins. The side margins, used often for side notes, keep one’s thumbs from covering the main text. Two attributes seem to be specifically typical of Hochuli and his disciples, though the first may be traceable to Hostettler: a preference for anchoring the running heads with a thin rule, and the common use of solidly colored endpapers.

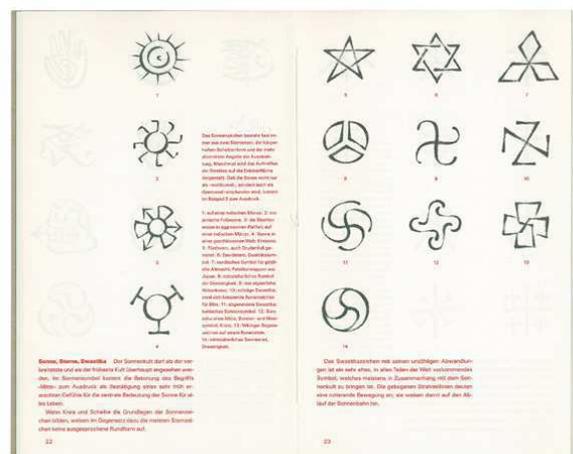
Although Tschichold is the godfather and Hostettler is the father of the St. Gallen style, it is Jost Hochuli (b. 1933) who is the pivotal figure in the exhibition. It is Hochuli who organized the show, handled the installation and designed the accompanying catalogue. After studying at the Kunstgewerbeschule St. Gallen where Willi Baus (1909–1985) was one of his teachers, he worked for Zollikofer under Hostettler. He then spent the year 1959 studying with Adrian Frutiger at the Ecole Estienne in Paris. In 1979 he co-founded the cooperative publisher VGS Verlagsgemeinschaft St. Gallen for whom he has been the chief book designer until recent years. For Typotron AG, a typesetter and printer in St. Gallen, he designed (and sometimes wrote) a series of small booklets from 1983 to 1998 as promotional efforts. Since 2000 he has designed a similar series of booklets for Edition “Ostschweiz”.

Hochuli paid homage to Hostettler, his friend as well as mentor, in the first Typotron book, *Epitaph für Rudolf Hostettler* (1983), a book not on display in the AIGA exhibition. Instead, *Zeichen* by Adrian Frutiger (Typotron 7, 1989), Hochuli’s other mentor, is on display. Hochuli’s other works in the show are *Herbstlaub* by Rudolf Widmer (Edition Ostschweiz 4, 2003) and two titles from a 58-book series of the works of Thomas Mann (Berlin: S. Fischer Verlag, 2002).

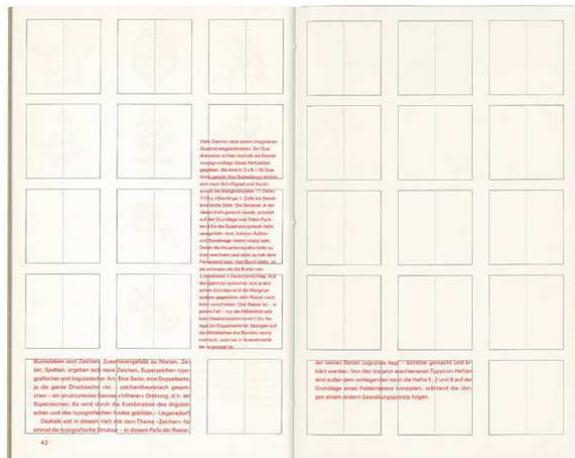
Hochuli has had a slow but steadily widening impact on modern book design and typography in the past thirty years. First, through his work, especially that for VGS and Typotron; then through his teaching at the Schule für Gestaltung in St. Gallen; and more widely through his publications on typography and book design and the traveling exhibitions such as this one that he has orchestrated. In the late 1980s Agfa Compugraphic (now Monotype Imaging) published *Detail in Typography* (1987), *Designing Books* (1990) and *Alphabugs* (1990), each in several



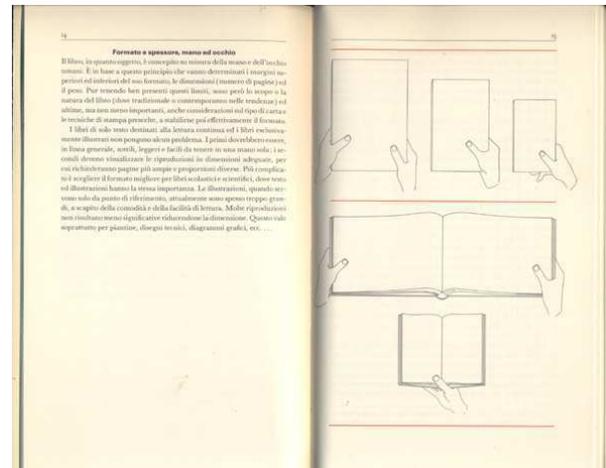
*Zeichen* by Adrian Frutiger (St. Gallen: Typotron, 1989). Cover, designed by Jost Hochuli.



*Zeichen* by Adrian Frutiger (St. Gallen: Typotron, 1989). Title page. Designed by Jost Hochuli.



*Zeichen* by Adrian Frutiger (St. Gallen: Typotron, 1989). Spread showing the book's grid structure. Design by Jost Hochuli.



*Come si fa un libro* by Jost Hochuli (Wilmington, Massachusetts: Agfa Compugraphic, 1989). Spread showing the importance of designing books that can be held comfortably by readers. Design by Jost Hochuli.

languages. *Designing Books* became the central part of *Bücher machen* (St. Gallen: VGS, 1996) — translated into English as *Designing Books: Practice and Theory* (London: Hyphen Press, 1996; reprinted 2007 but now out of print) — and *Detail in Typography* was reissued in English in an updated version in 2009 by Hyphen Press. An overview of Hochuli's career can be found in *Jost Hochuli: Printed Matter, Mainly Books* (Sulgen, Switzerland: Niggli, 2002). Through these books, especially the English language edition of *Designing Books*, he has become probably the most influential theorist of book design since Jan Tschichold.

The AIGA exhibition includes several of his students, most notably Roland Steiger (b. 1970) of TGG Hafen Seen Steiger — the other partners are Dominik Hafen (b. 1967) and Bernhard Senn (b. 1965) — and Gaston Isoz (b. 1969). Steiger is responsible for *Spitzen umschreiben Gesichter* (1997), *Cultura Sangallensis* by Peter Ochsenbein (2000) and *Leopold Iklé* by Anne Wanner-JeanRichard (2002), the latter two another pair of narrow books. *Spitzen umschreiben Gesichte* is also narrow but it opens vertically, something which Tschichold would surely have criticized, despite the fact that this structure guarantees a strong spine. *Cultura Sangallensis* is unusual in this show for having traditional footnotes. Isoz, who works in Berlin, is represented by *Arbeiterlyrik 1842–1932* (2003) and *Geschichte der Partei des gemäßigten Fortschritts im Rahmen des Gesetzes* (2005). Both of these deviate from the pure St. Gallen style in the use of an exposed binding without a headband. In the latter the boards are flush with the pages, making opening the book and

riffling its pages difficult. In its style of illustration it also shows more of the influence of Oldrich Hlavsa (1909–1965), author and designer of the three-volume *Typographia* (1976, 1981 and 1986).

Another of the teachers at the *Schüle für Gestaltung* in St. Gallen was Max Koller (b. 1933). Koller worked for Zollikofer and later for Gerstner + Kutter and its successor GGK. He is responsible for *Schriftverzeichnis* (1989), the Druckerei Zollikofer type specimen with the lovely patterned letters in the show. Another example of his work for Zollikofer can be found at <http://wiedler.ch/felix/books/story/146>.

There are two other books in the AIGA exhibition that are worth mentioning: *Künstlerheft* by Ian Anüll (St. Gallen: Vexer Verlag, 1987) and *Rund ums Blaue Haus* edited by Ernst Ziegler (St. Gallen: Ophir-Verlag, 1993). Both fit into the St. Gallen style even though they are not designed by individuals with direct connections to Hostettler, Koller or Hochuli. The first is designed by sculptor and painter Josef Felix Müller (b. 1955) for his own Vexer Verlag, a publishing house specializing in books by artists. *Künstlerheft* is one of the most covetable books in the entire exhibition with its translucent pages that change the images as the pages turn. *Rund ums Blaue Haus* is notable for the way designer Antje Krausch of Atelier Tachezy, Kleger & Partner has used an abstract motif of blue lines derived from the timbers of the titular blue house.

*60 Years of Book Design at St. Gallen, Switzerland* is a quiet show, one that requires time and patience to reveal its treasures. Hochuli's exhibition



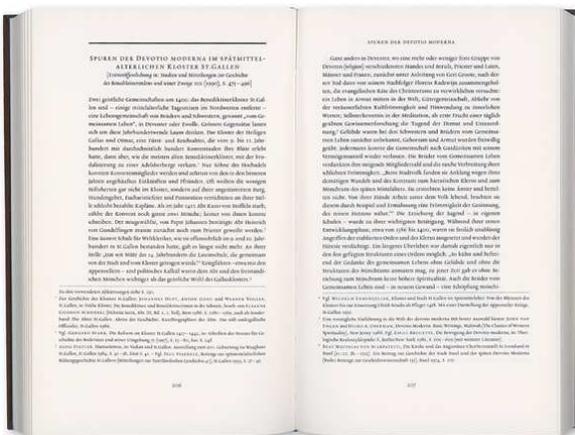
*Cultura Sangallensis* by Peter Ochsenbein (St. Gallen: Verlag am Klosterhof, 2000). Title page, designed by Roland Steiger of TGG Hafen Senn Steiger.



*Künstlerheft* by Ian Anüll (St. Gallen: Vexer Verlag, 1987). Spread. Designed by Josef Felix Müller.



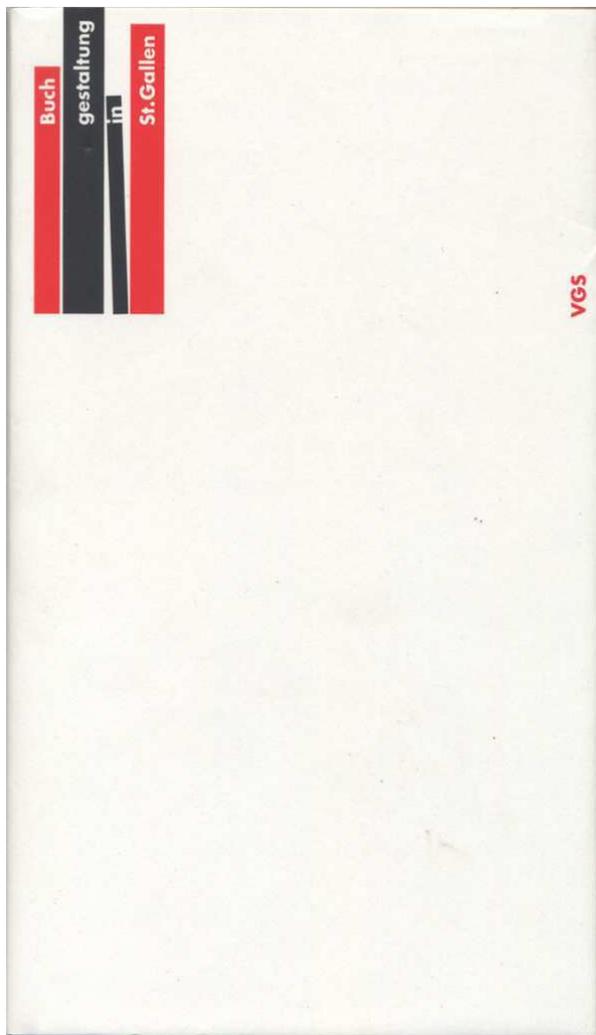
*Cultura Sangallensis* by Peter Ochsenbein (St. Gallen: Verlag am Klosterhof, 2000). Spread designed by Roland Steiger of TGG Hafen Senn Steiger.



*Cultura Sangallensis* by Peter Ochsenbein (St. Gallen: Verlag am Klosterhof, 2000). Spread designed by Roland Steiger of TGG Hafen Senn Steiger.

design, a model of simplicity and restraint, encourages close reading of the books on display. The cases, arranged in a zig-zag fashion, are lined in gray to make the paper of each book stand out. There are multiple copies of each book so that the viewer can see them from a variety of perspectives — cover, title page, sample spread for instance — as should be done for an interactive, three-dimensional object. The downside is that this means only a few titles are in each case, making for a small show. But the intent of Hochuli and the organizers is not quantity but quality and that is something that they have achieved.

The catalogue mentioned above is available for sale at AIGA, but unfortunately it is in German only. For book designers and die-hard book lovers it is still worth purchasing since it is itself an example of the St. Gallen style of bookmaking — small and light — and reasonably priced at \$20. The one quibble I have with both it and the exhibition labels — written by Hochuli himself and usually sharply observed — is that the typefaces used in each book are not routinely noted. For the record, Collis by Christoph Noordzij (TEFF, 1993) is used in the catalogue.



*Buchgestaltung in St. Gallen* by Roland Früh  
(St. Gallen: VGS Verlagsgemeinschaft St. Gallen,  
2008). Jacket. Design by Jost Hochuli.

- ◇ Paul Shaw  
Reprinted with permission from *Print*,  
July 7, 2011
- © F+W Media Inc. 2010
- [http://imprint.printmag.com/  
inspiration/sixty-years-of-book  
-design-at-st-gallen-switzerland](http://imprint.printmag.com/inspiration/sixty-years-of-book-design-at-st-gallen-switzerland)

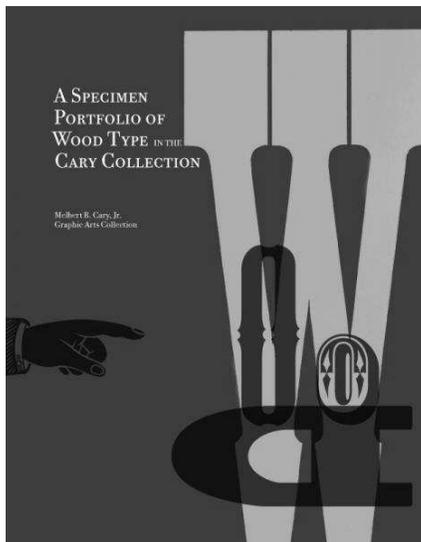
## TUG Institutional Members

- American Mathematical Society,  
*Providence, Rhode Island*
- Aware Software, Inc., *Midland Park, New Jersey*
- Banca d'Italia,  
*Roma, Italy*
- Center for Computing Sciences, *Bowie, Maryland*
- Certicom Corp., *Mississauga, Ontario, Canada*
- CSTUG, *Praha, Czech Republic*
- diacriTech, *Chennai, India*
- 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*
- LAMFA CNRS UMR 6140, *Amiens, France*
- MacKichan Software, Inc.,  
*Washington/New Mexico, USA*
- Marquette University, Department of  
Mathematics, Statistics and Computer Science,  
*Milwaukee, Wisconsin*
- Masaryk University, Faculty of Informatics,  
*Brno, Czech Republic*
- MOSEK ApS, *Copenhagen, Denmark*
- New York University, Academic Computing Facility,  
*New York, New York*
- Springer-Verlag Heidelberg, *Heidelberg, Germany*
- Stanford University, Computer Science Department,  
*Stanford, California*
- Stockholm University, Department of Mathematics,  
*Stockholm, Sweden*
- University College, Cork, Computer Centre,  
*Cork, Ireland*
- Université Laval, *Ste-Foy, Québec, Canada*
- University of Ontario, Institute of Technology,  
*Oshawa, Ontario, Canada*
- University of Oslo, Institute of Informatics,  
*Blindern, Oslo, Norway*

---

**Book review: *A Specimen Portfolio of Wood Type in the Cary Collection***

William Adams



*A Specimen Portfolio of Wood Type in the Cary Collection*. RIT Cary Graphic Arts Press, Rochester. 2011. 305 pp. Spiral-bound, US\$19.95. ISBN 978-1-933360-44-7.

Foreword by R. Roger Remington, Curator's Note by David Pankow, and Introduction by David P. Wall, whose master's thesis forms the basis of the work.

This is a fascinating book providing an interesting cross-section of wood type fonts. It is at once a valuable record of what was saved as well as a testament to how tenuous the survival of many of these types was and a memorial of how much was lost — showcasing 412 out of some 20,000 wood type fonts which are believed to have existed.

Although the book references many other publications and has an excellent bibliography, it is a stand-alone work, suitable for even a beginning graphic designer, typographer or student. Rather than rely on an external classification system, the

types are cataloged in accordance with a scheme modeled on an ISO standard (9541-1, Annex A, Electronic Font Interchange) which is presented in its entirety at the beginning of the book. These classification pages represent a useful text in and of themselves, providing at once a broad overview and specific notes on typeface forms.

Types are shown on spreads, one or more fonts per spread, with the verso being used for an expansive actual-size exemplar (save for a few fonts which are too large even for a full-page showing) and identification and sizing notes, while the recto shows a complete sample of the fonts in the collection — in many instances, a complete alphabet, the classification, identification and notes on the manufacturer and date. It is the incomplete showings which are most interesting, however, offering fascinating glimpses into how the typefaces were actually used, or how it is that they might have been saved from destruction (*AUCTION* on pg. 19, a dollar sign and set of numerals on pgs. 75, 79 and 101, six letters which might spell out “TOUCAN” on pg. 129).

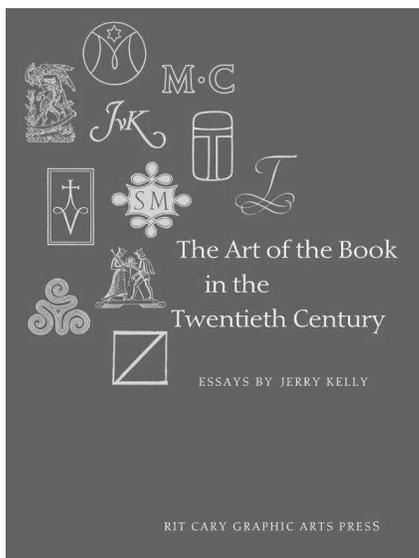
Younger designers and students will be surprised at the variety of the typefaces, their not being limited to just the stereotypical condensed slab serifs of Hollywood Western “Wanted” posters and encompassing many designs not represented in Adobe's *Wood Type Font Collections*. Old hands will be pleased to see what has been preserved and may be moved to search through some old drawers or storage chests and find some pieces to donate and share. Although Rochester Institute of Technology plans to eventually provide online access to digital images, the immediacy and ease of reference of this excellently organized text makes purchasing the very affordable print version an easy decision.

◇ William Adams  
 will dot adams (at) frycomm dot com  
[http://mysite.verizon.net/william\\_franklin\\_adams](http://mysite.verizon.net/william_franklin_adams)

**Book review: *The Art of the Book in the Twentieth Century***

Boris Veytsman

Jerry Kelly, *The Art of the Book in the Twentieth Century*. RIT Cary Graphic Arts Press, Rochester. 2011. 200 pp. Hardcover, US\$39.95. ISBN 78-1-933360-46-1.



This book does not mention  $\TeX$  on any of its two hundred pages. However, I think publication of the review in the pages of *TUGboat* is justified. The famous exhortation [2, p. 303] “Go forth now and create masterpieces of the publishing art” is as much a part of *The  $\TeX$ book* as the explanation of `\noexpand` or `\futurelet`. Thus discussions about the art of the book are relevant for TUG publications, and we will continue reviewing books about publishing, typography and typesetting. This collection of essays by Jerry Kelly is a good example of them. Of course, the fact that the last chapter in the book is written about the Wizard of Fonts, permanent honorary board member of TUG, Hermann Zapf, only adds to the allure of this book for  $\TeX$  users.

The last century saw many quick changes in the technology of book making. At its beginning, most typographers used letterpress. Hot metal machines, phototypesetting, offset printing and other inventions came and went, to be supplanted today by digital typography. Book making was deeply influenced by these changes. The tumultuous history of society and art in 20th century was another source of influence for the old art of typography. Thus writing about book making in this period is a very difficult — but a very interesting — task.

Boris Veytsman

Jerry Kelly approaches his subject with a biographical method. His book is a collection of short essays about the leading typographers of the last century: Daniel Updike, Bruce Rogers, Joseph Blumenthal, Stanley Morison, Francis Meynell, Giovanni Mardersteig, Jan van Krimpen, Jan Tschichold, Max Caffisch, Gotthard de Beauclair and Hermann Zapf. Each essay has a length (and is written in a style) comparable to an entry in a typography encyclopedia and is accompanied by a selection of reproductions of the typographer’s masterpieces. Of course, the chosen format does not allow the author to do a deep analysis of the life and art of his subjects, but the book features a good bibliography section where a reader can find materials for further reading. Even these short bios, however, can lead to interesting thoughts about the way the works of the masters reflected their lives and the society around them.

For example, Kelly says that the Great Depression disrupted the fine printing business model, and the typographers tried to cope with this. This is an interesting story — for example, how much was the success of the famous *Nonesuch Press* caused by their large inventory of titles and the novel ways to lower the cost? — which probably deserves a separate book.

Further, Kelly mentions the arrest and imprisonment of Tschichold by the Nazis and posits that this arrest might be one of the causes of the famous “conversion” of Tschichold to the more classic typography. Again, it is only a part of the story. Initially these “revolutionary” artists of 1920s sometimes identified with the nascent totalitarian ideologies of Communism and Nazism. Bauhaus and Tschichold’s *die neue Typographie* in Western Europe corresponded to VKHUTEMAS Art School and the works of Lisitzky and other Constructivists in Russian book design (see [1] for a discussion). However, when these regimes came to the power, both they and the artists became disappointed in each other. The latter discovered that the former were not especially eager to overthrow the “old art” and preferred the pseudo-classicist imitation of the styles of previous epochs.

As seen from these two examples, Jerry Kelly just mentions such topics, and leaves a more detailed discussion for other books. However, the fact that his essays might spark thoughts about the evolution of book arts speaks about the quality of his book.

Of course, tastes differ, and some readers may suggest other lists of influential typographers. Kelly himself concedes that “The selection — both of the designers and their works — is somewhat arbitrary.” However, the selection of subjects and the style of

the book are the sole prerogative of the author, and Kelly certainly deserves the right to develop his own vision of the history of typography of 20th century.

While the essays themselves are rather short (but interesting), the illustrations are superb. The author complains that

Unfortunately reproductions, no matter how carefully produced, can only partially convey the effect of a finely produced book: we cannot reproduce the paper it was printed on, nor create a facsimile of the particular impression of type on that paper, to say nothing of binding designs and materials, or even elements as basic as the size, heft, and the overall “feel” of a particular volume (all characteristics which are carefully considered in fine printing).

Nevertheless his illustrations are a great boon for many of us who cannot visit all the various museums and libraries where the gems of the typographic art are kept — or peruse them at will. The book contains 104 full page plates and 12 smaller illustrations in the Introduction, plus portraits of all the subjects of his essays. One can spend hours just looking at these illustrations. They are carefully chosen and reproduced with great care. While the reader cannot see the gold leaf roundels in Bruce Rogers’ *Odyssey* (plate 19), or appreciate the full size of Tschichold’s poster (plate 71), still the book manages to give the feeling of these masterpieces.

The author approvingly quotes Morison’s dictum “The history of printing is in large measure the history of the title-page.” Accordingly, most of his plates are title pages of the books by great designers, but Kelly also gives nice examples of body pages and spreads. Many T<sub>E</sub>X users are interested in the design of mathematical books; plate 20 reproducing a page of Euclid’s *Elements* printed by Random House in 1944 might provide some food for thought.

Of course a book about typographic art should in itself be an example of typographic art. This book is tastefully designed and produced. It is typeset in Aldus and carefully bound. It is a pleasure to open and read.

My only peeve is the way paragraphs are separated: there is neither indentation nor separating

vertical space, and the only indication of the paragraph start is the shorter last line of the preceding paragraph (to achieve this effect in T<sub>E</sub>X one sets `\parskip=0pt` and `\parindent=0pt`). Jan Tschichold with his usual forcefulness called this style “an ill-conceived mannerism” and “dubious practice” [3, p. 106]. To tell the truth, though, even he conceded that it might be acceptable if the compositor takes care to make the last lines of the paragraphs short enough to signal the end to the reader — which is the case for this book.

This book is a wealth of useful reading for a student of typographic art or anybody interested in the history of publishing. It is a beautiful book which will be a good item for a book collector. For a lavishly illustrated and lovingly published book it is relatively inexpensive and definitely worth buying. (The publisher is offering a discount to TUG members. Discount information is given in the TUG members’ area: <https://www.tug.org/members>.)

TUG members might also be interested in other books from the same publisher — including several by Hermann Zapf. They can be browsed at <http://carypress.rit.edu>.

## References

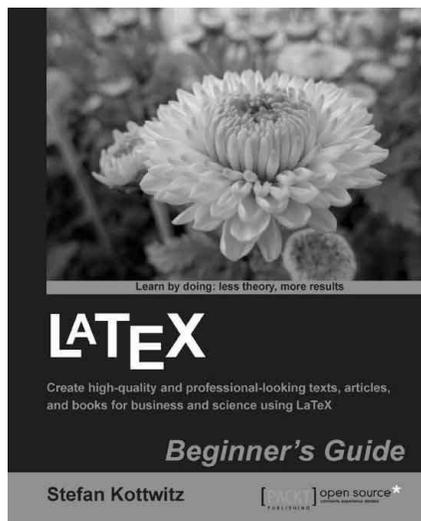
- [1] Alan Bartram. *Bauhaus, Modernism and the Illustrated Book*. Yale University Press, New Haven, CT, 2004.
- [2] Donald Ervin Knuth. *The T<sub>E</sub>Xbook*, volume A of *Computers & Typesetting*. Addison-Wesley Publishing Company, Reading, MA, 1994. Illustrations by Duane Bibby.
- [3] Jan Tschichold. *The Form of the Book. Essays on the Morality of Good Design*. Hartley & Marks, Point Roberts, WA, 1991. Robert Bringhurst, editor.

◇ Boris Veytsman  
 Computational Materials Science  
 Center, MS 6A2  
 George Mason University  
 Fairfax, VA 22030  
 USA  
 borisv (at) lk dot net  
<http://borisv.lk.net>

**Book review: *L<sup>A</sup>T<sub>E</sub>X Beginner's Guide***

Boris Veytsman

Stefan Kottwitz, *L<sup>A</sup>T<sub>E</sub>X Beginner's Guide*. Packt Publishing Birmingham-Mumbai, 2011. 314 pp. Paperback, US\$44.99.



To tell the truth, I was rather skeptical when I agreed to review this book. There are already good guides for L<sup>A</sup>T<sub>E</sub>X beginners, including the book by the author of the language [4], the excellent introduction [3], the popular free course [5], the mathematics-oriented books [1, 2] and many, many others. How can yet another introductory L<sup>A</sup>T<sub>E</sub>X book be different?

I was completely wrong. The new book by Stefan Kottwitz is very much unlike other introductory L<sup>A</sup>T<sub>E</sub>X courses. The author, evidently, wanted to make a L<sup>A</sup>T<sub>E</sub>X book for the new generation — the generation of Twitter and mobile devices. The book's advantages and disadvantages stem from this.

First, the contents. Besides the topics one would expect from any introductory L<sup>A</sup>T<sub>E</sub>X book — L<sup>A</sup>T<sub>E</sub>X commands, document structure, logical markup etc. — the guide by Kottwitz includes also items traditionally omitted from beginners' books. For example, it explains how to install a T<sub>E</sub>X system. Usually this has been considered a feat reserved for computer gurus (the manual [4] contains references to a *Local Guide* presumably written by one of these lofty sages). The T<sub>E</sub>X Live system makes installation much simpler, and this book helps a novice to understand the necessary steps. *L<sup>A</sup>T<sub>E</sub>X Beginner's Guide* discusses dozens of packages and spends considerable time in explaining their usage in customization of the documents — another topic traditionally relegated to more advanced users. Many of the items discussed in the book are “modern” in the sense that they

became possible or popular in the last decade: getting the full advantage of pdfetex features with the `microtype` package, employing many fonts besides the traditional ones, using Unicode and Latin Modern fonts for European languages, adjusting page dimensions, creating hyperlinks, and a number of other interesting and relevant topics, which are not well covered in other introductory texts. Stefan Kottwitz has been a moderator for several online support forums. This gave him a good understanding of what is important for the modern users. The selection of topics is perhaps the strongest side of the *L<sup>A</sup>T<sub>E</sub>X Beginner's Guide*.

Second, the way this book explains these topics is notably different from the style of many other L<sup>A</sup>T<sub>E</sub>X books. It is a part of the series of *Beginner's Guides* from Packt Publishers. The series is based on the hands-on approach. Its slogan is *Learn by doing — start working right away*; the books shun “boring bits” of theory for practical recipes and examples. Accordingly, this book has a huge number of well annotated code snippets and exercises. An impatient reader who wants to get something *here and now* will be thrilled by this book.

On the other hand, sometimes this refusal to deal with the theory makes the explanations rather superficial, for example, the treatment of the familiar problem of `\footnote` inside a `\section` argument on page 100. Although to be fair, the explanations of this problem in other introductory texts are also more or less handwaving.

The style of the book is rather “chunky”. A topic takes a half page or a page, then comes another topic, then another one. In this way the author covers a large volume of material, but it gives a “Twitter-like” look and feel to the book.

Another decision of the author, to use “cool” slang, seems to be rather controversial. The examples in the book are subtitled *Time for action*, the suggestions for independent work are put under the heading *Have a go hero*, and all the quizzes are, of course, *Pop quizzes*. Evidently the reader should imagine himself or herself a mighty hero, involved in a lot of action, from time to time interrupted by a short quiz.

The typography of the book is also quite different from the typography of most other books about T<sub>E</sub>X. Some people (especially outside of the USA) consider the default styles of L<sup>A</sup>T<sub>E</sub>X to be “too loud”. They should open Kottwitz's book with a certain caution: it may give them a heart attack. The book designer used for emphasis and headers the following: (a) margin changes with and without icons and square brackets on the margins; (b) fake

crop marks; (c) boxes with shadows; (d) really huge fonts; (e) heavy bold fonts; (f) reverse video (white on black); (g) combinations of the above (like huge heavy bold fonts in reverse video). Even folios (page numbers) of this book are boldfaced and in bold square brackets. The typography of this book does not hide like classical typography does: it jumps at you with all the subtlety of a ransom note.

Still, I admit it would be very difficult to find a more subdued typography style corresponding to the “chunky” character of the book. The text itself has these abrupt changes of topics, like PowerPoint (or should I say `beamer`?) slides. It might be argued that prominent typesetting devices are necessary to present this text. Also, the use of these devices is quite consistent throughout the book. Moreover, the designer made a good judgement that traditional justified text with a serifed font would not work with this style of headings and emphasis. Thus the choice of ragged right paragraphs with a sans serif body font seems to be a good call. Still, it is somewhat strange when the author talks about justification and uniform grayness of the pages, when his own book presents anything but.

While the typography of the book may have a certain merit, the illustrations are just plainly bad. The book has many examples of typesetting presented as magnified fragments of `TeX` output. They all look like low resolution screen captures obtained by pressing the `PrintScreen` button. These fragments are unbearably ugly and do not even begin to show the beauty of `TeX` documents. There are many ways to get high resolution `TeX` output at large magnification. The author and editor should have used one of them.

Still, despite its flaws, this is an interesting book. It covers many useful topics not covered elsewhere,

it is well written and is understandable for a novice. This book can be a basis of a crash course in `LaTeX` or as a self-study tool—as long as the student does not use it as an example of the typographic art.

Final note: the publisher is offering a discount to TUG members who want to buy the book (in print or electronic form). The discount code is given in the TUG members’ area: <https://www.tug.org/members>.

## References

- [1] George Grätzer. *Math into LaTeX*. Birkhäuser, Boston, third edition, 2000.
- [2] George Grätzer. *More Math into LaTeX*. Springer, New York, fourth edition, 2007.
- [3] Helmut Kopka and Patrick W. Daly. *Guide to LaTeX: Tools and Techniques for Computer Typesetting*. Addison-Wesley, Boston, fourth edition, 2003.
- [4] Leslie Lamport. *LaTeX: A Document Preparation System*. Addison-Wesley Publishing Company, Reading, Mass., second edition, 1994. Illustrations by Duane Bibby.
- [5] Tobias Oetiker, Hubert Partl, Irene Hyna, and Elisabeth Schlegl. *The Not So Short Introduction to LaTeX 2<sub>ε</sub>, Or LaTeX 2<sub>ε</sub> in 174 Minutes*, April 2011. <http://mirrors.ctan.org/info/lshort>.

◇ Boris Veytsman  
 Computational Materials Science  
 Center, MS 6A2  
 George Mason University  
 Fairfax, VA 22030  
 USA  
 borisv (at) lk dot net  
<http://borisv.lk.net>

---

**An appreciation: *The Art of Computer Programming, Volume 4A***

David Walden

Donald E. Knuth, *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1*. Addison-Wesley, Boston. Jan. 2011. 912 pp. Hardcover, US\$74.99. ISBN 0-201-03804-8.

A book of algorithms relating to computer programming and their analysis (and about problem solving more generally) would not normally be reviewed in this journal about typesetting and fonts. However,  $\text{\TeX}$ , TUG, and *TUGboat* would not exist without the problem Knuth faced in 1976 with the typesetting of the second edition of Volume 2 of *The Art of Computer Programming (TAOCP)*; and thus *TAOCP* is a key part of the history of the  $\text{\TeX}$  world. Many readers of this journal already know this story (told in chapter 1 of Knuth's book *Digital Typography*).<sup>1</sup> Addison-Wesley had gotten rid its Monotype technology in 1963 and could not reproduce with the photo-optical machines of the time the high quality typesetting of the original printings Volumes 1–3 (and new printings of Volumes 1 and 3 done with Monotype machines still found in Europe). Consequently, in 1977 Knuth began developing a new typesetting system to restore the high quality typesetting of books, in particular *TAOCP*. Eventually  $\text{\TeX}$  was available and became popular with various groups of users; and the  $\text{\TeX}$  Users Group and *TUGboat* came into being.

I bought Volumes 1, 2, and 3 of this series immediately after publication of each book in 1968, 1969, and 1973 and used them frequently in my profession as a computer programmer. I also bought new editions of these books as they came out over the years, keeping my complete set of *TAOCP* up to date. Thus, to maintain my complete set of *TAOCP*, I bought Volume 4A immediately upon its publication and have been dipping into it to get an overall sense of it. (As I suspect is the case with many other readers of this series, I have never read a volume completely through, but rather skimmed each book enough to know what was in it and then studied particular sections more deeply as needed for the project on which I was working, or when I just wanted to have some fun reading about algorithms.)

Over the years since the original editions of volumes 1–3 were published, Knuth's original plan for a 7-volume, 1–chapter series has gradually evolved as some the topics of his originally planned chap-

ters have been covered in depth by other books and as the topics covered by some of the chapters expanded dramatically (perhaps partially as a result of Knuth's example of rigorous, comprehensive books describing computer algorithms). Today Knuth hopes to produce five volumes, of which the current volume 4 will have at least three parts (books): <http://www-cs-faculty.stanford.edu/~uno/taocp.html>

Part 1 (that is, book 1) of Volume 4 (the overall volume is on combinatorial algorithms) covers an Introduction, Zeros and Ones (with four subsections) and Generating All Possibilities (with one subsection containing seven subsubsections). Curiously, the second and third subsections on Generating All Possibilities are not due until Volume 4B. Perhaps at 912 pages (and after publication of the groups of pages from the book over the past half dozen or more years as a succession of five fascicles), Knuth or the publisher decided that Volume 4A was already long enough.

As with the previous volumes of *TAOCP*, this book is substantially about the analysis of the algorithms presented and not just a cookbook of algorithms. A reader can either just find what Knuth says is the best method and use it, or can learn why it is a good method, why other methods are not so good, and how to do the math to analyze the performance of one's own situations where the algorithms might be used. The book also includes Knuth's usual sets of exercises and hundreds of pages of answers to exercises.

Of the parts of Volume 4A I have touched on so far, I greatly enjoyed the discussion of Latin and Greek squares (the clearest I have ever read), and I know I am going to enjoy reading more of the discussion on bitwise tricks and techniques, a topic that has always fascinated me. I also have looked at some of the resources on Knuth's web site augmenting discussions in the book (and the reader's own use of the methods Knuth describes and analyzes). I also enjoy skimming pages of Knuth's *TAOCP*, skipping the real math and reading bits of math-and-algorithm history. Section 7.2.1.7, History and further references, looks like it will be particularly fun reading. I never try to work any *TAOCP* exercises but rather will dip straight into the comprehensive answer sections to find additional information I need that is not covered in the main text.

Not being a mathematician myself, I sought out a comment on the book from mathematician (and puzzle master) Bill Gosper<sup>2</sup> (who has four entries in

---

<sup>1</sup> CSLI Publications, Stanford, CA, 1999

<sup>2</sup> [http://en.wikipedia.org/wiki/Bill\\_Gosper](http://en.wikipedia.org/wiki/Bill_Gosper)

---

### Comment from Bill Gosper

I am delighted to report that Knuth is still his usual precise, profound, and playful self.

The book is surprisingly therapeutic—it will help you lose any guilt you may feel over designing and working puzzles.

On page 172 Knuth says: “For example, after Martin Gardner introduced John Conway’s game of Life to the world in 1970, more computer time was probably devoted to studying its implications than to any other computational task during the next several years—although the people paying the computer bills were rarely told!”

However, the above follows his inflammatory remark on page 2: “On the other hand, the exact value of L[angford arrangement]<sub>100</sub> will probably never be known, even as computers become faster and faster.”

Has Knuth any idea how many computational resources will now be expended trying to prove him wrong?

On page 2: “In Section 7.2.2.1 we shall study an algorithm called ‘dancing links,’ which is a convenient way to generate all solutions to such problems.”

And on page 464: A technique called “dancing links,” which we will discuss extensively in Section 7.2.2.1, is used here to remove and restore items from and to doubly linked lists.

At last, my chance to hear it from the Master! Eagerly flipping forward. . . ., 7.2.1.6, 7.2.1.7, Answers to Exercises. ARGHH! To Be Continued!

And to think I had been salivating over page viii: “(See the discussion of NP-completeness in Section 7.9.)” !

The Table of Contents looks positively meager. How could this require 883 pages? Clue: The Index takes 50 pages. Open to one page at random. Can you plow through it in an hour? A day? This is no cookbook. Don’t open it unless you plan to *learn* something. 34.4 percent of the book is Answers to Exercises.

PS, Don’t miss Knuth’s brilliant new twist on “This page intentionally left blank.”

---

the Volume 4A index). Bill’s reply (email of June 3, 2011) is in the sidebar.

Volume 4A looks like the previous volumes (in their latest editions)—the same design and great care with details (the Knuthian way). In my memory, some details have evolved since the first edition of Volume 1. For instance, with each new volume and each new edition (maybe even with new printings), including the middle names of cited people and correct presentation of their names in their own language have become ever more complete.

Knuth’s editor at A-W, Peter Gordon, has stated that the A-W production staff sees the end product

of Knuth’s work; Knuth supplies PostScript files to A-W, which the A-W printer converts to PDFs. The colophon of Volume 4A says, “This book was produced on an HP Compaq 2510p using Computer Modern typefaces, using T<sub>E</sub>X and METAFONT software as described in the author’s books *Computers & Typesetting* (Reading, Mass.: Addison-Wesley, 1986), Volumes A–E. The illustrations were produced with John Hobby’s METAPOST system.” A close look by Karl Berry at a PDF page from the book suggested that Knuth is using `tex|dvips` and his original bitmapped CM fonts, not the Type 1 fonts that are the default in current T<sub>E</sub>X distributions. (While providing the rest of us with a system that has been extended in many ways, Knuth apparently sticks with the original system he created to produce a beautiful edition of *TAOCP*, using his tool to control every pixel on the page.)

In the world of computer historians (i.e., often people who have not been computer professionals themselves), there was some interesting commentary immediately after Volume 4A of *TAOCP* was published. In essence the question (maybe tongue in cheek) was what took Knuth so long, given how profoundly volumes 1–3 impacted the field of computing—why did he delay this most important work to do other things judged less important by computer historians.

In my view, the historians may over-estimate the impact of *TAOCP* on the field of computer programming. Most programmers don’t use the books, and many people don’t think highly of the books as potential textbooks for typical courses for computer programmers. (Volumes 1–3 clearly did have deep impact in their early comprehensive and rigorous coverage of a range of parts of what was becoming the discipline of computer science.)

The historians may also under-estimate the importance of Knuth’s other work. In my view, Knuth in his field is like Picasso in his. He has had multiple simultaneous and serial careers, any of which would be more than the lifetime achievement of most people. Writing *TAOCP* is one of Knuth’s most important achievements, but I don’t think it is singularly important.

As I see it, the first three volumes of *TAOCP* revolutionized how to analyze algorithms for the purpose of accomplishing some task in a computer program. From these books, I learned new algorithms to use in my programming, and I learned about how to think better about methods I and my fellow programmers were already using (sorting, hashing, . . .). (By the way, I agree with Knuth’s often criticized decision to write the books using assembly language for his hy-

pothetical original and more modern machines rather than in a high-level language.)

Volume 4A is not such a revolution because Knuth no longer can give comprehensive coverage (and because he already showed us the path to rigorous analysis of algorithms in Volumes 1–3). As Knuth has explained, after volumes 1–3 of *TAOCP*, the field exploded, and he no longer could do what he set out to do. Nonetheless, Volume 4A is a further example of his stunning ability to understand vast amounts of material, choose interesting parts, and present them in a fascinating way.

As noted, Knuth also felt the need to work on digital typesetting so a revision of Volume 2 of *TAOCP* would not look bad to him. In so doing, he revolutionized digital typesetting and font design. This incidentally gave many mathematicians, physicists, economists, etc., a new way to do their writing and began what is probably the longest running open source software success story. Over the years the breadth of use of  $\text{\TeX}$  et al. has continued to grow (critical editions of literature, non-Latin alphabet writing, etc.), even as commercial typesetting systems with their GUI interfaces have become the norm in the population at large (with these commercial systems gradually adopting most of the algorithms  $\text{\TeX}$  had long ago). Some might argue that  $\text{\TeX}$  and Knuth’s investigations into digital typesetting and font design have had more impact on the world than Knuth’s self-proclaimed masterwork, *TAOCP*.

It also may be that, after the first three volumes of *TAOCP*, Knuth had to regroup to ready himself for the next step in the series.

- He wrote two books organizing and extending the math approach to analysis of algorithm: one with Daniel Green [*Mathematics for the Analysis of Algorithms*, third edition, Birkhäuser, 1990], and one with Ronald Graham and Oren Patashnik [*Concrete Mathematics*, second edition, Addison-Wesley, 1994].
- He created a new, modern hypothetical machine to use for his examples and wrote the book about the machine [*MMIXware: A RISC Computer for the Third Millennium*, Springer-Verlag, 1999].
- He created the Stanford GraphBase system to help with combinatorics problems and wrote the book [*The Stanford GraphBase: A Platform for Combinatorial Computing*, ACM Press, 1994].
- Also in this period, he brought out the eight or so volumes of his collected papers, some of which could be a good text for a seminar in some of the topics in *TAOCP* as originally conceived

which he is now never going to get to. He brought out the most recent of these volumes in late 2010.

- Of course, he also developed the concept of literate programming and the software to compile and document large systems such as  $\text{\TeX}$  [*The CWEB System of Structured Documentation*, with Silvio Levy, Addison-Wesley, 1993].

(Knuth also did some research not related to computer science: for instance, a carefully created book relating to Bible study (see <http://www.tug.org/TUGboat/tb12-2/tb32reviews.pdf>); but who is to say that that Bible study was not also a useful preparatory step toward Volume 4 of *TAOCP*.)

Apparently finally Knuth was ready again to tackle Volume 4 which, after resigning as a Stanford professor to have more time, he has been doing for a number of years now (e.g., pushing out the fascicles). Despite all Knuth’s contributions in a variety of areas, he still felt the need to finish what he calls “the interesting parts” of his original plan for *TAOCP*. The preface to Volume 4A now suggests (to me) that he may never get beyond vol 4B, 4C, ... (I don’t know if he intended to say this — he does say he will surely never get to 4Z.)

My admiration for Knuth is unbounded. Volume 4A is another stunning example of Knuth’s breadth, thoroughness, and desire to produce a beautiful book.

As a purely personal matter, I adopted the use of  $\text{\TeX}$ -based systems when I made the decision in the 1990s to stop using word processing systems with graphical user interfaces. I chose  $\text{\TeX}$  as my replacement word processing system because of my admiration for Knuth and the desire to use something he had created. I haven’t been disappointed.

More generally, I stand by my comparison with Picasso. Knuth is as much an artist as he is a technician. He goes where his muse takes him and he does so with unmatched (for the computer field) skill, care, depth, breadth, and artistry. We in the  $\text{\TeX}$  community remain major benefactors of Knuth’s skill and artistry.

For someone like me who still feels a strong connection to the world of computer programming, there is another thing to marvel at regarding Knuth. He is perhaps unique as a person of his stature as a theoretician for how many lines of code he apparently still writes every day. For Knuth programming is an art he practices every day.

◊ David Walden  
<http://www.walden-family.com/texland>



## The Treasure Chest

This is a list of selected new packages posted to CTAN (<http://ctan.org>) from April–July 2011, with descriptions based on the announcements and edited for brevity.

Entries are listed alphabetically within CTAN directories. A few entries which the editors subjectively believed to be of especially wide interest or otherwise notable are starred; of course, this is not intended to slight the other contributions.

We hope this column and its companions will help to make CTAN a more accessible resource to the T<sub>E</sub>X community. Comments are welcome, as always.

◇ Karl Berry  
<http://tug.org/ctan.html>

---

### biblio

**vak** in `biblio/bibtex/contrib`  
 Bibliography support for Russian standards.

---

### fonts

**boondox** in `fonts`  
 Math alphabets derived from the STIX fonts.

**cantarell** in `fonts`  
 A contemporary humanist sans serif.

**dutchcal** in `fonts`  
 Reworking of the calligraphic math font ESSTIX13, adding bold.

**eststix** in `fonts`  
 Precursor to STIX with some novel features.

**fdsymbol** in `fonts`  
 Math symbol font designed as a companion to Fedra.

\***mathalfa** in `fonts`  
 Support for loading many math script, blackboard bold, etc., alphabets.

---

### graphics

**grafcet** in `graphics/pgf/contrib`  
 Sequential functional chart in PGF.

**pst-layout** in `graphics/pstricks/contrib`  
 Typesetting quasi-tabular material such as menu pages, music programs, business cards, etc.

**pst-rubans2** in `graphics/pstricks/contrib`  
 Drawing three-dimensional tapes.

---

### info

**latex4wp-it** in `info`  
 Italian translation of `latex4wp`.

**lshort-czech** in `info`  
 New Czech translation of `lshort`.

\***macros2e** in `info`  
 Unofficial list of internal macros to L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> which can be useful to package authors.

---

### language

**serbian-apostrophe** in `serbian`  
 Typeset Serbian words with an apostrophe.

---

### macros/generic

**upca** in `macros/generic`  
 Printing UPC-A barcodes.

---

### macros/latex/contrib

**cascadilla** in `macros/latex/contrib`  
 Typesetting conforming to the style sheet of the Cascadilla Proceedings Project.

**chemmacros** in `macros/latex/contrib`  
 Convenient typesetting of chemistry documents.

**chemnum** in `macros/latex/contrib`  
 Numbering of chemical compounds.

**chet** in `macros/latex/contrib`  
 Make T<sub>E</sub>Xing faster, inspired by `harvmac`.

**cmpj** in `macros/latex/contrib`  
 Support for the *Condensed Matter Physics* journal.

**collectbox** in `macros/latex/contrib`  
 Robustly process an argument as a horizontal box.

**decorule** in `macros/latex/contrib`  
 Decorative swelled rule using a standard font symbol.

**ifnextok** in `macros/latex/contrib`  
 Customizing `\ifnextchar`'s space skipping; handling `\` followed by printed brackets.

**jvlisting** in `macros/latex/contrib`  
 Allow indentation and customization of verbatim environments.

\***l3experimental** in `macros/latex/contrib`  
 Code intended mainly for testing by interested users. Over time, this material may move to one of the following two divisions.

\***l3kernel** in `macros/latex/contrib`  
 Code which will be present in a L<sup>A</sup>T<sub>E</sub>X 3 kernel in its current form; replaces `expl3`.

\***l3packages** in `macros/latex/contrib`  
 Code which is broadly 'stable', and which adds new functionality to L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>. This material may be present in a L<sup>A</sup>T<sub>E</sub>X 3 kernel, but the interfaces may change.

`macros/latex/contrib/l3packages`

**ltxkeys** in `macros/latex/contrib`  
 Defining and managing keyword-value interfaces.

**moreenum** in `macros/latex/contrib`  
 Greek, hex, and other enumeration styles.

**morewrites** in `macros/latex/contrib`  
 Hooks on `\immediate`, `\openout`, `\write`, `\closeout` to avoid limiting the number of write streams.

**nuc** in `macros/latex/contrib`  
 Automatic notation for nuclear isotopes.

**othelloboard** in `macros/latex/contrib`  
 Creating annotated Othello board diagrams.

**regstats** in `macros/latex/contrib`  
 Usage of  $\TeX$  registers (count, dimen, etc.).

**sapthesis** in `macros/latex/contrib`  
 Style for theses at Sapienza — University of Rome.

**schwalbe-chess** in `macros/latex/contrib`  
 Typesetting the German chess problem magazine *Die Schwalbe*.

**serbian-lig** in `macros/latex/contrib`  
 Explicit list of Serbian words with ligatures disabled.

**sitem** in `macros/latex/contrib`  
 Save optional arguments to `\item` in a box.

**srbook-mem** in `macros/latex/contrib`  
 Serbian section numbering for memoir.

**temlines** in `macros/latex/contrib`  
 Emulate the old `\emlines` from  $\TeX$ cad and `em $\TeX$` .

**thumbs** in `macros/latex/contrib`  
 Creating customizable thumb indexes.

**uafthesis** in `macros/latex/contrib`  
 The Official Unofficial document class for theses at the University of Alaska Fairbanks.

**unamthesis** in `macros/latex/contrib`  
 Style for Universidad Nacional Autónoma de México theses, both graduate and undergraduate.

**xhfill** in `macros/latex/contrib`  
 Modifying width and color of `\hrulefill`.

---

### **macros/latex/contrib/babel-contrib**

**serbianc** in `m/l/c/babel-contrib`  
 Serbian Cyrillic support for Babel.

**serbian-date** in `m/l/c/babel-contrib`  
 Replace `\date` for Serbian (Latin).

---

### **macros/latex/contrib/beamer-contrib**

**beamersubframe** in `m/l/c/beamer-contrib`  
 Embed frames with details without reordering source.

---

### **macros/latex/contrib/biblatex-contrib**

**biblatex-juradiss** in `m/l/c/biblatex-contrib`  
`biblatex` support for German law theses.

**uni-wtal-ger** in `m/l/c/biblatex-contrib`  
`biblatex` support for literary studies at Bergische Universität Wuppertal.

---

### **macros/luatex**

**\*interpreter** in `macros/luatex/generic`  
 Preprocess input files without an external program.

**luabibentry** in `macros/luatex/latex`  
 Repeating bibliographic entries in the document, like `bibentry` for  $\LaTeX$ .

**lualatex-math** in `macros/luatex/latex`  
 Fixes for math-related Lua $\LaTeX$  issues.

---

### **macros/plain**

**getoptk** in `macros/plain/contrib`  
 Defining new macros with the keyword-based `\hrule` interface style.

---

### **macros/xetex**

**fontbook** in `macros/xetex/latex`  
 Generate a font book.

**kannada** in `macros/xetex/latex/polyglossia-contrib`  
 Kannada support for Polyglossia.

---

### **support**

**ant-worker-tasks** in `support`  
 Apache Ant tasks for document creation, PDF manipulation, etc.

---

### **systems**

**lualatex-platform** in `systems/luatex/contrib`  
 Extension module for platform-specific code.

**mactex** in `systems/mac`  
 Mac $\TeX$  2011.

**protext** in `systems/win32`  
 Pro $\TeX$ t 3.0 for 2011.

**texlive** in `systems`  
 $\TeX$  Live 2011.

---

*Les Cahiers GUTenberg*  
Contents of issue 54–55 (2010)

*Les Cahiers GUTenberg* is the journal of GUT, the French-language T<sub>E</sub>X user group (<http://www.gutenberg.eu.org>).

PAUL ISAMBERT, LuaT<sub>E</sub>X: vue d'ensemble [LuaT<sub>E</sub>X: An overview]; pp. 3–12

MANUEL PÉGOURIÉ-GONNARD, Un guide pour LuaL<sup>A</sup>T<sub>E</sub>X [A guide to LuaL<sup>A</sup>T<sub>E</sub>X]; pp. 13–35

This document is a map, or tourist guide, for the new world of LuaL<sup>A</sup>T<sub>E</sub>X. The intended audience ranges from complete newcomers (with a working knowledge of conventional L<sup>A</sup>T<sub>E</sub>X) to package developers. This guide is intended to be comprehensive in the following sense: it contains pointers to all relevant sources, gathers information that is otherwise scattered, and adds introductory material.

MAXIME CHUPIN, LuaL<sup>A</sup>T<sub>E</sub>X pour les non-sorciers, deux exemples [LuaL<sup>A</sup>T<sub>E</sub>X for non-wizards, two examples]; pp. 37–56

This article presents a way to use LuaT<sub>E</sub>X without being an expert in T<sub>E</sub>X or Lua. The examples illustrate the treatment of external files by Lua, and the use of Lua in order to perform some computations hardly implementable in T<sub>E</sub>X. These examples are the generation of L<sup>A</sup>T<sub>E</sub>X tabular code from an external data file and the implementation of the method of least squares and its graphical presentation.

MANUEL PÉGOURIÉ-GONNARD, Attributs et couleurs [Attributes and colors]; pp. 57–85

This article presents a new tool provided by LuaT<sub>E</sub>X to extend T<sub>E</sub>X: attributes, and how they can be used to implement colors. First, we study the general concept of attributes and the T<sub>E</sub>X and Lua interfaces. Then, we recall the main points of the classical color implementation in L<sup>A</sup>T<sub>E</sub>X and its well-known limitations. Finally, a solution to these problems, using attributes, is presented, and demonstrates a few general principles in the use of attributes, which are obviously not limited to colors.

PAUL ISAMBERT, Ponctuation française avec LuaT<sub>E</sub>X [French punctuation with LuaT<sub>E</sub>X]; pp. 87–100

If T<sub>E</sub>X had been created by a French man, maybe it would have a primitive dedicated to inserting spaces before some punctuation signs (question mark, exclamation mark, colon, semi-colon) as is usual in the French typographical tradition — but this wasn't the case. LuaT<sub>E</sub>X is not written by a French team either, but it enables handling character lists while texts are being typeset. The goal of

this work is to illustrate its power by presenting Lua algorithms meant to insert the proper space before those symbols that require it.

TACO HOEKWATER, LuaT<sub>E</sub>X 0.65 et les mathématiques [LuaT<sub>E</sub>X 0.65 and mathematics]; pp. 101–127

The math machinery in LuaT<sub>E</sub>X has been completely overhauled since version 0.40. The handling of mathematics in LuaT<sub>E</sub>X has been extended quite a bit compared to how T<sub>E</sub>X82 (and therefore pdfT<sub>E</sub>X) handles math. First, LuaT<sub>E</sub>X adds primitives and extends some others so that Unicode input can be used easily. Second, all of T<sub>E</sub>X82's internal special values (for example for operator spacing) have been made accessible and changeable via control sequences. Third, there are extensions that make it easier to use OpenType math fonts. And finally, there are some extensions that have been proposed in the past that are now added to the engine.

This article is an update of the original article that was published in *MAPS* 38, documenting the changes in LuaT<sub>E</sub>X between version 0.40 and version 0.65.

THIERRY BOUCHE, Colophon; pp. 128–130

---

*Die T<sub>E</sub>Xnische Komödie 2/2011*

*Die T<sub>E</sub>Xnische Komödie* is the journal of DANTE e.V., the German-language T<sub>E</sub>X user group (<http://www.dante.de>). [Editorial items are omitted.]

NORMAN WATTENBERG, DANTE 2011 in Bremen; pp. 25–27

After last year's autumn meeting in Trier this year's spring meeting took place in Bremen. The L<sup>A</sup>T<sub>E</sub>X community gathered in the old Hanseatic city to discuss the latest developments in the world of L<sup>A</sup>T<sub>E</sub>X and to exchange ideas.

CHRISTINE RÖMER, Gewichten Wichtiges und Unwichtiges mit L<sup>A</sup>T<sub>E</sub>X markieren [Documenting packages in German]; pp. 28–35

This article explains why it is reasonable to write packages in German or to translate package documentation into German. Furthermore the article describes how to prepare and typeset package documentation.

MARKUS KOHM, ALEXANDER WILLAND, Alles in einem — Texte und Tabellen mit LuaL<sup>A</sup>T<sub>E</sub>X [All in one — Texts and tables with LuaL<sup>A</sup>T<sub>E</sub>X]; pp. 36–47

Using an invoice this article shows how to enter values in a table using Lua, calculate various values

(e.g. VAT) and to typeset the results in a  $\LaTeX$  table. The advantage of this solution is an increased transparency of the calculation and the absence of error-prone imports from spreadsheet software.

WILFRIED EHRENFELD, Die Dokumentenklasse `iwhdp` [The `iwhdp` document class]; pp. 48–54

This article describes the `iwhdp` document class of the Halle Institute for Economic Research which is used for German and English discussion papers. Little more than a year has passed since the DTK article introducing the first version of this class. The ideas mentioned there have been implemented and many more. Finally the document class has matured so that it could be published on CTAN.

BOGUSŁAW JACKOWSKI and PIOTR STRZELCZYK, How to make more than one OTF math font?; pp. 55–56

Since 2007, when Microsoft released their math-equipped Word and the relevant font Cambria Math, the world has seen only two more math fonts which can be used with MS Word/ $X_{\text{}}\TeX$ / $\text{Lua}\TeX$ : Asana by Apostolos Syropoulos and Khaled Hosny's XITS. Another one is upcoming: Latin Modern Math. Our real aim is, however, not to provide just one new math font, but to create several of them, almost at once. We will present the pre-release of the Latin Modern Math font and, at the same time, explain how we intend to use our and others' experience to facilitate the creation (painful in any case) of math OTFs. Moreover, we will discuss the relationship between the Latin Modern and  $\TeX$  Gyre math projects.

[Received from Herbert Voß.]

---

### Zpravodaj 20(4), 2010

Editor's note: *Zpravodaj* is the journal of  $\zeta\text{TUG}$ , the  $\TeX$  user group oriented mainly but not entirely to the Czech and Slovak languages (<http://www.cstug.cz>).

JAROMÍR KUBEN, Úvodník [Opening letter from the  $\zeta\text{TUG}$  president], p. 265.

ROBERT MAŘÍK, ROMAN PLCH, PETRA ŠARMANOVÁ, Tvorba interaktivních testů pomocí systému  $\text{Acro}\TeX$  [Publishing interactive tests using  $\text{Acro}\TeX$ ], pp. 266–291.

In this paper we describe the preparation of interactive tests in PDF with  $\text{Acro}\TeX$  eDucation Bundle, <http://acrotex.net/>.

PAVEL STRÍŽ, Představení formátu  $X_{\text{}}\LaTeX$  [Introduction to  $X_{\text{}}\LaTeX$ ], pp. 292–296.

The article warmly welcomes us to the world of  $X_{\text{}}\TeX$  (<http://www.tug.org/xetex/>), especially to the  $X_{\text{}}\LaTeX$  format commonly used these days along with  $\text{Lua}\LaTeX$ .  $X_{\text{}}\TeX$  supports processing UTF-8 coded documents and direct TTF and OTF font loading and use. The author presents several basic examples accompanied by output previews.

PETER WILSON, Mohlo by to fungovat. I – Porovnávání řetězců [This is a translation of the column *Glisterings* which appeared in *TUGboat*, 22:4, 2001. Translation to Czech by Jan Šustek.], pp. 297–301.

JIRÍ RYBIČKA, OSSConf 2010 v Žilině [Impressions from the OSSConf 2010 conference], p. 302.

VLASTIMIL OTT, Shrnutí konference Otvorený softvér vo vzdelávaní, výskume a v IT riešeniach 2010 [Impressions from the OSSConf 2010 conference in Žilina, Slovakia], pp. 303–307.

JAROSLAV HAJTMAR, Postřehy z  $\TeX$ Xové dvojkonference [Impressions from  $\TeX$  conferences (4th International Con $\TeX$ t meeting and 3rd  $\TeX$ perience conference) in Břežlov, The Czech Republic], pp. 308–314.

PAVLÍNA HABROVANSKÁ, Zpráva z konference  $\TeX$ perience 2010 [Impressions from  $\TeX$ perience 2010], pp. 315–323.

Pozvánky na akce [Conference invitations], p. 324.

Pozvánka na OSSConf 2011 [Invitation to the OSSConf 2011 conference], p. 325.

Pozvánka na  $\TeX$ perience 2011 [Invitation to the  $\TeX$ perience 2011 conference], pp. 326–328.

Zpráva o činnosti  $\zeta\text{TUG}$ u za rok 2010 [Activities of  $\zeta\text{TUG}$  in 2010], pp. 329–330.

Zápis z Valné hromady  $\zeta\text{TUG}$ u – Brno 11. 12. 2010 [Report from  $\zeta\text{TUG}$  annual meeting in Brno], p. 331.

Plán práce  $\zeta\text{TUG}$ u na rok 2011 (a dál) [ $\zeta\text{TUG}$  plans for 2011 and on], pp. 331–332.

[Received from Pavel Stríž.]

## T<sub>E</sub>X Consultants

The information here comes from the consultants themselves. We do not include information we know to be false, but we cannot check out any of the information; we are transmitting it to you as it was given to us and do not promise it is correct. Also, this is not an official endorsement of the people listed here. We provide this list to enable you to contact service providers and decide for yourself whether to hire one.

TUG also provides an online list of consultants at <http://tug.org/consultants.html>. If you'd like to be listed, please see that web page.

### **Aicart Martinez, Mercè**

Tarragona 102 4<sup>o</sup> 2<sup>a</sup>  
08015 Barcelona, Spain  
+34 932267827  
Email: [m.aicart \(at\) ono.com](mailto:m.aicart@ono.com)  
Web: <http://www.edilatex.com>

We provide, at reasonable low cost, L<sup>A</sup>T<sub>E</sub>X or T<sub>E</sub>X 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](mailto:typesetting@dangerouscurve.org)  
Web: <http://dangerouscurve.org/tex.html>

We are your macro specialists for T<sub>E</sub>X or L<sup>A</sup>T<sub>E</sub>X fine typography specs beyond those of the average L<sup>A</sup>T<sub>E</sub>X macro package. If you use X<sub>Y</sub>L<sup>A</sup>T<sub>E</sub>X, we are your microtypography specialists. We take special care to typeset mathematics well.

Not that picky? We also handle most of your typical T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X typesetting needs.

We have been typesetting in the commercial and academic worlds since 1979.

Our team includes Masters-level computer scientists, journeyman typographers, graphic designers, letterform/font designers, artists, and a co-author of a T<sub>E</sub>X book.

### **Hendrickson, Amy**

Brookline, MA, USA  
Email: [amyh \(at\) texnology.com](mailto:amyh@texnology.com)  
Web: <http://www.texnology.com>

L<sup>A</sup>T<sub>E</sub>X macro writing our speciality for more than 25 years: macro packages for major publishing companies, author support; journal macros for American Geophysical Union, Proceedings of the National Academy of Sciences, and many more.

### **Hendrickson, Amy (cont'd)**

Scientific journal design/production/hosting, e-publishing in PDF or HTML.

L<sup>A</sup>T<sub>E</sub>X training, at MIT, Harvard, many more venues. Customized on site training available.

Please visit our site for samples, and get in touch. We are particularly glad to take on adventurous new uses for L<sup>A</sup>T<sub>E</sub>X, for instance, web based report generation including graphics, for bioinformatics or other applications.

### **Latchman, David**

4113 Planz Road Apt. C  
Bakersfield, CA 93309-5935  
+1 518-951-8786  
Email: [david.latchman \(at\) gmail.com](mailto:david.latchman@gmail.com)  
Web: <http://www.elance.com/s/dlatchman>

Proficient and experienced L<sup>A</sup>T<sub>E</sub>X typesetter for books, monographs, journals and papers allowing your documents and books to look their possible best especially with regards to technical documents. Graphics/data rendered either using TikZ or Gnuplot. Portfolio available on request.

### **Peter, Steve**

295 N Bridge St.  
Somerville, NJ 08876  
+1 732 306-6309  
Email: [speter \(at\) mac.com](mailto:speter@mac.com)

Specializing in foreign language, multilingual, linguistic, and technical typesetting using most flavors of T<sub>E</sub>X, 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 T<sub>E</sub>X-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](mailto:rshanmugam92@yahoo.com)

As a Consultant, I provide consultation, training, and full service support to individuals, authors, typesetters, publishers, organizations, institutions, etc. I support leading BPO/KPO/ITES/Publishing companies in implementing latest technologies with high level automation in the field of Typesetting/Prepress, ePublishing, XML2PAGE, WEBTechnology, DataConversion, Digitization, Cross-media publishing, etc., with highly competitive prices. I provide consultation in building business models &

**Shanmugan, R.** (cont'd)

technology to develop your customer base and community, streamlining processes in getting ROI on our workflow, New business opportunities through improved workflow, Developing eMarketing/E-Business Strategy, etc. I have been in the field BPO/KPO/ITES, Typesetting, and ePublishing for 16 years, handled various projects. I am a software consultant with Master's Degree. I have sound knowledge in T<sub>E</sub>X, L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>, XMLT<sub>E</sub>X, Quark, InDesign, XML, MathML, DTD, XSLT, XSL-FO, Schema, ebooks, OeB, etc.

**Sievers, Martin**

Im Treff 8, 54296 Trier, Germany

+49 651 4936567-0

Email: [info \(at\) schoenerpublizieren.com](mailto:info@ schoenerpublizieren.com)

Web: <http://www.schoenerpublizieren.com>

As a mathematician with ten years of typesetting experience I offer T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X 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 (BIBT<sub>E</sub>X, bibl<sub>at</sub>ex) to typesetting your math, tables or graphics — just contact me with information on your project.

**Veytsman, Boris**

46871 Antioch Pl.

Sterling, VA 20164

+1 703 915-2406

Email: [borisv \(at\) lk.net](mailto:borisv@lk.net)

Web: <http://www.borisv.lk.net>

T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X consulting, training and seminars. Integration with databases, automated document preparation, custom L<sup>A</sup>T<sub>E</sub>X packages, conversions and much more. I have about sixteen years of experience in T<sub>E</sub>X and twenty-nine years of experience in teaching & training. I have authored several packages on CTAN, published papers in T<sub>E</sub>X related journals, and conducted several workshops on T<sub>E</sub>X and related subjects.

**Young, Lee A.**

127 Kingfisher Lane

Mills River, NC 28759

+1 828 435-0525

Email: [leeayoung \(at\) live.com](mailto:leeayoung@live.com)

Web: <http://www.latexcopyeditor.net>

<http://www.editingscience.net>

Copyediting your .tex manuscript for readability and mathematical style by a Harvard Ph.D. Experience: edited hundreds of ESL journal articles, economics and physics textbooks, scholarly monographs, L<sup>A</sup>T<sub>E</sub>X manuscripts for the Physical Review; career as professional, published physicist.

## Letters

**Status of the American core CTAN node**

Jim Hefferon

Users can download materials from many CTAN mirrors<sup>1</sup> but installation or updating of those materials happens at only a few sites. Recently I withdrew from running one of these, leaving core sites in Germany and Great Britain.

In my AZ project I developed a set of keyword and hierarchical characterizations of the packages that CTAN holds.<sup>2</sup> I now run a personal site that offers a web view of CTAN enhanced — I hope — by these extra characterizations (and by a full-text search of the package descriptions). Because I am no longer doing uploads I have suppressed the functionality that allows authors to edit the package

description and characterizations as part of their upload process. I have kept links for people to suggest better characterizations.

My host's canonical address is <http://alan.smcvt.edu>; the bandwidth is generously sustained by Saint Michael's College in Colchester, Vermont. If you like the site and want to bookmark it, I suggest the address <http://tug.ctan.org>, which as of this writing points to my site.

Note that you can use these characterizations without making a special trip to my site since T<sub>E</sub>X Live's tlmgr will search them.<sup>3</sup>

During my more than decade-long time with CTAN I met and worked with many wonderful and helpful people. My thanks to them all!

◇ Jim Hefferon  
Saint Michael's College  
Colchester, Vermont USA  
[ftpmain \(at\) alan dot smcvt dot edu](mailto:ftpmain@alan.smcvt.edu)

<sup>1</sup> See <http://ctan.org/mirrors>.

<sup>2</sup> "CTAN packages get keywords", *TUGboat* 31:2, 2010.

<sup>3</sup> A description is at <http://tug.org/texlive/doc/tlmgr.html#taxonomies>.

# Calendar

**2011**

- Aug 7–11 SIGGRAPH 2011, Vancouver, Canada.  
[www.siggraph.org/s2011](http://www.siggraph.org/s2011)
- Aug 26 L<sup>A</sup>T<sub>E</sub>X for Beginners course, UK TUG,  
University of East Anglia, London,  
England. [uk.tug.org](http://uk.tug.org)
- Sep 11–14 IStype: Istanbul Typography Seminars,  
Istanbul, Turkey. [istype.com](http://istype.com)
- Sep 14–18 Association Typographique Internationale  
(ATypI) annual conference, Theme: the  
letterform “eth”,  
Reykjavik, Iceland. [www.atypi.org](http://www.atypi.org)
- Sep 18–23 XML Summer School, St Edmund  
Hall, Oxford University, Oxford, UK.  
[www.xmlsummerschool.com](http://www.xmlsummerschool.com)
- Sep 19–22 ACM Symposium on Document  
Engineering, Mountain View, California.  
[www.documentengineering.org](http://www.documentengineering.org)
- Sep 19–24 The fifth ConT<sub>E</sub>Xt user meeting,  
Bassenge-Boirs, Belgium.  
[meeting.contextgarden.net/2011](http://meeting.contextgarden.net/2011)
- Sep 28 –  
Oct 2 T<sub>E</sub>Xperience 2011 (4<sup>th</sup> T<sub>E</sub>Xperience  
Conference, organized by C<sub>S</sub>TUG and the  
Faculty of Management and Economics,  
Tomas Bata University in Zlín),  
Železná Ruda, The Czech Republic.  
[striz9.fame.utb.cz/texperience](http://striz9.fame.utb.cz/texperience).
- Sep 30 –  
Oct 2 DANTE Herbsttagung and 45<sup>th</sup> meeting,  
Garmisch-Partenkirchen, Germany.  
[www.dante.de/events/mv45.html](http://www.dante.de/events/mv45.html)
- Oct 8 NTG 48<sup>th</sup> meeting, Groningen, Netherlands.  
[www.ntg.nl/bijeen/bijeen48.html](http://www.ntg.nl/bijeen/bijeen48.html)
- Oct 14–15 American Printing History Association’s  
36<sup>th</sup> annual conference, “Printing at  
the Edge”, University of California,  
San Diego, California,  
[www.printinghistory.org/about/  
calendar.php](http://www.printinghistory.org/about/calendar.php)

- Oct 14–16 The Ninth International Conference  
on the Book, University of Toronto,  
Ontario, Canada.  
[booksandpublishing.com/conference-2011](http://booksandpublishing.com/conference-2011)

**TUG 2011****Trivandrum, India.**

- Oct 19–21 The 32<sup>nd</sup> annual meeting  
of the T<sub>E</sub>X Users Group.  
T<sub>E</sub>X in the eBook era. [tug.org/tug2011](http://tug.org/tug2011)
- 
- Oct 20–22 TYPOLondon 2011, “Places”, University  
of London, UK. [www.typolondon.com](http://www.typolondon.com)
- Nov 10–11 Tenth annual St Bride Library  
Conference, “Critical Tensions”,  
London, England. [stbride.org/events](http://stbride.org/events)

**2012**

- Jan 27 “The Design of Understanding”,  
St. Bride Library, London, England.  
[stbride.org/events](http://stbride.org/events)
- Jun 26–29 SHARP 2012, “The Battle for Books”,  
Society for the History of Authorship,  
Reading & Publishing. Dublin, Ireland.  
[www.sharpweb.org](http://www.sharpweb.org)
- Jul 16–22 Digital Humanities 2012, Alliance of  
Digital Humanities Organizations,  
University of Hamburg, Germany.  
[www.digitalhumanities.org/conference](http://www.digitalhumanities.org/conference)
- Aug 5–9 SIGGRAPH 2012, Los Angeles, California.
- Oct 8–12 EuroT<sub>E</sub>X 2012 and the sixth ConT<sub>E</sub>Xt  
user meeting, Breskens, The Netherlands.  
[meeting.contextgarden.net/2012](http://meeting.contextgarden.net/2012)

*Status as of 25 July 2011*

For additional information on TUG-sponsored events listed here, contact the TUG office (+1 503 223-9994, fax: +1 206 203-3960, e-mail: [office@tug.org](mailto:office@tug.org)). For events sponsored by other organizations, please use the contact address provided.

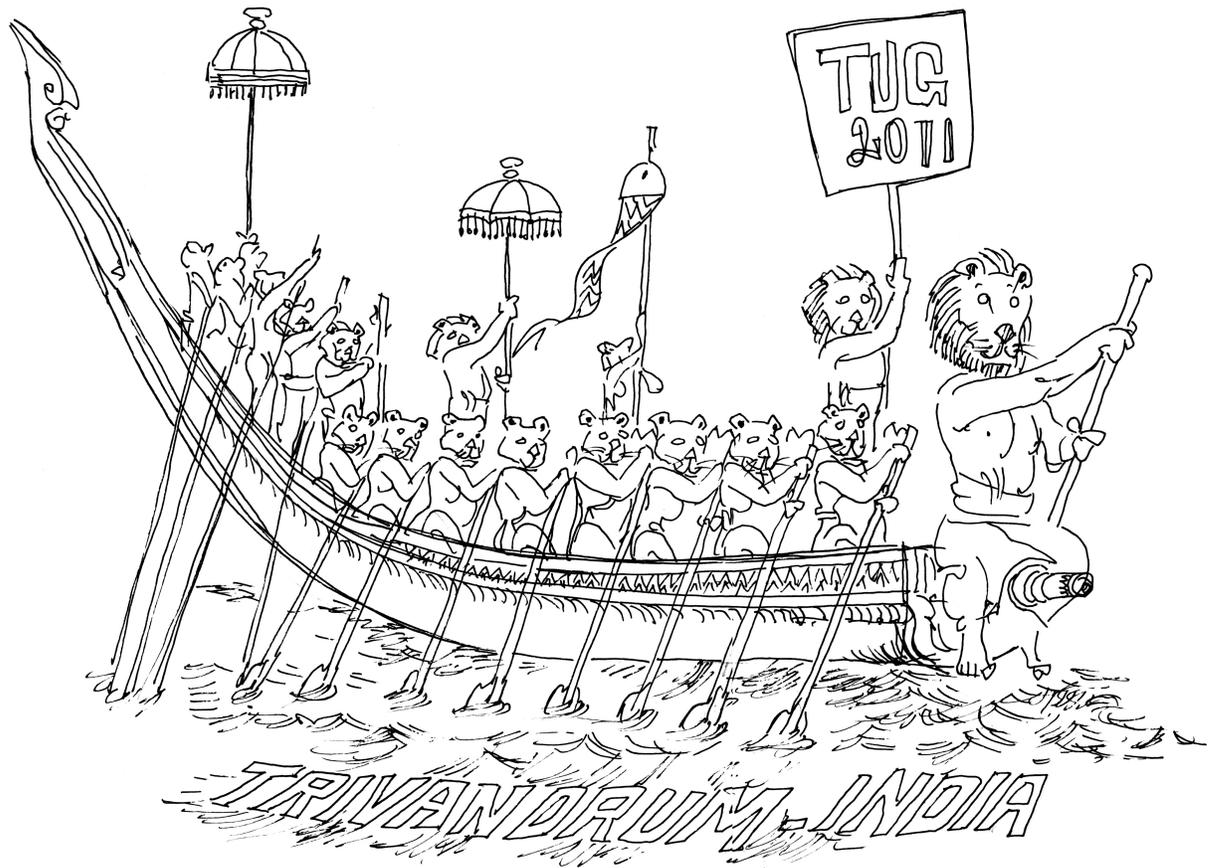
A combined calendar for all user groups is online at [texcalendar.dante.de](http://texcalendar.dante.de).

Other calendars of typographic interest are linked from [tug.org/calendar.html](http://tug.org/calendar.html).

# TUG 2011: T<sub>E</sub>X in the eBook era

Presentations covering the T<sub>E</sub>X world  
The 32<sup>nd</sup> Annual Meeting of the T<sub>E</sub>X Users Group

<http://tug.org/tug2011> ■ [tug2011@tug.org](mailto:tug2011@tug.org)



**October 19–21, 2011**

**River Valley Technologies  
Trivandrum, Kerala  
India**

- September 1, 2011 — preprints deadline
- October 19–21, 2011 — conference and workshop
- October 31, 2011 — deadline for final papers

*Sponsored by the T<sub>E</sub>X Users Group, DANTE e.V.,  
and River Valley Technologies.*

## Introductory

- 225 *William Adams* / Book review: *A Specimen Portfolio of Wood Type in the Cary Collection*
  - review of collection of wood type specimens (RIT Press)
- 131 *Barbara Beeton* / Editorial comments
  - typography and TUGboat news
- 131 *Karl Berry* / From the President
  - software; conferences; interviews
- 238 *Jim Hefferon* / Status of the American core CTAN node
  - web view of CTAN enhanced with extra characterizations
- 145 *Krzysztof Pszczola* / Teaching L<sup>A</sup>T<sub>E</sub>X to the students of mathematics
  - experiences with and approaches to teaching L<sup>A</sup>T<sub>E</sub>X
- 132 *T<sub>E</sub>X Collection editors* / T<sub>E</sub>X Collection 2011 DVD
  - very high-level overview of the 2011 software releases
- 226 *Boris Veytsman* / Book review: *The Art of the Book in the Twentieth Century*
  - review of this new book by Jerry Kelly (RIT Press)
- 228 *Boris Veytsman* / Book review: *L<sup>A</sup>T<sub>E</sub>X Beginner's Guide*
  - review of this new book by Stefan Kottwitz (Packt Publishers)
- 230 *David Walden* / An appreciation: *The Art of Computer Programming, Volume 4A*
  - discussion of Knuth's magnum opus (Addison-Wesley)

## Intermediate

- 233 *Karl Berry* / The treasure chest
  - new CTAN packages, April–July 2011
- 158 *William Cheswick* / iT<sub>E</sub>X — Document formatting in an ereader world
  - producing bundles for practical iPad reading
- 152 *Hans Hagen* / E-books: Old wine in new bottles
  - reflections on using and producing ebooks, especially with ConT<sub>E</sub>Xt
- 133 *Stefan Löffler* / T<sub>E</sub>Xworks — As you like it
  - new scripting and other features in T<sub>E</sub>Xworks 0.4
- 211 *Aditya Mahajan* / ConT<sub>E</sub>Xt basics for users: Paper setup
  - predefined and custom page and print sizes in ConT<sub>E</sub>Xt
- 206 *Luca Merciadri* / Merciadri packages: An overview
  - `bigints`; `dashundergaps`; `plantslabels`; matrices with borders
- 164 *Michael Sharpe* / Math alphabets and the `mathalfa` package
  - survey of and package for available math script, double-struck, and fraktur fonts
- 217 *Paul Shaw* / Sixty years of book design at St. Gallen, Switzerland
  - discussion of this design exhibition, with many examples
- 169 *Ulrik Vieth* and *Mojca Miklavc* / Another incarnation of Lucida: Towards Lucida OpenType
  - review of font technologies and a new implementation of Lucida
- 213 *David Walden* / Experiences with notes, references, and bibliographies
  - a variety of practical approaches to bibliographies, with many examples
- 202 *Peter Wilson* / Glisterings
  - ornaments with the Web-O-Mints font

## Intermediate Plus

- 185 *Michael Le Barbier Grünwald* / Macro interfaces and the `getoptk` package
  - survey of macro interfaces; implementing a new keyword interface for plain T<sub>E</sub>X
- 146 *Paul Isambert* / Drawing tables: Graphic fun with LuaT<sub>E</sub>X
  - using Lua for drawing, with PDF output
- 193 *Oleg Parashchenko* / The `calls` package: Multipage tables with decorations
  - an advanced table package
- 139 *Herbert Voß* / Reading and executing source code
  - typesetting and executing L<sup>A</sup>T<sub>E</sub>X and other source

## Advanced

- 136 *Taco Hoekwater* / MetaPost 1.750: Numerical engines
  - supporting back-end numerical libraries, MPFR and decNumber
- 177 *Luigi Scarso* / MFLua
  - integrating Lua into METAFONT for post-processing glyphs

## Contents of other T<sub>E</sub>X journals

- 235 *Les Cahiers GUTenberg*: Issue 54–55 (2010); *Die T<sub>E</sub>Xnische Komödie*: Issue 2/2011; *Zpravodaj*: Issue 20(4) (2010)

## Reports and notices

- 224 Institutional members
- 237 T<sub>E</sub>X consulting and production services
- 239 Calendar
- 240 TUG 2011 announcement