

# TUGBOAT

Volume 35, Number 2 / 2014

TUG 2014 Conference Proceedings

<b>TUG 2014</b>	126	Conference sponsors, participants, program, and photos
	130	David Latchman / <i>TUG 2014 in Portland</i>
	134	Tracy Kidder / <i>Visiting TUG 2014</i>
<b>Fonts</b>	135	Michael Sharpe / <i>Recent additions to T<sub>E</sub>X's font repertoire</i>
<b>Publishing</b>	139	Jim Hefferon and Lon Mitchell / <i>Experiences converting from PDF-only to paper</i>
	142	Joseph Hogg / <i>Texinfo visits a garden</i>
<b>Software &amp; Tools</b>	145	Richard Koch / <i>MacT<sub>E</sub>X design philosophy vs. TeXShop design philosophy</i>
	152	Adam Maxwell / <i>T<sub>E</sub>X Live Utility: A slightly-shiny Mac interface for T<sub>E</sub>X Live Manager (tlmgr)</i>
	157	Doug McKenna / <i>On tracing the trip test with JSBox</i>
	168	Julian Gilbey / <i>Creating (mathematical) jigsaw puzzles using T<sub>E</sub>X and friends</i>
	173	Pavneet Arora / <i>SUTRA — A workflow for documenting signals</i>
<b>Graphics</b>	192	David Allen / <i>Dynamic documents</i>
	179	Andrew Mertz, William Slough and Nancy Van Cleave / <i>Typesetting figures for computer science</i>
<b>Typography</b>	195	Leyla Akhmadeeva and Boris Veytsman / <i>Typography and readability: An experiment with post-stroke patients</i>
<b>Macros</b>	198	SK Venkatesan and CV Rajagopal / <i>T<sub>E</sub>X and copyediting</i>
	202	Boris Veytsman / <i>An output routine for an illustrated book: Making the FAO Statistical Yearbook</i>
<b>Electronic Documents</b>	205	Keiichiro Shikano / <i>xml2tex: An easy way to define XML-to-L<sup>A</sup>T<sub>E</sub>X converters</i>
	209	Robert A. Beezer / <i>MathBook XML</i>
	212	William Hammond / <i>Can L<sup>A</sup>T<sub>E</sub>X profiles be rendered adequately with static CSS?</i>
<b>Abstracts</b>	219	TUG 2014 abstracts (Bazargan, Berry, Crossland, Cuning, de Souza, Doob, Farmer, McKenna, Mittelbach, Moore, Raies, Robertson, Tétreault, Wetmore)
	222	S Parthasarathy / <i>Let's Learn L<sup>A</sup>T<sub>E</sub>X: A hack-to-learn ebook</i>
<b>Book Reviews</b>	223	Jeffrey Barnett / <i>Book review: Fifty Typefaces That Changed The World, by John Walters</i>
<b>TUG Business</b>	225	TUG 2015 election
	226	TUG institutional members
<b>Advertisements</b>	226	T <sub>E</sub> X consulting and production services
<b>News</b>	227	TUG 2015 announcement
	228	Calendar

## 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

2014 dues for individual members are as follows:

- Regular members: \$105.
- Special rate: \$75.

The special rate is available to students, seniors, and citizens of countries with modest economies, as detailed on our web site. Also, anyone joining or renewing before March 31 received a \$20 discount.

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 \$105 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.

### 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.

T<sub>E</sub>X is a trademark of American Mathematical Society. METAFONT is a trademark of Addison-Wesley Inc. PostScript is a trademark of Adobe Systems, Inc.

[printing date: September 2014]

Printed in U.S.A.

## Board of Directors

Donald Knuth, *Grand Wizard of T<sub>E</sub>X-arcana*<sup>‡</sup>  
Steve Peter, *President*<sup>\*</sup>  
Jim Hefferon<sup>\*</sup>, *Vice President*  
Karl Berry<sup>\*</sup>, *Treasurer*  
Susan DeMeritt<sup>\*</sup>, *Secretary*  
Barbara Beeton  
Kaja Christiansen  
Michael Doob  
Steve Grathwohl  
Taco Hoekwater  
Klaus Höppner  
Ross Moore  
Cheryl Ponchin  
Arthur Reutenauer  
Philip Taylor  
Boris Veytsman  
David Walden  
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

+1 815 301-3568

### 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)

Copyright © 2014 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.

**2014 Conference Proceedings**

TeX Users Group  
Thirty-fifth Annual Meeting  
Portland, Oregon  
July 28–30, 2014

# TUGBOAT

COMMUNICATIONS OF THE T<sub>E</sub>X USERS GROUP

TUGBOAT EDITOR      BARBARA BEETON

PROCEEDINGS EDITOR      KARL BERRY

VOLUME 35, NUMBER 2

PORTLAND

•

OREGON

•

2014

U.S.A.

# TUG 2014 — Portland, Oregon, USA

July 28–30, 2014 ■ Mark Spencer Hotel

## Sponsors

**TeX Users Group** ■ DANTE e.V. ■ O'Reilly ■ Pearson  
with special assistance from many individual contributors. *Thanks to all!*

## Conference committee

Karl Berry ■ Jim Hefferon ■ Robin Laakso ■ David Walden

## Bursary committee

Taco Hoekwater, chair ■ Jana Chlebkova ■ Kaja Christiansen  
■ Bogusław Jackowski ■ Paul Thompson ■ Alan Wetmore

## L<sup>A</sup>T<sub>E</sub>X workshop leader

Cheryl Ponchin

## Participants

*Leila Akhmadeyeva*, Bashkir State  
Medical University  
*David Allen*, University of Kentucky  
*Pavneet Arora*, Bespoke Spaces  
*Justin Bailey*, Portland, OR  
*Kaveh Bazargan*, River Valley Technologies  
*Nelson Beebe*, University of Utah  
*Barbara Beeton*, AMS  
*Robert A. Beezer*, University of Puget Sound  
*Karl Berry*, TeX Users Group  
*Steven Brenton*, Long Beach, CA  
*Andrew Caird*, Ann Arbor, MI  
*Dennis Claudio*, Alameda County Superior Court  
*Jennifer Claudio*, Synopsys Outreach Foundation  
*David Crossland*, Metapolor  
*Michael Doob*, University of Manitoba  
*David Farmer*, American Institute of Mathematics  
*William Hammond*, San Diego, CA  
*Jim Hefferon*, Saint Michael's College  
*Julian Gilbey*, London, UK  
*Steve Grathwohl*, Duke University Press  
*Klaus Hoepfner*, DANTE e.V.  
*Joseph Hogg*, Los Angeles, CA  
*Richard Koch*, University of Oregon  
*Yusuke Kuroki*, Yokohama, Japan  
*Evguenia Kvistorf*, ICAO  
*Robin Laakso*, TeX Users Group  
*Jason Lake*, Minneapolis, MN  
*David Latchman*, TeXnical Designs

*Richard Leigh*, St Albans, UK  
*Adam Maxwell*, Port Angeles, WA  
*Doug McKenna*, Mathemaesthetics  
*Andrew Mertz*, Eastern Illinois University  
*Frank Mittelbach*, L<sup>A</sup>T<sub>E</sub>X3 Project  
*Ross Moore*, Macquarie University  
*Kruti Pandya*, Portland State University  
*Cheryl Ponchin*, Center for Communications  
Research, Princeton, NJ  
*Daniel Raies*, University of Oregon  
*CV Rajagopal*, River Valley Technologies  
*Will Robertson*, University of Adelaide  
*Chris Rowley*, L<sup>A</sup>T<sub>E</sub>X3 Project  
*Herbert Schulz*, Naperville, IL  
*Heidi Sestrich*, Carnegie Mellon University  
*Michael Sharpe*, UC San Diego  
*Joe Shields*, Portland State University  
*Keiichiro Shikano*, Tokyo, Japan  
*William Slough*, Eastern Illinois University  
*Alistair Smith*, Sunrise Setting  
*Paulo Ney de Souza*, Books in Bytes  
*Étienne Tétreault-Pinard*, Plotly  
*Nancy Van Cleave*, Eastern Illinois University  
*SK Venkatesan*, TNQ Books and Journals  
*Boris Veytsman*, George Mason University  
*David Walden*, East Sandwich, MA  
*Alan Wetmore*, US Army Research Lab  
*Ward Cunningham* (*special guest*), Portland, OR  
*Tracy Kidder* (*special guest*), MA & ME

# TUG 2014 — program and information

Sunday, July 27: opening reception and registration.

Monday, July 28: concurrent L<sup>A</sup>T<sub>E</sub>X workshop, Cheryl Ponchin.

Monday, July 28: post-session workshops: TeXShop, Herb Schulz; and arXiv.org, Steve Grathwohl et al.

<b>Monday</b> <b>July 28</b>	8:00 am	<i>registration</i>	
	8:50 am	Robin Laakso, TUG	<i>Opening</i>
	9:00 am	Ross Moore	<i>“Fake spaces” with pdfT<sub>E</sub>X — the best of both worlds</i>
	9:35 am	Frank Mittelbach	<i>Regression testing L<sup>A</sup>T<sub>E</sub>X packages with Lua</i>
	10:10 am	Doug McKenna	<i>Li<sub>t</sub>erate <math>\frac{T<sub>E</sub>X}{C}</math> Top part: JSBox</i>
	10:45 am	<i>break</i>	
	11:00 am	Doug McKenna	<i>Li<sub>t</sub>erate <math>\frac{T<sub>E</sub>X}{C}</math> Bottom part: literac</i>
	11:35 am	Paulo Ney de Souza	<i>A call for standards on the way to internationalization of T<sub>E</sub>X</i>
	12:10 am	Robert Beezer	<i>MathBook XML</i>
	12:45 pm	<i>lunch</i>	
	1:45 pm	<i>group photo</i>	
	2:00 pm	David Farmer	<i>Converting structured L<sup>A</sup>T<sub>E</sub>X to other formats</i>
	2:35 pm	William Hammond	<i>Will static CSS someday suffice for online rendering of profiled L<sup>A</sup>T<sub>E</sub>X?</i>
	3:10 pm	Keiichiro Shikano	<i>The easy way to define customized XML-to-L<sup>A</sup>T<sub>E</sub>X converters</i>
	3:45 pm	<i>break</i>	
	4:00 pm	Julian Gilbey	<i>Creating mathematical jigsaw puzzles using T<sub>E</sub>X and friends</i>
	4:35 pm	Pavneet Arora	<i>SUTRA — A workflow for representing signals</i>
	5:15 pm	q&a	
	<b>Tuesday</b> <b>July 29</b>	8:55 am	<i>announcements</i>
9:00 am		Karl Berry	<i>Building T<sub>E</sub>X Live</i>
9:35 am		Adam Maxwell	<i>T<sub>E</sub>X Live Utility: A slightly-shiny Mac interface for t<sub>l</sub>mgr</i>
10:10 am		Richard Koch	<i>MacT<sub>E</sub>X design philosophy vs. TeXShop design philosophy</i>
10:45 am		<i>break</i>	
11:00 am		Dave Crossland	<i>Metapolator: Why Metafont is finally catching on</i>
11:35 am		Michael Sharpe	<i>Recent additions to T<sub>E</sub>X’s font repertoire</i>
12:10 am		Etienne Tétreault-Pinard	<i>Plotly: Collaborative, interactive, and online plotting with T<sub>E</sub>X</i>
12:45 pm		<i>lunch</i>	
2:00 pm		Joseph Hogg	<i>Texinfo visits a garden</i>
2:35 pm		SK Venkatesan and CV Rajagopal	<i>T<sub>E</sub>X and copyediting</i>
3:10 pm		Boris Veytsman	<i>An output routine for an illustrated book</i>
3:45 pm		<i>break</i>	
4:00 pm		Kaveh Bazargan	<i>Creating a L<sup>A</sup>T<sub>E</sub>X class file using a graphical interface</i>
4:35 pm		Will Robertson and Frank Mittelbach	<i>L<sup>A</sup>T<sub>E</sub>X3 and exp13 in 2014: Recent developments</i>
5:15 pm	q&a		
<b>Wednesday</b> <b>July 30</b>	8:55 am	<i>announcements</i>	
	9:00 am	David Allen	<i>Experiences with tikzDevice</i>
	9:35 am	Andrew Mertz, William Slough, and Nancy Van Cleave	<i>Typesetting figures for computer science</i>
	10:10 am	Michael Doob	<i>Using animations within L<sup>A</sup>T<sub>E</sub>X documents</i>
	10:45 am	<i>break</i>	
	11:00 am	Dan Raies	<i>L<sup>A</sup>T<sub>E</sub>X in the classroom</i>
	11:35 am	Jim Hefferon	<i>Moving an online book to paper</i>
	12:10 pm	Kaveh Bazargan	<i>PDF files both to print and to read on screen</i>
	12:45 pm	<i>lunch</i>	
	2:00 pm	Leyla Akhmadeeva and Boris Veytsman	<i>Typography and readability: An experiment with post-stroke patients</i>
	2:35 pm	Alan Wetmore	<i>A quarter century of naïve use and abuse of L<sup>A</sup>T<sub>E</sub>X</i>
	3:10 pm	Ward Cunningham	<i>Another wiki? OMG why?</i>
	3:45 pm	<i>break</i>	
	4:00 pm	Tracy Kidder	<i>Trying to write about technical topics</i>
	4:35 pm	David Walden, moderator	<i>Panel: T<sub>E</sub>X and the Wider Wilder World — Hefferon, McKenna, Mittelbach, Rowley, Sharpe</i>
	≈ 5:15 pm	<i>end</i>	
	6:30 pm	<i>banquet</i>	at Bluehour ( <a href="http://bluehouronline.com">bluehouronline.com</a> ), 409 SW 11th Ave.

Some snapshots from the TUG 2014 meeting in Portland, Oregon. Photo credits (as noted): Pavneet Arora, Dennis Claudio, and Alan Wetmore. Thank you.



Michael Doob



Steve Grathwohl



Tracy Kidder, at the banquet



Cheryl Ponchin, Andrew Mertz  
(background: Keiichiro Shikano,  
Yusuke Kuroki)



Herb Schulz and Frank Mittelbach,  
at lunch

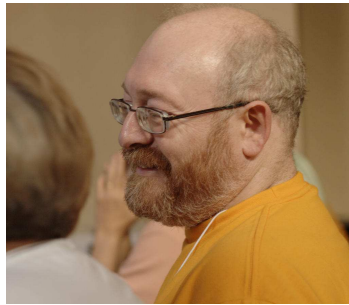


The traditional group photo. Smile!



PA

Paulo Ney de Souza



PA

Boris Veytsman



PA

Alan Wetmore, Richard Leigh,  
Heidi Sestrich



PA

SK Venkatesan



PA

Michael Sharpe



PA

CV Rajagopal, David Farmer



PA

Nelson Beebe



AW

Leyla Akhmadeeva



PA

Daniel Raies



PA

Will Robertson, Doug McKenna



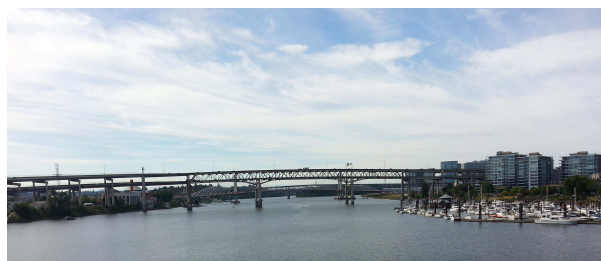
DC

David Latchman, Keiichiro Shikano,  
Pavneet Arora, Yusuke Kuroki,  
Will Robertson

## TUG 2014 in Portland

David S. Latchman

This year’s TUG conference took place in the city of Portland, the home of the  $\text{\TeX}$  Users Group. The city straddles the Willamette River near its confluence with the Columbia River, physically dividing the city into east and west Portland. The downtown area, where the Mark Spencer Hotel is located and where the conference was held, is in the west. Commonly known as the “City of Roses”, Portland features many famous rose gardens, most prominently the International Rose Test Garden. There is certainly a lot to do, from the micro-breweries, restaurants and food trucks found all over the city. If hands-on science demonstrations is something you might be interested in, the Oregon Museum of Science and Industry (OMSI) is worth a visit.<sup>1</sup> You can also visit the USS Blueback, a diesel–electric fast attack submarine moored in the river by the museum.



**Figure 1:** View from the SW Hawthorne Bridge

### 1 OMSI

The Oregon Museum of Science and Industry is a science and technology museum that features a variety of hands-on permanent exhibits focused on natural history, sciences, industry and technology. The museum was established in 1944, when it displayed its first collection of natural history objects at the Portland Hotel. In 1949, the museum moved to its first location on 908 NE Hassalo but by 1955, growing annual attendance entailed the building of a new site at Washington Park (this building is now occupied by the Portland Children’s Museum). Museum attendance would continue to rise and by the mid-1990s, continued expansion at Washington Park was deemed infeasible. The museum moved to the Station L power plant, donated by Portland General Electric, where it resides today.

<sup>1</sup> Editor’s note: OMSI has a place in  $\text{\TeX}$  history. The manual for the OMSI Pascal compiler was produced in  $\text{\TeX}$  in 1982 by Dave Kellerman and Barry Smith, before they went off to form Kellerman & Smith, which handled the first  $\text{\TeX}$  distribution for VAX/VMS systems.

David S. Latchman



**Figure 2:** An animatronic Dilophosaurus



**Figure 3:** The USS Blueback

#### 1.1 The USS Blueback (SS-581)

In 1994, the museum purchased the USS Blueback. This submarine appeared in the first season of the 1970s TV show, “Hawaii Five-O”. She also appeared in the 1990 film, “The Hunt for Red October”, where her crew was paid \$50US to cut their hair and put on Soviet Navy uniforms.

The USS Blueback is a Barbel-class submarine. This diesel–electric fast-attack submarine was one of the first production warships to utilize a teardrop shaped hull. The Barbel class submarines were designed to dive to a depth of over 700 feet, with a crush depth of 1050 feet. At the time of construction, the Blueback and her two sisters were the most technologically advanced submarines in the world.

The Blueback was the last non-nuclear submarine to join the United States Navy, entering service on 15 October 1959. She was also the last conventionally-powered combat-capable submarine to be decommissioned, leaving the US Navy with a fully nuclear submarine fleet.

## 2 Monday, July 28

### 2.1 “Fake spaces” with pdf $\text{\TeX}$ — the best of both worlds

The first presentation was by Ross Moore of Macquarie University. Moore began his talk by explaining that we can’t resize text and have it reflow in



PDFs in the same way text can be resized in an Internet browser or word processor. This isn't a problem with the PDF but rather with  $\text{\TeX}$  itself.

Moore explained that when Donald Knuth created  $\text{\TeX}$ , he chose to omit space characters from the output and carefully position the start of each word and punctuation character. Though this allowed for better full-justification and typesetting, the 'missing' spaces prevent on-the-fly justification and re-flow.

Moore said that this can be solved by getting the right spaces into the document—a special space character that can be stretched as needed. These interword space characters are available as new primitives in  $\text{\pdf\TeX}$  as of  $\text{\TeX}$  Live 2014 and can be controlled when and where they are used, e.g., they're not needed in mathematical content. These new word spaces conform to ISO-19005 standards and meet the PDF/A archival standards.

## 2.2 A modern regression test suite for $\text{\TeX}$ programming

Regression testing is the process of testing changes made to a system to ensure that older code still works and isn't in conflict with new code. One problem discussed in this talk by Frank Mittelbach is the hidden dependencies in  $\text{\LaTeX}$ ; packages can hook onto various layers or replace macros.

For many years,  $\text{\LaTeX} 2_{\epsilon}$  used a custom Perl script to perform regression testing on a large number of files to ensure that any changes do not break anything. This type of test is also useful for highlighting changes to  $\text{\TeX}$  Live. One of the steps in the development of  $\text{\LaTeX} 3$  was to produce a similar system to ensure that the development process was as smooth as possible.

Mittelbach discussed the various ways in which this can be done, e.g. making use of the  $\text{\TeX}$  log file. One problem is that the log contains much unnecessary information. Today,  $\text{\TeX}$  ships with  $\text{\Lua\LaTeX}$ . The included standalone Lua interpreter allows users to write platform-independent scripts without the need to install additional frameworks. Joseph Wright has rewritten the  $\text{\LaTeX} 3$  build scripts in Lua to avoid maintaining two separate systems. This should allow for an easy-to-use regression test suite for the  $\text{\TeX}$  community.

## 2.3 Liberate $\text{\TeX}$ , top part: JSBox

Doug McKenna discussed his monumental work in progress, JSBox, a  $\text{\TeX}$ -language interpreter written in portable C. McKenna hopes to remove some of the limitations of the  $\text{\TeX}$  engine by providing integrated support for 21-bit Unicode, namespaces, and OpenType, to name a few. To illustrate what is

possible with this new interpreter, McKenna showed how a fractal image could be placed and dynamically rendered in a  $\text{\TeX}$  document.

## 2.4 Liberate $\text{\TeX}$ , bottom part: literac

McKenna's second presentation focused on his program `literac`, a command line C program that enables literate commenting by converting source code written in languages that use C-style commenting syntax into a  $\text{\LaTeX}$  document.

Within the comments there are special commands to support common typesetting tasks. The `literac.c` is itself documented using the literate style and can typeset itself into its own manual. McKenna demonstrated how this can be done.

## 2.5 A call for standards on the way to internationalization of $\text{\TeX}$

Though  $\text{\TeX}$  can typeset documents in hundreds of languages, there has been little to no internal support with regard to language. Paulo Ney de Souza explained how the introduction of standards like ISO-639, ISO-15924, ISO-3166-1 and RFC-5646 can help  $\text{\TeX}$  users, especially those for whom English is not the preferred language.

## 2.6 MathBook XML

Robert Beezer discussed the need for a source format that can capture the structure of a scholarly document and convert it into a variety of formats. Though XML may have a bad reputation, Beezer contended that it can be compact and a good solution; MathBook XML is both lightweight and practical and simple enough for authors to create mathematical and scientific texts. He discussed a current project to convert XML to  $\text{\LaTeX}$  and HTML. As the project matures, conversions to other formats such as Sage and SageMathCloud worksheets and iPython notebooks will be developed.

## 2.7 Converting structured $\text{\LaTeX}$ to other formats

David Farmer explained that while PDF is good for printing, reading on screen is less than ideal, especially when a reader clicks on a link to view a reference or equation—the page “jumps around” making it difficult for a reader to visually scan a document.

Farmer said that his project, a collaboration with Beezer, to convert mathematics papers and textbooks from  $\text{\LaTeX}$  to HTML can solve this problem and provide an alternative to online PDF documents. When a reader clicks on a link, the reference appears in a box below which improves readability.

## 2.8 Will static CSS someday suffice for online rendering of profiled L<sup>A</sup>T<sub>E</sub>X

In this talk, William Hammond discussed the use of static CSS and its potential to render L<sup>A</sup>T<sub>E</sub>X on the web. The use of MathJax has shown that it is possible for CSS, along with JavaScript, to render mathematics online in HTML pages. Hammond explained that the problem with this method isn't quality but speed.

In recent times, CSS has become more powerful and though quality of rendering remains an issue, CSS by itself has become an adequate fallback option for online presentations. Hammond showed what CSS is capable of.

## 2.9 The easy way to define customized XML-to-L<sup>A</sup>T<sub>E</sub>X converters

In his talk, Keiichiro Shikano discussed the need for an XML-to-L<sup>A</sup>T<sub>E</sub>X converter. Shikano explained that though XML syntax may be considered ugly, it can be used to typeset books by applying a preferred L<sup>A</sup>T<sub>E</sub>X style.

This means that `xm12tex` is not a single converter application but rather a framework that adds a presentation layer in L<sup>A</sup>T<sub>E</sub>X; rules relating each XML element directly to a L<sup>A</sup>T<sub>E</sub>X command or environment is needed. Shikano demonstrated a straightforward and intuitive tool written in a dialect of Scheme to do this. The tool has been used to create and publish dozens of books.

## 2.10 Creating mathematical jigsaw puzzles using T<sub>E</sub>X and friends

Julian Gilbey first demonstrated a free Windows-based WYSIWYG software, Formulator Tarsia, that can be used to author mathematical puzzles. These puzzles are made of triangles and squares with questions and answers printed along the edges. Some of these puzzles were distributed to the audience for their solving pleasure.

Gilbey then described a similar T<sub>E</sub>X and Python-based system he created which can run on most operating systems.

## 2.11 SUTRA — A workflow for representing signals

Pavneet Arora talked about a practical workflow for representing signals of an electrical system.

## 3 Tuesday, July 29

### 3.1 Building T<sub>E</sub>X Live

Karl Berry gave the first presentation of the second day, an overview of T<sub>E</sub>X Live's construction. Berry explained that T<sub>E</sub>X Live follows the T<sub>E</sub>X Directory

Structure (TDS) which segregates files by type rather than placing packages in separate folders of their own. E.g., the `.cls` or `.sty` files are located in one folder while the documentation is located in another. Berry explained what this means for users and package designers. He ended with a discussion of how Knuth's original T<sub>E</sub>X is built from source.

### 3.2 T<sub>E</sub>X Live Utility: A slightly-shiny Mac interface for `tlmgr`

In his talk, Adam Maxwell discussed his program T<sub>E</sub>X Live Utility, a Mac OS X graphical user interface for the T<sub>E</sub>X Live Manager command-line tool. Maxwell explained that many Mac users appreciated a native Mac GUI interface.

Maxwell discussed some of the goals of a GUI interface. `tlmgr` works behind the scenes and does most of the heavy lifting. The program uses a model/view/controller design to give users a consistent Mac-like experience.

### 3.3 MacT<sub>E</sub>X design philosophy vs. TeXShop design philosophy

Richard Koch explained that MacT<sub>E</sub>X is an unmodified installation of T<sub>E</sub>X Live that, when installed, is fully configured and ready to use. TeXShop, on the other hand, is a Macintosh T<sub>E</sub>X front-end, written in Cocoa that incorporates some modern features that Apple recently introduced. This means that TeXShop is not cross-platform, unlike MacT<sub>E</sub>X.

Programming the front-end in Cocoa means that it automatically supports and can take advantage of many new Mac features, such as support for automatic saving and reloading of documents and the Retina display.

### 3.4 Metapolator: Why Metafont is finally catching on

This talk by Dave Crossland was a followup to his 2008 TUG presentation, "Why didn't METAFONT catch on?" Crossland introduced the Metapolator project, a tool allowing typeface designers to harness the power of METAFONT's parametric capabilities without writing any METAFONT code. Crossland gave several demonstrations.

### 3.5 Recent additions to T<sub>E</sub>X's font repertoire

In this talk, Michael Sharpe examined some of the newer additions and changes to the font families in L<sup>A</sup>T<sub>E</sub>X. Most of the additions have been new Roman text fonts. Sharpe also talked about the issues of matching math fonts.

### 3.6 Plotly: Collaborative, interactive, and online plotting with $\TeX$

Etienne Tétreault-Pinard introduced Plotly, an interactive online graphics platform that allows a user to embed and export graphics for publication. In the same way a core goal of  $\TeX$  is to allow anyone to produce high-quality documents with minimal effort, Plotly hopes to do the same with graphics.

Users can use data from Python, MATLAB, R, and Excel, among others, to create graphs. Graphs are rendered using JavaScript and annotations are done using  $\TeX$  in a MathJax display engine. As all data is stored in the cloud, users can not only get the same results across multiple platforms, they can share graphs online, publicly or privately.

### 3.7 Texinfo visits a garden

Joseph Hogg discussed using GNU Texinfo to document a collection of about 400 plants in the Herb Garden at The Huntington in San Marino, CA. The Huntington maintains a plant list as a resource for gardeners, docents, and visitors. This list was done in Word before The Huntington moved over to  $\LaTeX$ .

Though the Herb Garden is one of the smaller gardens, the plant list is updated every six months. By using Texinfo and other free software, Hogg and other volunteers are able to list and map on a grid the plants in the garden, which can then be downloaded or printed as a PDF. Visitors to the garden can view this PDF as they tour the gardens.

### 3.8 $\TeX$ and copyediting

SK Venkatesan and CV Rajagopal described how their `copyediting` package can be used to help copyeditors. It provides macros to help editors make corrections and improve consistency.

### 3.9 An output routine for an illustrated book

In this talk, Boris Veytsman showed how  $\LaTeX$  can be used to produce an illustrated book. Veytsman explains that producing an illustrated book in  $\TeX$  is difficult because the engine's algorithms assume there is a lot of text occasionally interrupted by an image;  $\TeX$  assumes that the text tells the story. In an illustrated book, on the other hand, the illustrations tell the story rather than the text. Veytsman discussed a method for creating an illustrated book using  $\TeX$ .

### 3.10 Creating a $\LaTeX$ class file using a graphical interface

Kaveh Bazargan explained that writing  $\LaTeX$  class files is a difficult and specialized job that requires an

intimate knowledge of  $\TeX$ . This makes it difficult for users to get into the deeper workings of  $\LaTeX$  as, quite often, potential programmers don't know all the options and inner workings of the individual packages that go into a class file. A graphical user interface (GUI) might help address this problem.

In his talk, Bazargan demonstrated such a GUI. A programmer can drag and drop, even order packages in the user interface and see the various options that are available. This system, named Batch Commander and created using LiveCode, dynamically updates the output so the effects of any changes are immediately visible. It is data-driven so that any package or parameter can be supported.

### 3.11 $\LaTeX$ 3 and `expl3` in 2014: Recent developments

In the last talk of the day, Will Robertson and Frank Mittelbach reported on  $\LaTeX$ 3 and `expl3`. Robertson explained that  $\LaTeX$ 3 and `expl3` are not the same and neither is  $\LaTeX$ 3 the next version of  $\LaTeX$ ; rather,  $\LaTeX$ 3 is an alternative to  $\LaTeX$  where some of its ideas can be layered onto  $\LaTeX$  2 $\epsilon$ . `expl3` originally stood for experimental  $\LaTeX$ 3. Though this has changed, the name has stuck.

Robertson explained that `expl3`'s goal is to make it easier to write  $\LaTeX$  packages and is now being used by many people "in the wild" and for many different package types. Both Robertson and Mittelbach discussed some ideas for rounding out the feature set.

## 4 Wednesday, July 30

### 4.1 Dynamic documents

Dynamic documents are documents that can be updated as the data changes. David Allen talked about using R, `tikzdevice`, `knitr` and  $\LaTeX$  to accomplish this. As an example, Allen used polling results from Kentucky's current senate election race to demonstrate how the above components can be used to track, graph and generate a report of both candidates' current standing.

### 4.2 Using animations within $\LaTeX$ documents

Though  $\TeX$  exists primarily for typesetting books, the software has evolved to where animations can be placed inside a document. Michael Doob discussed how this can be done, and how it can help students solve mathematical problems.

### 4.3 $\LaTeX$ in the classroom

Dan Raies discussed his experiences of using  $\LaTeX$  to teach mathematics to his undergraduate students.

Raies said that using L<sup>A</sup>T<sub>E</sub>X is great for students to submit assignments, and learning to use L<sup>A</sup>T<sub>E</sub>X is an important skill for their academic studies. Raies showed the audience some techniques for helping students get past common problems.

#### 4.4 Typesetting figures for computer science

Colleagues Andrew Mertz, William Slough and Nancy Van Cleave highlighted some L<sup>A</sup>T<sub>E</sub>X graphics packages to produce informative, high-quality diagrams and figures in computer science.

#### 4.5 Moving an online book to paper

Jim Hefferon talked about his experiences bringing his *Linear Algebra* book into print. The book was initially available only online, as a free download, but Hefferon was eventually convinced to produce a printed book that students could purchase. Hefferon described the self-publishing landscape and many different price considerations.

#### 4.6 PDF files both to print and read

Kaveh Bazargan talked about techniques one can use to make a PDF suitable for both print and online reading. Instead of creating two separate PDFs, Bazargan mentioned various packages that will allow a user to pack the information for both into one file. One example is the use of PDF layers. A reader can select between color and black and white images for screen reading or printing respectively. These layers can be shown or hidden as needed.

#### 4.7 Typography and readability: An experiment with post-stroke patients

Reading is a complex mental process and cognitive impairments may influence how well a patient can read and understand instructions. In this talk, Leyla Akhmadeeva and Boris Veytsman discussed their experiments on whether typeface design has an effect on readability on post-stroke patients.

Tests were done comparing legibility of the Para-type serif and sans serif fonts with post-stroke patients. Reading speed and comprehension were compared with results from healthy volunteers. If typography does indeed influence comprehension, publishers of books for post-stroke patients should take this into account. Though the sample size was small, the study found that differences in reading speeds between both font types were minimal. It is possible that, even after a stroke, the human brain remains remarkably adaptable.

Tracy Kidder

#### 4.8 A quarter century of naïve use and abuse of L<sup>A</sup>T<sub>E</sub>X

Alan Wetmore recounted his experiences with T<sub>E</sub>X and how it has evolved to what it is today. When Wetmore first encountered T<sub>E</sub>X, it was much more difficult to install and use but the strong community around T<sub>E</sub>X made this easier. Today, installing and maintaining T<sub>E</sub>X is much easier and its capabilities have grown.

#### 4.9 Another wiki? OMG why?

Ward Cunningham, the inventor of the wiki, talked about and demonstrated rebuilding the wiki concept to solve more complex sharing situations addressing some of society's toughest problems.

#### 4.10 Trying to write about technical topics

In the final talk, Pulitzer Prize-winning writer Tracy Kidder described some of his experiences with a computer engineering team as they created a next-generation computer, eventually becoming his book *The Soul of a New Machine*. He also mentioned his current work in progress, where he is coming back to the technical world for the first time since *Soul*.

◇ David S. Latchman  
Bakersfield, CA 93309  
david.latchman (at)  
texnical-designs dot com  
<http://texnical-designs.com>

---

#### Visiting TUG 2014

Tracy Kidder

I came to the conference partly to spend time with my friend Karl, partly to meet others who could help to educate me on the history of computer programming, and partly to find out what TUG is all about. For the past year and a half I have been hanging around in a corner of the world of commercial software development. I thought it would be both interesting and refreshing to be among people who are working on the extension and maintenance of a venerable piece of free software. During many of the formal talks, I found myself quite confused, lost in the technical details, but pleasantly so. One could do much worse than spend time among smart people who are obsessed with fonts. In short, the conference was everything I'd hoped for. And I am grateful for the great kindness and hospitality I was shown. ■

---

## Recent additions to T<sub>E</sub>X's font repertoire

Michael Sharpe

### Garalde family

The first 150 years of the printing industry, beginning with Gutenberg in 1450, bear a striking resemblance to the early years of the personal computer industry. Both were intensely commercial enterprises, though with some high-toned gloss — Bibles then, scientific computing now. However, the real money driving the printers of the late 15th century was to a considerable extent *indulgences* — big money-makers for the Church as well as printers. As I learned from the fascinating books of Andrew Pettegree [2, 3], some monasteries were ordering from printers and selling to sinners hundreds of thousands of generic indulgences as soon as the technology to do so became available. The closest modern analogue may be the claim that pornographic movies drove the rapid growth of VCR and, later, DVD players.

The Lutheran Revolt of the early 16th century against the excesses of the Church did not hurt printers, as they worked overtime to bring forth the voluminous tracts generated by the religious conflict. (One must bear in mind that the first newspaper did not appear until 1605.)

Given the importance of printed media in that period, it is not surprising that much talent coalesced around the technology, and the fonts developed during that brilliant advance are, in my opinion, some of the most appealing ever created. They are referred to now as “old-style” or *Garalde* in honor of Aldus Manutius and Claude Garamont [Garamond].

Gutenberg worked with fonts that we now call **Blackletter**, which remained the dominant typeface in the German countries through the first part of the 20th century. Caxton, the first English printer to use Gutenberg's technology, apprenticed in Belgium and set up the first printing house in England in 1476, and also used **Blackletter** exclusively. The first Roman font was developed by Nicolas Jenson of Venice, then the dominant commercial center of Europe, in the 1470s. Twenty years later, there appeared one of the great figures in publishing history — Aldus Manutius, also of Venice. Among other innovations, his company, the Aldine Press, invented the pocket book, italic type, greatly reduced the cost of books, standardized punctuation (introducing the semicolon), redefined book layout, and, through its “punchcutter” Francesco Griffo, whom we would now call a type designer, made a beautiful Roman font for the short book *De Aetna* by the poet Pietro Bembo, who became a major literary figure in the Italian Re-

naissance — lover of Lucrezia Borgia, major influence in standardizing the Italian language, creator of the *madrigal* form, and later, Cardinal of the Church. (The love letters between him and Lucrezia Borgia were considered by Lord Byron to be among the “prettiest” ever penned.) Modern revivals of the font used for *De Aetna* usually involve the name *Bembo*, though the basic free version is called *Cardo*, an obvious contraction of *Cardinal Bembo*. The fairly recent **fbf** package is based on *Cardo*, but with many changes — the ancient glyphs were stripped out, a kerning table was constructed for the Roman font, there being none in *Cardo*, and a Bold-Italic variant was created. Glyphs were added in all variants so that **fbf** has a full slate of `textcomp` characters and figures are available in proportional lining and old-style as well as tabular lining and oldstyle. CAPS are provided in all variants. (*Cardo* had small caps only in Roman, regular weight.)

SAMPLE OF **fbf**:

This is **fbf**, a free font package similar to Bembo. It has CAPS, a fine *Italic*, and a choice of number styles such as tabular oldstyle 0123456789.

Fifty years later, in Paris, Garamont introduced and continued to refine his Roman and Italic fonts, based initially on the *De Aetna* font and Griffo's later italic. Among the notable changes was the taming of *De Aetna*, reducing its ascenders and its over-arching ‘f’, planing off some of its more prickly features and creating more elegant capital letters. The remarkable account of Garamont's fonts, their origins and influences, by Beatrice Warde [1] is highly recommended. It contains, among other things, reproductions of much of the famous Egenolff-Berner specimen from 1592. The short version is that most Garamond fonts created in the early twentieth century were in fact based on later fonts by Jannon (c. 1620), not Garamont. Stempel Garamond (1925) is an exception, being based on a copy of the Egenolff-Berner specimen (see [1]) owned by the Stempel foundry.

By the late sixteenth century, fine printing was well established in parts of Europe, though not in England, judging by the mediocre quality of printing in Shakespeare's plays published during that period.

Recent Garamonds (URW++ Garamond No. 8, Garamond Premier Pro, EBGaramond) have followed Egenolff-Berner and Garamont's metal punches which appear to have been passed down to the Plantin foundry in Antwerp.

L<sup>A</sup>T<sub>E</sub>X now has a choice of two Garamonds:

- **garamondx** is an extension of Garamond No. 8, adding small caps and oldstyle figures in both weights and both shapes. Because of the license,

which is rather permissive but does not allow charging a fee, it cannot be distributed as part of T<sub>E</sub>X Live. Navigate to the url <http://tug.org/fonts/getnonfreefonts> for a script you can download that will install `garamondx` on Unix-like systems.

- `ebgaramond` (regular and italic only, no bold yet) is a very fine realization of Garamond that was recently added with L<sup>A</sup>T<sub>E</sub>X support.

SAMPLE OF `garamondx`:

This is `garamondx`, an extension of URW++ Garamond No 8. It has SMALL CAPS in all four styles, *Italics* and *Bold Italics*, and a choice of figures in all four styles, such as tabular oldstyle `o123456789`.

SAMPLE OF `ebgaramond`:

This is `ebgaramond`, a new realization of Garamond based on the Ebenolf-Berner specimen. It has very nice SMALL CAPS, a very fine *Italic*, and a choice of figures in all four styles, such as tabular oldstyle `o123456789`.

### Other serified roman families

PALATINO:

Named for the Italian writing master Giambattista Palatino, and inspired by Italian Renaissance fonts, Palatino has a larger x-height than typical old-style fonts and is more readable on-screen. It was one of the earliest fonts outside the Computer Modern family to gain T<sub>E</sub>X support, and remains one of the best-represented fonts for T<sub>E</sub>X.

- OpenType:
  - TeX Gyre Pagella + Asana Math;
  - TeX Gyre Pagella + Pagella Math.
- PostScript:
  - `newpxtext` + `newpxmath`;
  - TeX Gyre Pagella + `newpxmath`;
  - `mathpazo` (text and math), fewer features than the preceding;
  - `eulervm` as math can be used for a more informal appearance.
- Kpfonts (complete text and math) is based on URW++ Palatino clones, but has its own distinctive, light appearance.

TIMES:

Many choices are now available.

- OpenType:
  - STIX (text + math), and its unofficial extension XITS;
  - TeX Gyre Termes + STIX math;
  - TeX Gyre Termes + Termes Math.
- PostScript:
  - `newtxtext` + `newtxmath`/STIX;

- TeX Gyre Termes + `newtxmath`/STIX;
- STIX (text and math).

- STIX math has an unparalleled collection of mathematical symbols and alphabets matched to Times;
- STIX text fonts, as of version 1.1, lack some of the features of packages such as TeX Gyre Termes and `newtxtext`, but more is promised for 2.0.0 (<http://stixfonts.org>);
- the main difference between TeX Gyre Termes and `newtxtext` is that the latter has an option to use oldstyle figures as the default in text mode;
- MathTime (commercial but reasonably priced) is still a worthwhile Times-based math package with symbols generally lighter than STIX and having a number of features distinct from STIX;
- older choices such as `mathptmx` have now outlived their usefulness.

LIBERTINE:

LinuxLibertine is no longer new, but has undergone many fairly recent changes. It works well with the math package [`libertine`]`newtxmath`. In my opinion, this is an excellent choice for both screen and print. A number of recent math e-publications have used this combination.

SAMPLE OF `libertine`:

This is LinuxLibertine. It has SMALL CAPS, *Italic*, and a choice of number styles such as tabular oldstyle `o123456789`.

BASKERVILLE:

A “transitional” font (c. 1760), as was Plantin, the Times precursor. Baskerville (“the English Manutius”), was a master of fine detail, having been in the furniture finishing business (japanning) for a number of years. He set out to improve on Caslon, the then-dominant font throughout England and its colonies. Baskerville’s fonts, which bear the unmistakable heritage of oldstyle fonts, were favorites of Benjamin Franklin. Many commercial versions are available, most notably Storm Baskerville Pro. Free versions include:

- Baskervald (BaskervaldADF) was not designed with T<sub>E</sub>X in mind, and requires modifications to its ligature side bearings, its basic math character heights, and its kerning tables.
- (OpenType): `Baskervaldx.otf`, derived from BaskervaldADF, works OK with T<sub>E</sub>X.
- (PostScript): `Baskervaldx` + [`baskervaldx`]`newtxmath` works OK. `Baskervald[x]` lacks the high contrast that gives Baskerville its distinction as a print font,

and when scaled up to an x-height that matches the italic, it becomes a rather heavy Roman font.

- GFSBaskerville — for Greek, not Roman use.
- LibreBaskerville — lacks Bold Italic, and is designed as a web font, with larger x-height, larger counters and wider spacing than fonts intended for print output.

SAMPLE OF `Baskervaldx`:

This is `Baskervaldx`, a font similar to Baskerville. It has `SMALL CAPS`, *Italic*, and a choice of number styles such as `tabular oldstyle 0123456789`.

UTOPIA:

The design goals for Utopia seem to have been to avoid any trace of old-style ornamentation, and in this Adobe has been very successful. The font looks quite austere, with tightly packed letters and, in my opinion, overly small inter-word spacing.

Adobe donated the four basic PostScript fonts to the X Consortium in 1992, though the terms of the license were not clear. In 2006, it was rereleased to the T<sub>E</sub>X Users Group under clarified terms which allow modification and redistribution provided no name trademarked by Adobe is used.

- Fourier (Utopia text, fourier math) will make use of full (expert, Adobe) Utopia, if available.
- MathDesign [`utopia`] (Utopia text, MathDesign math) can also use expert fonts from Adobe.
- The ADF Venturis fonts are based on Utopia.
- An extension of the (free, basic part of) Utopia by Andrey Panov, dubbed Heuristica (Evrstika), is available now from CTAN, T<sub>E</sub>X Live and MiK<sub>T</sub>E<sub>X</sub> along with L<sup>A</sup>T<sub>E</sub>X support files. (The OpenType Heuristica fonts there have been modified, adding a number of lookup tables, so that they are parsed better by `otftotfm` and should work much better with `fontspec`.) It has added ligatures, `oldstyle` and superior figures and Roman small caps, and can be used with matching math via [`utopia`]`newtxmath`. (Fourier and MathDesign cannot currently use the Heuristica extensions, being tied to Adobe's organization of Utopia Expert.)
- The L<sup>A</sup>T<sub>E</sub>X support files for Heuristica now contain an option to set the factor by which to multiply the inter-word spacing, `\fontdimen2`. The default value is 1, and the value 1.2 is suggested as a starting point.
- As of version 1.04, a `sinf` lookup has been added for subscript figures, superior letters have been added to each font and the `sups` lookups have been extended to cover letters so that French abbreviations like  $M^{\text{me}}$  are available.

SAMPLE OF `Heuristica`:

This is Heuristica, an extension of Utopia. It has `SMALL CAPS`, *Italic*, and a choice of number styles such as `tabular oldstyle 0123456789`.

CHARTER:

Bitstream contributed their four basic Charter fonts to the X Consortium under a very liberal license, and they have been available in T<sub>E</sub>X for many years. Their low contrasts, high x-heights and use of piecewise linear outlines where possible may make them interesting again as fonts that will render well on small devices and perhaps projected slides. (It's worth noting that their designer, Matthew Carter, created Georgia for Microsoft. It is widely considered to be one of the clearest serified fonts for viewing on screen, and bears a number of similarities to Charter, though the latter is heavier.)

The XCharter fonts add superior figures and small caps in all styles, plus `oldstyle` figures (proportionally spaced only) with options to select the form of 'one' — `oldstyle` gives you 1 (the default if no option is specified) and `oldstyleI` gives you I. The original Charter fonts had some idiosyncratic kerning, especially with P-comma, P-period and P-hyphen. These have now been corrected in all styles.

SAMPLE OF `XCharter`:

This is XCharter, an extension of Charter. It has `SMALL CAPS`, *Italic*, and a choice of number styles such as `proportional oldstyle 0123456789`.

### Typewriter fonts

The `courier` font that has long been available on CTAN is too light and too spread out for any use I can imagine in T<sub>E</sub>X, except to generate examples of what not to use. There are now several choices that are more attractive than you might expect for a monospaced font. Most are not new, but have been renovated recently so may appear new to you.

SERIFED TYPEWRITER FONTS:

- The `zlm` package provides access to all features of TeX Gyre Latin Modern Typewriter, a very substantial extension of `cm`. Best suited to lighter Roman fonts, though it can be scaled to be a better match up for some heavier Roman faces. The fonts themselves have been described thoroughly by Will Robertson in [4]. `SMALL CAPS` are available in regular, upright only. The font does have a bold variant, but the boldness is almost imperceptible due to the design goal of keeping the widths of bold glyphs the same as those in regular weight. The individual pieces are regrettably inconvenient to access through the `lmodern` package.

A sample of text using `lmtt` and its bold variant.

- The `newtxtt` package is built on an enhanced version of the typewriter fonts from the `txfonts` package, with the addition of several choices of forms for ‘zero’. The fonts are of the same width as `cmtt`, but are heavier and taller, matching Times weight and size. **SMALL CAPS** are available in upright shape only. The newest version of the package has an option to reduce the interword space, so that, while it is no longer monospaced, it looks better for blocks of text that do not need to be aligned letter by letter.

A sample of text using `newtxtt` and its bold variant.

#### SANS SERIF TYPEWRITER FONTS:

Two good packages are now available.

- `Inconsolata--zi4` is an extension of the original `Inconsolata` package by Karl Berry, offering regular and bold weights, a choice of styles for ‘zero’, ‘l’ and quotes. It is based on an extension of Raph Levien’s fine *Inconsolata* fonts, which are not dissimilar to Microsoft’s *Consolas*.

A sample of text using `inconsolata` and its bold variant.

- The `beramono` package is based on Bitstream’s Vera Sans Mono. All glyphs are unmistakable. It is available only in T1 and TS1 encodings. The more recent `DejaVu Sans Mono` package is a further extension with many more encodings and accented glyphs.

A sample of text using `beramono` and its bold variant.

#### Sans serif fonts

There are now several choices of (proportionally spaced) sans serif fonts available to  $\text{\TeX}$  users, among the more recent being `cabin` (similar to Gill Sans), `raleway` and `SourceSansPro`. As fonts of this type are frequently made available in a multiplicity of weights, their support files can profit from use of the `mweights` package that allows you to choose which weight will be called “regular” and which will be called “bold”, independent of the corresponding choices for roman and typewriter.

Sans serif fonts are often used to render slides, as their simpler geometry is usually less corrupted by rasterization than serifed fonts. They are in many

cases poor at distinguishing homoglyphs such as upper case ‘i’, lower case ‘el’ and the lining figure ‘one’ [5]. This makes their use for file names problematic. In much the same way, it can be difficult to distinguish a sans serif proportional glyph from a sans serif typewriter glyph, and if both are used to indicate distinct objects (e.g., sans serif for menu items and display items in a computer program, `tt` for file names), then confusion is quite possible. (If you use sans serif for headings, as in German typography, and only for headings, this is not an issue.) If you make serious use of sans serif for other than headings, it may be wise to choose a serifed typewriter font even though your eye may wander to sans serif typewriter.

Note one peculiarity of `cabin` if you use it for email addresses, as in `person@example.org` — that white-on-black @ is unfortunate and the font would benefit from an alternate, black-on-white, symbol.

#### Further

For an expanded collection of descriptions and samples of many of the fonts mentioned above, see <http://math.ucsd.edu/~msharpe/RcntFnts.pdf>.

#### References

- [1] Beatrice Warde, writing as “Paul Beaujon”. The “Garamond” types, sixteenth and seventeenth century sources reconsidered. *The Fleuron*, pp. 131–179, 1926. [http://www.garamond.culture.fr/kcfinder/files/3\\_3\\_4\\_article\\_beatrice\\_warde.pdf](http://www.garamond.culture.fr/kcfinder/files/3_3_4_article_beatrice_warde.pdf)
- [2] Andrew Pettegree. *The Invention of News: How the World Came to Know About Itself*. Yale University Press, New Haven, 2011.
- [3] Andrew Pettegree. *The Book in the Renaissance*. Yale University Press, New Haven, 2014.
- [4] Will Robertson. An exploration of the Latin Modern fonts. *The Prac $\text{\TeX}$  Journal*, 2006-1, 2006. <http://tug.org/pracjourn/2006-1/robertson/robertson.pdf>
- [5] Charles Bigelow. Oh, oh, zero! *TUGboat* 34:2, pp. 168–181, 2013. <http://tug.org/TUGboat/tb34-2/tb107bigelow-zero.pdf>

◇ Michael Sharpe  
<http://math.ucsd.edu/~msharpe>



---

## Experiences converting from PDF-only to paper

Jim Hefferon and Lon Mitchell

### Abstract

Offering a textbook for free download has become common. With the growth of on-demand printing, adding a paper option is an easy way to distribute the work in a format that some users prefer. We will give an overview of today's landscape of print on demand and discuss some differences in delivering works in the two media.

### 1 Background

The text *Linear Algebra*<sup>1</sup> by the first author has been available since 1996, with L<sup>A</sup>T<sub>E</sub>X source, under a license that is Free, in this case either the GNU Free Documentation License (GFDL) or Creative Commons CC-BY-SA. It covers a standard US first course, has extensive question sets with worked answers for every question, and is supplemented by classroom beamer slides and a lab manual. In electronic form it is widely accessed; during the academic year it has 30K–40K downloads per week. In the past it has only been available online but it is now also offered as a paper version. The experience of that addition is detailed here.

Until now, to get paper copies to students an instructor would download a PDF, make a master copy, produce bound copies of that, and then students paid to defray the cost, usually at their college bookstore.

This model has implications. First, it suggested that the page size should be US letter, 8.5×11 inches, since that is the most convenient form for printing, copying, and binding here in the US. Second, typically instructors would use a black and white printer so the book design couldn't be ambitious with color (hyperlinks are in blue, which prints to a dark gray). Finally, print resolution was an issue. In 1996 printers were typically 300 dots per inch. That is low enough that some shading and graphics vanished so illustrations had to limit those. Another impact of the low resolution is that, since in this model students got copies of copies, Computer Modern characters often had dropouts, particularly in subscripts or superscripts. Consequently the book style was switched to use Concrete, with Euler for math.

Thus, even with a distribution model that was online-only, printer capabilities influenced book design decisions.

---

<sup>1</sup> <http://joshua.smcvt.edu/linearalgebra>

### 2 Desire for paper

Through the almost twenty years the book has been available, requests for paper copies have come in regularly.<sup>2</sup> Some came from self-studiers who were simply used to a certain format. More worrying was that sometimes instructors who were potential adopters said that they would not use the text because of the hassle of having it printed.

Despite those requests, the prospect of dealing with bounced checks, return copies, and lost-in-the-mails kept the book officially available online only.

### 3 Publishing

Those who lay out their own text using T<sub>E</sub>X and post the result for free download on a website have provided (in a basic way) two of the main services historically provided by publishers: typesetting and marketing. Although many books produced this way could benefit from additional effort and advice in both areas, T<sub>E</sub>X and the Internet can make self-publishing feasible and attractive.

A print edition requires a printer, distribution, and retail (all of which are disjoint from what typically defines a publisher: owning an ISBN). Multiple companies now offer the convenience of a combination of two or all three, but beware those who would also bundle in other services such as marketing or book layout; such services, if needed, are almost always better when acquired separately. Perhaps the most important caveat is that companies that offer all three services may also act as the publisher (because they own and provide the ISBN). This is not necessarily bad, as it can save time and effort, but it will mean that you may lose control over some decisions normally made by a publisher, such as those discussed below.

Publishers choose a list price and a 'discount' for each title. Retailers pay the list price less the discount, and publishers must pay the print cost and distribution fees from this revenue. The industry standard discount is 55%, but some distributors will allow less. Smaller discounts can discourage retailers from carrying your title. For many years, online retailers and university bookstores seemed content with discounts as low as 25%, but that may be changing. This is particularly unfortunate as the higher costs of print on demand (when compared to offset printing) are amplified by higher discounts.

---

<sup>2</sup> Studies suggest up to 75% of students prefer a paper copy: <http://www.studentpirgs.org/reports/cover-cover-solution>

In our opinion, the ideal print edition of a Free book should have a (very) low price, be easily available to college bookstores and sold by multiple online retailers, and require little or no effort of the author beyond initial setup. Unfortunately, to the best of our knowledge, there is not currently an easy way for an author working alone to achieve all of these simultaneously. There is a great need for a new model in the book business that is attuned to open-source projects, but that business is dominated by a few large companies and change seems far off.

With *Linear Algebra*, using a “low-profit” company (L3C) created by the second author to act as the publisher, we have achieved (for the moment) each of the three goals above.

#### 4 Why do it yourself?

The GFDL has a fascinating premise, providing specific language that allows anyone to publish your book. This turns the usual relationship between author and publisher around—anyone can grab a GFDL-licensed work and offer a version for sale, including a print version. That has happened a number of times with *Linear Algebra*. This may be attractive to authors who would like to have a print edition but would rather not have to do any of the work. The GFDL may not be to everyone’s liking, however, as it does not place many limits on what the publisher can and cannot do; for example, previous print versions of *Linear Algebra* offered for sale have not been updated when the online text has been updated. One could easily adapt a more restrictive license to a similar function by offering specific exceptions to reputable potential publishers.

If you are interested in having your work published in this way, you can be of great help to potential publishers in your use of T<sub>E</sub>X. For example, it is likely a publisher will need to reformat your work to an industry-standard page size; using already provided descriptive commands (such as `\emph` instead of `\textit`) and/or providing new commands that can easily be redefined in the preamble (such as `\beforeexercisesskip` instead of a `\medskip`) is highly recommended.

Open source books need not be new material; if you own the copyright to an out-of-print book (or can acquire it from the publisher), you can make a quality contribution to this growing movement.

#### 5 Money

With reputable printers, costs are significantly less for industry-standard book sizes. The original size of *Linear Algebra*, US letter 8.5 × 11 inches, is not standard in printing, and is expensive. Lower-level

textbooks often have a larger page size than upper level ones, so we went with one of the larger standards, 7.5 × 9.25 inches. One advantage of this size is that we can retain L<sup>A</sup>T<sub>E</sub>X’s default text width and so downloaders have the option to print onto US letter.

We had some other costs: setup fees, an annual listing fee, and ISBNs. (Traditionally, if two copies of a book had the same ISBN then a buyer could depend on them being substantially the same, so any major or even many minor changes in a book should trigger a new ISBN.)

On the revenue side, having a little extra from each copy helps with things like being able to display and distribute free books at the Joint Math Meeting. In any event, rounding up to the nearest multiple of five is safer than rounding down.

Note that with a GFDL book, anyone could undercut the price. However, most readers interested in the book go to its web page, which points to the official paper edition. The price of that is so reasonable that we don’t see users looking for a non-official (and possibly outdated) version.

The takeaway is that we used Ingram’s print on demand service at a \$20 list price, which splits into (about) thirds for printing, the retailer, and us.

#### 6 The outside

Coming out with a paper copy seemed to call for giving it a nice cover. With print on demand services, the cover is uploaded as one file that shows the front, back, and spine. Any revision that adds or removes pages changes the size of the spine (*Linear Algebra*’s 498 pages yield a spine of 1.001 inches). This means that an author should create the cover using a process that is easily redone. In this case, the elements in the cover file were arranged in a L<sup>A</sup>T<sub>E</sub>X `picture` environment to get exact locations.

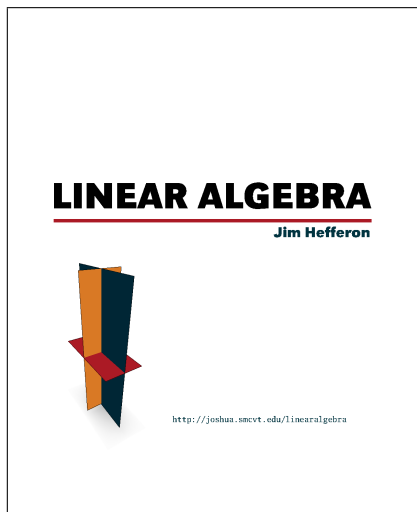
There is no printing on the inside of the cover. Consequently the driver file `book.tex` got a flag so that if the PDF is for paper then it makes an extra inside sheet, a title page, whose back is the page with the list of symbols. (This flag also prints hyperlinks in the book’s body in black instead of the electronic PDF’s blue.)

Traditionally the back cover has some marketing text, which is a challenge to write for an academic unused to selling. The ISBN and bar code go there also, in a box about an inch tall.

##### 6.1 Art

The main difficulty with the cover was not technical. Getting a capable person to execute a graphic proved to be hard. For example, a colleague in Media Studies recommended a talented student who

seemed interested and agreed to produce it for a fair price. But it never appeared. After a half dozen such episodes, out of embarrassment at how long it was taking, ten minutes spent in Asymptote drawing the planes and a half hour spent on Kuler<sup>3</sup> gave the result below.



## 7 The inside

Here are the basic dimensions.

```
\usepackage[papersize={7.5in,9.25in},
  textwidth=345pt,
  inner=.8in,
  top=.85in, bottom=.85in,
  headsep=12pt,
  bindingoffset=0.4in,
]{geometry}
```

The `top` and `bottom` setting squeezes more lines on a page. The more pages there are, the more the binding will use up page space, so the inside margins needed to be set a little larger.

Repaginating took a long time. The `microtype` package helped with linebreaks, and forbidding widows and orphans along with using `\raggedbottom` helped with page breaks. But each page required individual attention, including some rewriting.

### 7.1 Lurking culprits

At the first submission, besides the binding margin there was only one thing to fix: a non-embedded Type 3 font somewhere in the document. The program `pdffonts` gave a page number. The only suspicious things there were two graphs from *Sage*.

Some spelunking<sup>4</sup> revealed that *Sage* required passing to the plot the parameter `typeset=latex`.

<sup>3</sup> <http://kuler.adobe.com>

<sup>4</sup> <http://trac.sagemath.org/ticket/14664>

With that, the fonts were embedded and the print on demand technical requirements were satisfied.

### 7.2 Shades of gray

The major issue remaining in the text is that many of the illustrations use shaded lines. One example pictures the linear map  $h(x) = x + y$  from the plane to the line. The text has three drawings: first, a vector from the line  $x + y = w_1$ ; second, another vector from the line  $x + y = w_2$ ; and finally, the vector that is the sum of the two, drawn as ending at the line  $x + y = w_1 + w_2$ . Each of the three gives its vector in normal face with the associated line shaded. In the PDF a level of 0.85 gray seemed visually right (this is mostly white, with fifteen percent black). But on the printed page this sometimes came out too light, to the point where it was nearly invisible.

To bring out the lines on paper without compromising the online version, the illustrations were adjusted to make the lines thicker and to have the darkness 0.70. (A few illustrations in the book used this darkness level and it seemed to print fine.) However, even with this change, in the current paper version of the text some lines are too light and the quality of the illustration suffers. An author considering this issue may want to go with dashed lines, or some other larger stylistic change.

### 7.3 In print, mistakes look *really* bad

The book has been available for twenty years and there have been any number of corrections of typos, and some errors of fact as well. That was routine. But when the paper version came from the printer with a glaring error, it was startling. (A `LATEX` `picture` environment was picking up a `\setlength` that had not been put inside a group and so the environment was three times the size of the page.) The next PDF sent to the printer got very close scrutiny.

## 8 The result

The book is now available at the usual online retailers and can be ordered at wholesale pricing by college bookstores from Ingram's catalog. The download page gives a link. This answers the requests that have come in over the years, without requiring any book-toting by the author.

- ◇ Jim Hefferon  
Saint Michael's College  
`jhefferon (at) smcvt dot edu`
- ◇ Lon Mitchell  
Orthogonal Publishing  
`l3c (at) orthogonalpublishing dot com`

## Texinfo visits a garden

Joseph Hogg

### Abstract

Texinfo [6] offers PDF, HTML, and Info output formats from a single source file. This feature appealed to me because I wanted to produce both HTML and PDF files to describe the plant list for the Huntington's Herb Garden in San Marino, CA. Although used primarily to document GNU software, Texinfo offers a simple framework for publishing enumerated lists that can be manipulated with shell scripts to provide information about 400 plants in 25 beds in this one-quarter to one-half acre garden.

### 1 Introduction

The Herb Garden (Figure 1) is one of the smaller gardens at the Huntington Library, Art Collection, and Botanical Gardens located in San Marino, California ([huntington.org](http://huntington.org)). There is a Curator and full-time Gardener for the Herb Garden. In addition, about 60 docents and volunteers help maintain the Garden and interact with visitors. I have been a volunteer in this Garden for about five years.

There have always been plant lists for the Garden, but the mix of plants changes by season and a greater variety of plants is being introduced. Having an up-to-date plant list helps docents and volunteers learn the Garden and be able to inform visitors on a plant's identity, history, and uses.

### 2 File formats and file hosting

I initially thought an HTML version of the plant list would give users access to the plant list in a familiar web interface and the printed PDF version would be available in the Garden. I later learned that a web server hosted at the Huntington was not available for volunteers to upload files, but hosting PDF files for docents and volunteers has been available for several years. Huntington staff uploads PDF files as needed. Today, docents can view the PDF file on the Huntington's Volunteer web site, hard copy in the Garden, and look at the PDF file on their tablets or smart phones. I send the Plant List to docents and volunteers in an email attachment.

### 3 Texinfo: Simple markup, full-featured documents

In addition to the multiple output formats possible from a single source, another attractive feature of Texinfo is the simple markup scheme. Enumerated lists, `@enumerate`, are used in every bed. Each item in the list uses `@item` for a list item, `@i{italicized botanical name}`, and `@uref{web link}` for each

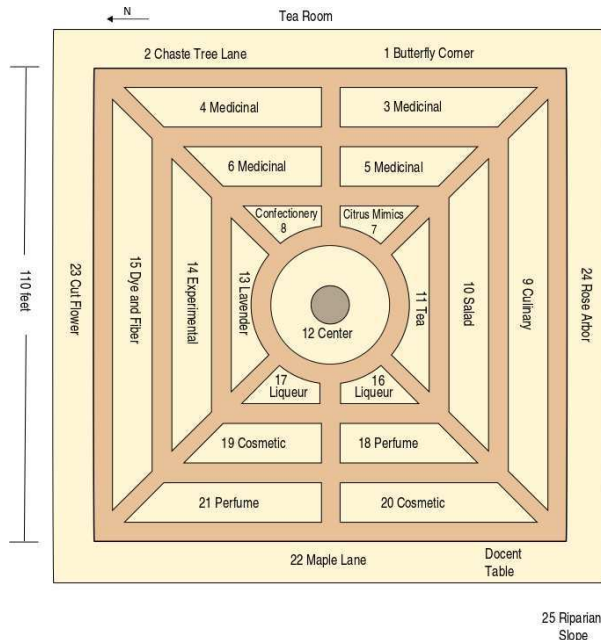


Figure 1: Herb garden map

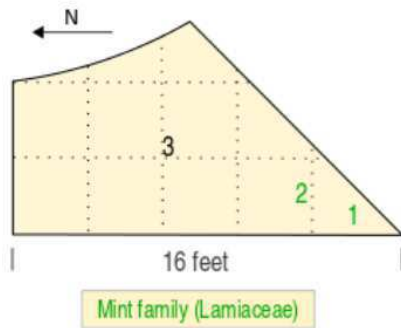
of the approximately 400 plants on the list. Common names and family names are not italicized for plants. A plant's names, family, and web links are placed on the same line. This is important for the shell scripts I wrote since counting plants is equivalent to counting lines. A table of contents, `@contents`, with entries from `@chapter` or `@unnumbered`, and an index, `@printindex`, are similarly straightforward.

With a list of plants done, I wondered how many there were in the list and how many were distinct in the Garden, which has 25 beds containing some duplicates. A botanically distinct plant appears only once in a list of plants for a specific bed even though its copies in that bed are indicated on a map drawn in Inkscape [4]. For example, there are two Bay trees in the Culinary bed and two in the Dye and Fiber bed. We count only one of them in each bed and only one as a distinct plant summarizing across the Garden. Since we captured plant families, I wanted to know how many families were represented in the Garden and how many distinct plants were members of each family.

### 4 Plant list summary

Attempting to answer these questions by manual counting would be error-prone and tedious. But writing a shell script [2] to calculate this information directly from the Texinfo source file was a matter of applying classic Unix tools [5] and piping them together to process each line in the Plant List. I was surprised by the results.

## 16 Liqueur, south



1. Wood Rosemary, *Rosmarinus officinalis* 'Wood', Lamiaceae, [pfaf](#), [wiki](#)
2. Clary Sage, *Salvia sclarea*, Lamiaceae, [pfaf](#), [wiki](#)
3. Centennial Hops, *Humulus lupulus* 'Centennial', Cannabaceae, [pfaf](#), [wiki](#)

Updated July 1, 2014

**Figure 2:** Bed 16 Liqueur page, output

In the Preliminary Plant List of July 15, 2014, there are 394 plants in the list of which 333 are distinct. These 333 distinct plants are distributed among 59 plant families. Five of these plant families contain 63 percent of the distinct plants in the Garden. They are the mint, sunflower, rose, geranium, and nightshade families. This represents a concentration of plants probably typical of herb gardens with their emphasis on edible, scent, and medicinal plants. Equally surprising to me was that 28 of the 59 families contain only one distinct plant. For example, the Henna Tree, Jojoba, and Caper bush. This represents diversity in the Garden. This concentration and diversity of plants gives docents many opportunities to develop histories, stories, and plant relationships to inform and entertain visitors. Each plant in the list has one or two web links that docents can use as starting points for developing their own information.

### 5 Web links

In an early discussion with the Gardener about requirements for this list, we talked about adding a few sentences about some of the plants. This had been a feature with plant lists in the past when web access was limited and resources like Wikipedia ([en.wikipedia.org](http://en.wikipedia.org)) and Plants for a Future ([www.pfaf.org](http://www.pfaf.org)) were not available. Trying to maintain these comments in a changing plant list seemed like a headache for both the Gardener and me. Besides, the authors at Wikipedia and Plants for a Future are

```
@node Liqueur (16)
@chapter Liqueur, south

@image{./graphics/16_liqueur070114, 4in, }

@sp 1
@enumerate
@item Wood Rosemary, @i{Rosmarinus officinalis} 'Wood',
  Lamiaceae, @uref{http://www.pfaf.org/..., pfaf}, ...
@cindex @i{Rosmarinus officinalis} 'Wood' (Rosemary, Wood)
@cindex Rosemary, Wood (@i{Rosmarinus officinalis} 'Wood')
@cindex Lamiaceae (Mint family)
@cindex Mint family (Lamiaceae)

@item Clary Sage, ...
...
@item Centennial Hops, ...
...
@end enumerate

@sp 1
@noindent Updated July 1, 2014
```

**Figure 3:** Bed 16 Liqueur page, Texinfo source

already knowledgeable, filter the information they present, and update their pages regularly.

### 6 Index

The Texinfo manual offers advice on creating indices that will be useful for a variety of readers. Many persons know a plant's common name, but not its botanical name, and may not know its family. Furthermore, a variety of common names may be used. In this Index, a plant's botanical name and common name are cross-referenced as are the plant's family name with its common name if it has one. That's four index entries for each of about 400 plants. Editorial judgment is also needed when redundancy in an alphabetized index is caused by the similarity of a common name and its botanical name, for example, Jasmine and *Jasminum*.

A shell script [3] is one way to reduce this effort, while adding accuracy and consistency. Since both botanical, common, and family names are available in the line describing each plant, it is, in theory, easy to create four index entries and write them directly into the Texinfo source file. Exceptions need to be made for names that are similar, or that have unusual common names (Vick's Plant, Jupiter's Beard). Each time the plant list is revised, I run the script to create index entries. Since this script takes the names from the list item, it is important to check for spelling errors that would ripple through the main document and Index. Figure 2 shows the page in the list about Bed 16 Liqueur, and Figure 3 the corresponding Texinfo source (abridged), after the index entries have been added.

## 7 Plant ranks

The Plant List Summary offers a glimpse into family memberships. That suggests that we should publish a list of all the plant families, the genera in each family, and the species in each genera. This information is already available in the `@item` line, but needs to be consolidated in its own hierarchy. The Unix `sort` utility organizes the hierarchy alphabetically, but Texinfo markup needs to be added to indicate sections and subsections. Again, a shell script works through the sorted plant hierarchy to add `@section` and `@subsection` markup and then writes this revised hierarchy to a file. This file is then integrated into the Texinfo source file with an `@include` statement before the Index.

Making the taxonomic hierarchy accessible raises awareness of relationships in the Garden and can help docents summarize information they provide to visitors. At a recent docent and volunteer meeting, one of the docents talked about the uses of plants in the ginger family. And, many docents are pleased to know that madder, dyer's woodruff, gardenia, and coffee are in the same family (Rubiaceae).

## 8 Hybrid plants

*The Chicago Manual of Style* [1] recommends using the mathematical times symbol to identify a hybrid plant, as in Sweet Lavender or *Lavandula* × *heterophylla*. In earlier versions of the Plant List, I used the familiar lower case roman 'x'. I was able to change it to the × symbol in the Texinfo source file, but could not get it to appear in the Index. How to do this turned into a question to Karl Berry by email and Karl replied with a `TEX` macro that worked. This adjustment to hybrid plants in the source files, Texinfo and plant ranks, is done with a one-line `sed` script after the other scripts have been run and just before processing the Texinfo file to PDF output. The typographical quality of the Plant List has improved. Thank you Karl.

## 9 Recap and next steps

The Gardener for the Herb Garden selects plants, supervises volunteers, and interacts with docents, volunteers, and visitors. She revises the designs of various beds as the Garden evolves. The diversity of plants and ongoing changes to the Garden make an accurate plant list an asset for the Curator, Gardener, docents, volunteers, and visitors who often ask for help identifying plants and are interested in the history and uses of these plants. Each of the

25 beds in the Garden contains plants used for a particular purpose, for example, perfume, dye and fiber, cooking, and medicine.

Each of these 25 beds appears in the Plant List with a diagram showing the location of each of the plants growing in that bed. The diagram is produced in a SVG format in Inkscape and then imported by Texinfo as a PNG graphic file and placed at the top of the PDF page for that bed.

Before this project, the Plant List was the responsibility of the Curator or the Gardener. This project has been a productive effort between a Huntington staff member and a volunteer. The Gardener has told me she would like to take over the maintenance of this list and I will work with her over the next year to make this transition successful.

The Huntington and the Gardener use MS Windows on their computers. Moving to Windows versions of Inkscape, Texinfo, and `TEX` should not be difficult. I also don't anticipate difficulties using a plain text editor for the Texinfo source and entering a few commands in some shell. The difficulty may come from having to move the scripts from my GNU/Linux (Ubuntu) machine to a Windows machine. The Unix-like environment Cygwin ([cygwin.com](http://cygwin.com)) is one option. Another is to rewrite the scripts in Perl.

We will work through these alternatives in the coming months; comments are welcome.

## References

- [1] *The Chicago Manual of Style*, Sixteenth edition, University of Chicago Press, 2010, ISBN-13: 978-0-226-10420-1.
- [2] *Classic Shell Scripting*, Arnold Robbins and Nelson H. F. Beebe, O'Reilly, 2005, ISBN: 978-0-596-00595-5.
- [3] *Effective awk Programming*, Third edition, Arnold Robbins, O'Reilly, 2001, ISBN: 978-0-596-00070-7.
- [4] Inkscape. <http://www.inkscape.org/en>.
- [5] *sed & awk*, Second edition, Dale Dougherty and Arnold Robbins, O'Reilly, 1997, ISBN-13: 978-1-565-92225-9.
- [6] Texinfo. <http://www.gnu.org/software/texinfo>.

◇ Joseph Hogg  
Los Angeles, CA, USA  
[joseph\\_dot\\_hogg \(at\) gmail\\_dot\\_com](mailto:joseph_dot_hogg_at_gmail_dot_com)

## MacTeX design philosophy vs. TeXShop design philosophy

Richard Koch

I went to the Apple Developer Conference in May, 2000. Developers at this conference were supposed to receive the release version of OS X, but in the keynote address, Steve Jobs announced that the new release would be renamed OS X Public Beta with a price reduced from \$130 to a handling fee of \$15. After the keynote, a knowledgeable friend translated: “OS X has been delayed by a year.”

As a sop to the audience, Apple held a software raffle during this conference, the only time I’ve heard of them doing so. Every developer got something, but it soon transpired that almost everybody got a schlocky piece of software on a CD, shrink wrapped against a flimsy piece of cardboard.

I was looking through this TUG talk and it isn’t very interesting. So I decided to give each attendee of the TUG conference a free piece of software.

The schlocky software Apple gave developers in 2000 was a forerunner of iTunes, in the days before the iPod and all that. I, unfortunately, have nothing up my sleeve.

### 1 The Global vs. LocalTeX PrefPanels

MacTeX installs a copy of TeX Live owned by root in `/usr/local/texlive`. It also installs a small data structure by Gerben Wierda and Jérôme Laurens in `/Library/TeX`, describing the distribution.

Each year’s TeX Live distribution is in a folder named by date in `/usr/local/texlive`. Users can keep old distributions around, in case a new distribution breaks something crucial. We install a Preference Pane for Apple’s System Preferences, allowing users to switch between distributions. A switch changes all GUI apps to use the selected distribution and also changes the command line so command line programs use it.

The PrefPane we install selects one distribution for all users and requires root access. I’m going to argue that we should have created a Local PrefPane instead, so each user could choose their own default TeX distribution and make this selection without root access. That’s how *programs* work on the Macintosh. Programs live in `/Applications` and are accessed by all users. But each user has personal Preference settings in `~/Library/Preferences` for these applications. One user’s default Word font might be Times Roman, while another’s might be Helvetica Neue.

The LocalTeX PrefPane shown below is such a Pane; it constitutes my schlocky gift. It can be

installed locally for one user or globally for all users, but it makes independent choices for each user and does not require a password. This Pane does not change any data created by the Global PrefPane, so it can be used together with the Global Pane, or when the Global Pane is completely missing.

The first item in the distribution list is always “Use Global Preference Pane”. Selecting this item activates the Global Pane for the current user. The next items are distributions with TeXDist data structures, so an individual user can select a different default than the one chosen by the Global Pane.

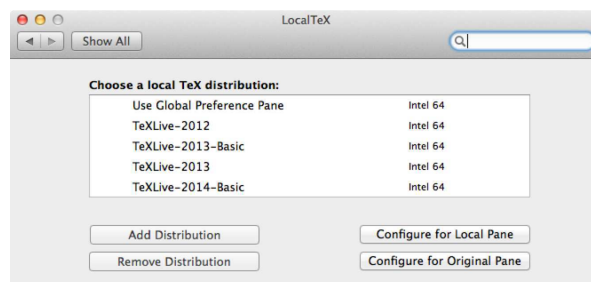


Figure 1: Local PrefPane choices

Scrolling down in the list of distributions in the Pane, we see that the LocalTeX pane can define and select distributions on external disks, or distributions installed in a user’s home directory. Although MacTeX cannot install TeX in such locations, the native TeX Live install script can.

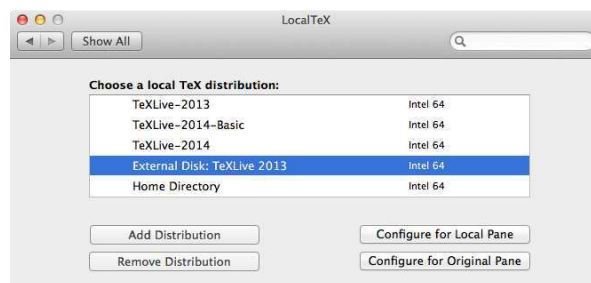


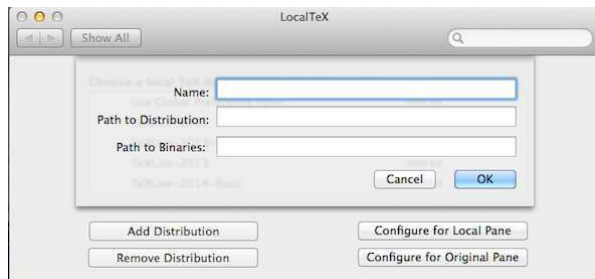
Figure 2: Local PrefPane supports external disks

Students may find this ability useful when they use a university-owned machine and don’t have root access. They can easily install TeX Live on a thumb drive, carry it with them, and have access to TeX in all locations.

The LocalTeX pane only shows distributions that are currently available, so if a thumb drive is removed, its distribution is no longer listed. Inserting the drive causes LocalTeX to list it again.

The “Add Distribution” button is used to inform the LocalTeX pane about TeX distributions without a TeXDist structure. It brings up a panel

shown below (fig. 3). The “Name” field can be any desired name, since it will only appear in the LocalTeX pane. The “Path to Distribution” and “Path to Binaries” fields can be filled in by dragging appropriate locations to the dialog.



**Figure 3:** Local PrefPane: adding distributions

The “Remove Distribution” button produces a list of extra distributions which can be removed one-by-one from those listed by the panel. Only distributions without a TeXDist data structure can be removed.

## 2 Installing and configuring the LocalTeX Pane

The LocalTeX Pane can be obtained from <http://pages.uoregon.edu/koch/LocalTeX.zip>. Then, installing the LocalTeX pane is easy: Find and double click `LocalTeX.prefPane`. This brings up a dialog offering to install the Pane for all users or for only one user. Choose “only one user” and the Pane is installed for the current user without requiring a password. Or choose “all users” and the Pane is installed for everyone, but acts as a local pane for these users; installing this way requires a password.

After the Pane is installed, push the button “Configure for Local Pane” on the right. This reconfigures TeXShop, TeX Live Utility, and BibDesk to use the new Pane. It also reconfigures the shell for *some* users, namely those whose home directory contains none of the three “hidden” files `.bash_profile`, `.bash_login`, and `.profile`. Other shell users can read the Local Pane documentation.

To return to the Global Pane and stop using the LocalTeX Pane, push “Configure for Original Pane” to reconfigure TeXShop, TeX Live Utility, and BibDesk.

## 3 How does the LocalTeX Pane work?

The LocalTeX Pane creates three symbolic links in `~/Library/TeX/LocalTeX`:

- `texroot` → directory of the default distribution
- `texbin` → binaries of the default distribution

- `texdist` → TeXDist structure for the default distribution, if it exists

GUI applications should be configured to look for TeX binaries in `~/Library/TeX/LocalTeX/texbin` rather than in `/usr/texbin`, the corresponding link for the Global pane. This is done automatically by the “Configure for Local Pane” button for TeXShop, TeX Live Utility, and BibDesk. Reconfigure other applications by hand. Many applications require a full path rather than one containing a tilde.

## 4 No system changes needed

Wierda and Laurens carefully selected the location for the link `/usr/texbin`, arguing that Apple would probably not remove this link. That reasoning turned out to be wrong, and users who upgrade OS X often find that they can no longer typeset even though their TeX distribution remains, because the link is gone. The location `~/Library` is not likely to present this problem because third party programs use it.

Creating Preference Panes with root access requires dealing with Apple’s often-changing security framework. The Local Pane is immune to security concerns. It currently runs on Yosemite betas. It requires Mountain Lion and above, since it uses Apple’s newer ARC memory protection scheme.

## 5 Removing everything

If you install the LocalTeX Pane and decide that you don’t want it, here is how to remove absolutely every trace from your computer.

- Using the Local Pane, push the “Configure for Original Pane” button to reconfigure TeXShop, TeX Live Utility, and BibDesk. If you configured other apps, return them to their original configuration.
- Move `LocalTeX.prefPane` from `~/Library/PreferencePanes` to the trash.
- Move the folder `LocalTeX` from `~/Library/TeX/` to the trash.
- Modify your shell startup script to change `~/Library/TeX/LocalTeX/texbin` back to `/usr/texbin`.
- The LocalTeX PrefPane stores its local data in the defaults system of OS X. To remove this data, type the following in Terminal:  

```
defaults remove \
    com.apple.systempreferences \
    localTeXExtrasData
```

## 6 LocalTeX and MacTeX

Will the LocalTeX preference pane be in a future edition of MacTeX? No. A choice between two Preference Panes would confuse most users. Moreover,



it is easy to configure the shell automatically for the global pane, but user intervention is required to do this for the Local Pane.

## 7 MacTeX design philosophy

Now I'll switch to the topic promised by the title. I work on the Macintosh in a small pond in the big TeX world. I wear two hats. I maintain MacTeX, the TeX install package for the Mac produced once a year by TUG. I also write, with collaborators, a GUI front end for TeX called TeXShop.

MacTeX is a “one button” package installing TeX, Ghostscript, and a few GUI applications. It presents a familiar interface for Mac users, asks no questions, and produces a completely configured installation. The installer was written by Jonathan Kew in an all-night programming session at the North Carolina TUG conference in 2005, and willed to me at breakfast the next day.

Jonathan's package installed a TeX distribution by Gerben Wierda, based on teTeX. But around this time, Thomas Esser abandoned teTeX and told his users to switch to TeX Live. Gerben produced a new distribution loosely based on TeX Live, which he announced at a TUG conference in Marrakesh in November of 2006. But at that same conference, he announced that he would immediately end support for the new distribution. This left us in a quandary and for several months it was unclear which distribution we would install. I had been attending TUG meetings since 2001, and oddly, in all that time, Karl Berry never asked me, “Why don't you Mac folks use TeX Live?” But as soon as we switched to it, we were happy and never looked back.

Here is the philosophy: MacTeX installs *a completely unmodified full version of TeX Live*. It is exactly the distribution used on GNU/Linux, Unix, and Windows (for those not using MiKTeX). We would never reach into the distribution and make configuration changes. When someone complains “my Mac collaborators cannot typeset my code” we get to respond vigorously, “Sir, it is *your* fault because Mac folks use standard TeX Live!”

Collaboration is common in research. Knuth worked very hard to make TeX produce the same results on all platforms. We have a responsibility to make TeX platform-independent. Open source forever!

(But a small voice: we are in Portland, Oregon, the home of Textures. Barry Smith rewrote the Pascal compiler for TeX, and then rewrote TeX to produce absolutely precise synchronization between source and output, and to support direct use of Macintosh fonts. His code was proprietary, not

open source. Textures users remember it with great passion. Every philosophy has a “yes, but ...”)

## 8 TeXShop design philosophy

Perhaps surprisingly, TeXShop has a very different design philosophy. A front end mediates between the paradigms of a computer platform and the paradigms of TeX. I'll argue that a GUI front end to TeX should rigorously follow the design standards of the particular platform it supports and should use the latest technology on that platform. This is difficult to achieve if the app supports many platforms.

To understand why, consider the following exchange from the TeX on OS X mailing list:

From: Warren Nagourney

I am using TeXShop 2.47 on a retina MBP and have noticed a slight tendency for the letters in the preview window to be slightly slanted from time to time. The slant is enough to make the text appear italicized, which is annoying.

From: Giovanni Dore

I think that this is not a problem of TeXShop. I use Skim and sometimes I have the same problem.

From: Victor Ivrii

Try to check if the same distortion appears in TeXworks and Adobe Reader: TeXShop and Skim are PDFKit based, while TW is poppler based and AR has an Adobe engine.

All three messages are from knowledgeable people active in the TeX on OS X list. As the third message states, TeXShop and Skim use Apple's PDFKit to display PDF files, while Adobe Acrobat Reader has its own PDF rendering code, and TeXworks uses poppler to render PDF. And indeed, TeXShop and Skim have a display problem but Acrobat Reader and TeXworks don't.

However, there is a missing ingredient here. The author of the original message has an Apple portable with a Retina display. TeXShop and Skim support the Retina display because they were written with Apple's Cocoa language. Acrobat Reader and TeXworks don't support the Retina display, so Apple runs them in “magnify by two” mode. The real problem is a bug in Apple's Cocoa Retina code, subsequently fixed. The bug also goes away if you turn off Retina support in TeXShop and Skim.

If you select “Get Info” in the Finder with a program selected, you get a panel of information about the program. That panel is shown below for TeXShop and Adobe Reader.

The key difference is the option to open in Low Resolution mode. This is selectable in TeXShop but is grayed out in Reader. That means that TeXShop

by default supports the Retina display, while Reader does not. In case of trouble, TeXShop can be converted to a mode in which it writes at normal resolution and the Mac magnifies by two, while Reader always runs in this magnify mode.

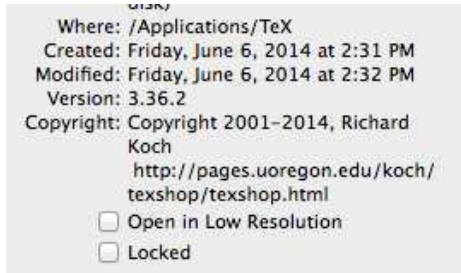


Figure 4: About TeXShop



Figure 5: About Adobe Reader

I had a very smart student who now works in the Portland software industry, so I boasted that TeXShop supported the Retina display from the start. But he was too smart, and without skipping a beat he said “yeah, and how many lines of code did that take?” The answer is zero.

There are many ways to write GUI apps on the Mac. *If an app is written in Cocoa, then it automatically supports the Retina display. Otherwise not.*

## 9 NeXT at Apple, 1997–2007

Many of you have read the book about Steve Jobs by Walter Isaacson. It is an interesting book, but has been criticized for getting the story of NeXT, and its role in Apple’s second act, wrong. I agree, and here’s a short version of that story from my perspective.

Apple bought NeXT in December of 1996, a sale that was finalized in February of 1997. Each May or June, Apple holds a Worldwide Developer Conference (WWDC). So in May of 1997, Apple had to give developers its strategy for using the NeXT operating system.

At the conference, Apple said that old Macintosh applications would continue to run in a sort of

purgatory called the Blue Box, but new applications needed to be written in Objective C using NeXT’s class library, then called OpenStep, later renamed Cocoa. Among commercial developers, the announcement went over like a lead balloon, and Apple got no significant endorsement at the conference.

So in 1998, Steve Jobs announced a completely different strategy. He called this new model “Carbon” because, he said, “Carbon is the basis of all life.” Carbon programs were written in C and C++ using the old Macintosh API, except that about 10% of the calls were replaced by new equivalents because the original calls wouldn’t work on a modern multi-tasking operating system. This made it possible to start with an old Macintosh program, find the changed calls using an Apple-supplied script, revise them, and release the code on OS X. Apple immediately received endorsements from Microsoft, Adobe, Wolfram Research, and others.

Many Apple engineers proclaimed that Cocoa was only for prototyping. At the 2000 developer conference I attended, the Carbon sessions were held in the main auditorium packed with thousands of developers, while the Cocoa sessions were in a small converted church across the street, attended by 35 people who all seemed to know each other.

I attended WWDC regularly from 2003 to 2011, and this pattern continued for several years.

The situation began to change in 2005, when Apple switched to Intel processors. At WWDC, they told developers that moving a Cocoa app to Intel involved a 10 minute recompile, while Carbon transition would often take a month.

In 2006 the developer conference was postponed until August. At the conference, Apple gave developers a preliminary copy of Leopard, the next version of OS X, promising a release in March of 2007. A key feature of this release was full 64-bit support for all of Apple’s important APIs. A key slide of the keynote explained that “Leopard has full 64 bit support for Carbon and Cocoa”.

But by June of 2007, Leopard was still not out. Why not? In January of that year, Apple announced the iPhone, and Apple engineers were pulled from the Leopard team to finish the software. Outside developers couldn’t program the iPhone, so the 2007 conference was essentially a repeat of the 2006 version, with a keynote address using the same slides.

There was just one electric moment in 2007. Unfortunately, I completely missed its significance. When Jobs came to the slide promising “full 64 bit support for Carbon and Cocoa”, the slide had been changed to read “full 64 bit support for Cocoa”. Lots of developers noticed, and they mobbed Apple engi-

neers during the lunch which followed the keynote. It rapidly became clear that Carbon was deprecated. Apple work on it had ceased.

So by 2007 Apple had the courage, and the prowess, to kill Carbon and throw its support totally behind Cocoa. Behind the scenes, they knew that both the iPhone and the as-yet-unannounced iPad could only be programmed in Cocoa. From 2008 on, there have been no Carbon sessions at WWDC. Commercial developers were among the last to switch to Cocoa, and some of their apps are still in Carbon.

During these turbulent times I was oblivious to the drama. TeXShop was written in Cocoa because I owned a NeXT machine, but it remained a 32-bit application. Finally, a few months before Lion, I made the transition to 64 bits. What I didn't know was that dramatic changes were happening at Apple, and my 64-bit conversion was done in the nick of time.

## 10 Fragile base classes and 64 bits

An *object* is a self-contained collection of code and data. Its data is referenced by variables known as *instance variables* and its code is known as *methods* or *functions*. According to a common metaphor, an object oriented program contains many objects, which talk to each other through method calls, and act on these calls by processing the data in their instance variables. Cocoa programs are object oriented.

To see how this works in practice, consider the Cocoa object called *NSView*. Each *NSView* corresponds to a rectangular portion of a particular window. The view has an instance variable pointing to its window, a second instance variable giving the coordinates of its rectangular region, and so forth. Among the methods defined for an *NSView* is `drawRect`, which draws the view on the screen.

When a program uses *NSView*, the developer defines a *subclass* of the view with a name like *myNSView*. This subclass has all the instance variables and methods of *NSView*, plus any other instance variables and methods added by the programmer. But in addition, it can override the original methods of *NSView*. For instance, the `drawRect` method in *NSView* does nothing, but *myNSView* can override `drawRect` so that it draws, say, our conference logo. In this situation, we call *NSView* the *base class*, defined in Cocoa, and call *myNSView* a *subclass*, defined by the programmer.

The advantage of all this is that base classes typically come already connected up. Cocoa calls `drawRect` when the window first appears, when a covering window is moved out of the way, when a dialog box goes away, etc. Apple once gave developers

a t-shirt with the text “Don't call us; we'll call you”. The slogan means that the programmer's *myNSView* doesn't have to worry about when to draw because Cocoa will tell it when to draw. It just has to draw the logo when called.

The takeaway is easy: a Cocoa program runs cooperatively, with some tasks handled by the base classes in Cocoa and other tasks handled by subclasses defined by the programmer.

After object oriented programming appeared, programmers began to dream of a time when the system could be improved by just revising the base classes, without even recompiling the programs. You could install Mavericks, and suddenly say “wow, Word never did *that* before!”

Unfortunately, a barrier stood in the way of realizing this dream. The barrier was called “the fragile base class problem”: *when revising base classes, Apple was not allowed to add extra instance variables or extra methods to the base class*. This was a problem in Objective C, in C++, in Java, and elsewhere. The problem wasn't quite as bad in Objective C as elsewhere because that language allowed extra methods in base classes. But still: no extra instance variables.

When Apple added 64-bit libraries in the Leopard timeframe, they realized that they had a once-in-a-lifetime opportunity to fix this problem. Since there were no existing 64-bit applications, every 64-bit app would have to be compiled from scratch. So they took the opportunity to make changes to Objective C when run in 64 bits, including completely solving the fragile base class problem. Incidentally, they also made these changes in the iPhone even though it ran in 32 bits. So Objective C on the iPhone, iPad, and 64-bit Mac applications is a different beast than Objective C in 32-bit Mac applications.

After this, Apple rapidly increased hardware requirements for its operating systems. Snow Leopard required Intel processors, Lion required 64-bit processors, and Mountain Lion required machines running the kernel in 64 bits. I believe that the reason for these policies is not that 64-bit programs run faster, but instead that Apple can now use all the extra added properties of Objective C, including adding instance variables and methods to base classes.

## 11 Lion remembers window positions

Lion is the first Apple system to make real this great dream of improving programs by revising the base classes. Programs written in 64 bits with Cocoa got crucial added functionality for free, even without recompiling.

One of the standard requests for TeXShop was that it remember window sizes and positions when

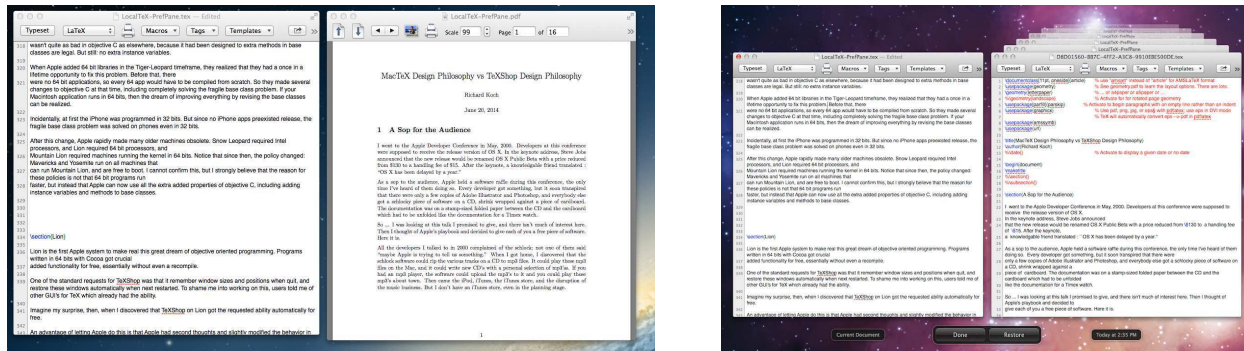


Figure 6: Editing (left) is normal, but past versions are available (right)

quit, and restore these windows automatically when next restarted. To shame me into working on this, users told me of other GUIs for  $\TeX$  which already had the ability.

Imagine my surprise, then, when I discovered that TeXShop on Lion got the requested ability automatically for free.

An advantage of letting Apple do this is that Apple had second thoughts and slightly modified the behavior in Mountain Lion and Mavericks. TeXShop inherited those changes for free. For instance, holding down the option key changes the Quit menu to “Quit and Close All Windows”, and if the shift key is down when opening a program, then old windows aren’t reopened. These tricks work in TeXShop just as in all other Cocoa applications.

## 12 Automatic file saving

Saving window positions is something I could have done myself if I weren’t lazy. But the second Lion feature is something I would never have tried on my own: automatic file saving.

Suppose you are using TeXShop in Lion, you have several source files open and have made changes in each. Suddenly you receive an emergency call and quit TeXShop. You won’t receive pesky dialogs asking you to save each file; instead, TeXShop immediately quits.

But the next time you open TeXShop, your (seemingly unsaved) edits will be there.

But wait — there’s more. TeXShop doesn’t just save when you quit. It saves every five minutes or so. If you live in a thunderstorm area with frequent power outages, no need to worry. When your computer starts up again, all changes you made will reappear.

“Gulp. Every five minutes the computer saves my 1000 page document?” Of course not. The program only saves changes, and in five minutes how much source did you change? In practice, you never notice the saving process.

“Whoa. When I send a document to someone else, are all those changes in the document? My reference letter says ‘works like a dog’, but originally I wrote ‘even a dog wouldn’t be interested in his line of research.’” Not to worry; files only contain the latest version.

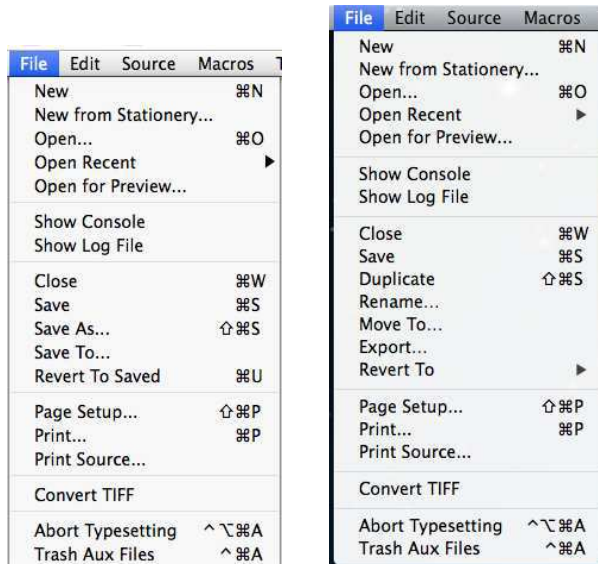
“But there are so many edge cases where this scheme could go wrong.” I absolutely agree. Indeed, I would never dare add automatic saving to TeXShop myself, or monkey around in any serious way with the file system. I have always dreaded getting a letter from a user claiming my program destroyed his only copy of a proof of the Riemann hypothesis. But Apple is doing this, with a thousand engineers testing the code. Their responsibility.

“But wait. Suppose I delete some material, type an experimental new sentence, and then decide not to keep it. In the old system, I just don’t save. But with automatic saving, the new stuff I don’t want may be part of the document.” The pictures at the top of this page (fig. 6) show why this is not a problem: the screenshot on the left shows a document you are reading while it was being edited; all looks normal. The right-hand image shows the effect of selecting the menu item Revert To → Browse All Versions, with a stack of old versions to work with.

As you see, this feature, called AutoSave, gives a Time Machine-like view of the document, and we can retreat to an earlier version, or copy a portion of an earlier version to the current document. Time Machine itself need not be running to get this. Any application with AutoSave activated gets it for free.

Apple has been refining the interface for AutoSave. It is intrusive on Lion, less intrusive on Mountain Lion, and less still on Mavericks. I couldn’t live without it. If your  $\TeX$  GUI has it, then it works the same as your other Mac applications.

AutoSave makes many changes under the hood. One of the most surprising is changes to program menus. The most controversial is the loss of a



**Figure 7:** File menu in source code (left) and as displayed by Mavericks (right)

“Save As...” menu item. I received many email messages demanding that I put it back. I replied that it was still present in my code, but Apple removes it while running the program. My correspondents found this explanation incomprehensible.

The truth is that Apple automatically modifies the program’s File menu when AutoSave is turned on. This is shown in fig. 7. On the left is the File menu as defined in current TeXShop source code. On the right is the actual menu as displayed in Mavericks. As can be seen, the middle section of the menu has been drastically altered.

After one email exchange on “Save As”, I wrote what I thought was a brilliant defense of Apple’s actions, telling my readers to “grow up and go with the flow”. The next day another user pointed out that “Save As...” had been restored by Apple in Mountain Lion. Sure enough, if you hold down the option key when accessing the File menu, “Duplicate” changes to “Save As”. Apparently the people on the mailing list were also writing Apple.

The main point I’m trying to make here is that for programmers who use Cocoa, solving the fragile base class problem allows Apple to make surprisingly many changes under the surface.

After all this, you probably want me to come clean. To implement AutoSave, how much code did I write? Well, Apple’s `NSDocument` object contains a function called `autoSavesInPlace`, which returns NO by default. In TeXShop I override it to return YES. That’s it. One line gives all of AutoSave.

Lots of collaborators help with TeXShop, pro-

viding features I haven’t mentioned. Today I just wanted to show what is made possible by adhering to Apple’s Cocoa standards.

TeXShop doesn’t adopt everything, of course. It isn’t in the Apple Store because working in a sandbox would limit its interaction with T<sub>E</sub>X Live and third party programs. It doesn’t allow you to store documents in the Cloud because the Cloud is only available to applications in the store. But when an addition makes sense, it will be adopted.

### 13 Automatic reference counting

One problem with object oriented programming is that a program can create thousands of objects as it runs. The program is supposed to throw away objects after it is done with them; if it doesn’t, then computer memory becomes clogged and the program becomes sluggish. On the other hand, objects can be passed around, so just because one part of the program is done with an object doesn’t mean that it isn’t used somewhere else. If an object is thrown away too soon, the program will crash when another part of the program tries to use the object.

There are three solutions. The first is to force programmers to manually handle memory management. That is how TeXShop worked until recently, and it is prone to errors that are hard to find.

The second method is called “garbage collection”. Apple introduced it in Leopard, but it didn’t work well on the iPhone.

Then as part of the enhancement of Objective C, Apple introduced Automatic Reference Counting, or ARC, the third memory management technique. In ARC, the compiler automatically adds the code to handle memory management, without the programmer needing to do anything. Since ARC does what a programmer would do managing memory manually, some files in a program can be compiled with ARC and some can be compiled without it.

This spring, I spent several weeks recompiling TeXShop with ARC, gradually working through the program file by file. The ARC code first appeared in TeXShop 3.34 and makes the program much more stable. A couple of remaining issues are solved in TeXShop 3.38, released at this conference, and this version ends the transition to ARC.

Adding ARC support is an example of extensive work with no immediate gain; no interface changes are visible. But it is essential work if the program is to survive for the long run.

◇ Richard Koch  
<http://pages.uoregon.edu/koch>

## **T<sub>E</sub>X Live Utility: A slightly-shiny Mac interface for T<sub>E</sub>X Live Manager (tlmgr)**

Adam R. Maxwell

### **Abstract**

T<sub>E</sub>X Live Utility is a Mac OS X graphical user interface for the T<sub>E</sub>X Live Manager command-line tool, `tlmgr`. I'll discuss the goals of the program, several usage examples, and some of the tricky issues in wrapping a tool in order to make it accessible to graphical user interface users. Many of these issues (e.g., error handling and selection of features to expose) are not platform-specific, so could also be of interest to non-Mac users.

### **1 Introduction**

The T<sub>E</sub>X Live Manager was introduced in T<sub>E</sub>X Live 2008 [4]. It allows users to manage a T<sub>E</sub>X Live installation, giving easy access to updates from CTAN (the Comprehensive T<sub>E</sub>X Archive Network) for various packages, and allowing global configuration of options such as paper size [2]. It has a graphical user interface (GUI) mode, based on the cross-platform Perl/Tk toolkit, as well as a command-line tool, `tlmgr`.

Although Mac users usually prefer a GUI over a command-line program, the initial response to the Perl/Tk interface ranged from tepid to antagonistic. In general, users felt that it was not “Mac-like”, and that it exposed more features than users commonly needed. Following a discussion on the MacT<sub>E</sub>X mailing list, initiated by Jérôme Laurens in October 2008, providing a native GUI for Mac OS X seemed useful.

The benefit of a native GUI is that it can be instantly familiar to a user of a particular platform, and is less likely to require installation of 3rd party libraries such as Perl/Tk in order to work. Ideally, this would serve to reduce complaints and pleas for help from Mac users to the T<sub>E</sub>X Live team, thereby insulating them from a vocal and occasionally obnoxious subgroup.<sup>1</sup>

I started work on a trivial program on 6 December 2008, and had an alpha release ready for the MacT<sub>E</sub>X working group the next day. Whatever your opinion of Apple, their Cocoa frameworks allow rapid development of software by doing much of the work for the developer. After several iterations

<sup>1</sup> My fellow Mac users may resent this generalization. However, it is not uncommon to receive reports from users saying “You need 25 pixels on the bottom of this window, and you have 23. Also, your text field in the preferences settings is 1 pixel misaligned at the right edge. See page thus-and-such of Apple’s *Human Interface Guide*.” I admit that I have sent such reports to Apple, regarding their own software.

and feedback on feature requirements, a beta version was announced on 30 December to the “T<sub>E</sub>X on Mac OS X mailing list.”<sup>2</sup> It was released under the new BSD license, and initially hosted on Google Code at <http://mactlmgr.googlecode.com>. As Google no longer allows hosting of binary downloads, as of August 2014 the project is now maintained on GitHub at <http://github.com/amaxwell/tlutility>.

### **2 Design**

The initial list of features was fairly minimal, and included:

- List/install updates
- List installed packages
- Reinstall `tlmgr` itself
- Change paper size

Over time, various other features have been added, but the most frequently used is the update feature, as far as I know. T<sub>E</sub>X users on Mac OS X tend to be compulsive updaters and early adopters, for better or for worse, often updating once a week or even more frequently. To the T<sub>E</sub>X Live team’s credit, they have done an excellent job at providing a reliable infrastructure for this purpose.

The overall design goals for T<sub>E</sub>X Live Utility have always been to:

1. Expose only the most common tasks
2. Give users a consistent (Mac-like) interface
3. Use `tlmgr`, instead of reimplementing it
4. Give useful feedback for errors
5. Do not require command-line interaction
6. Avoid blocking the GUI with long-running operations

The architecture of T<sub>E</sub>X Live Utility has remained largely consistent with the original releases. I could claim this is due to a great design, but it’s also due to a certain amount of inertia on my part; in other words, I’m too lazy to redesign it. Apple recommends a Model-View-Controller architecture ([1] gives a brief introduction), with logic for these general tasks split up into separate objects. It uses the Cocoa frameworks, and is written in Objective-C (and plain C, as Objective-C is a superset of the C language). Since MacT<sub>E</sub>X<sup>3</sup> requires administrative privileges to install, T<sub>E</sub>X Live Utility also requires administrative privileges to update and change various options. This is handled via a separate command-line tool called `tlu_ipctask`, which executes `tlmgr` with root privileges and passes its standard output and

<sup>2</sup> <https://email.esm.psu.edu/mailman/listinfo/macosex-tex>

<sup>3</sup> <http://www.tug.org/mactex>

Package	Status	Installed	Remote	Size
tcolorbox	Update available	34725 (3.11)	34818 (3.12)	3.1 MB
texlive-scripts	Updating...	34740	34889	84.8 KB
xetexko	Update available	34321 (2.7)	34864 (2.8)	256.8 KB
yathesis	Update available	34780 (0.99j)	34842 (0.99j)	1.3 MB
assocnt	Not installed		34934	221.1 KB
breqn	Not installed		34867	1.1 MB
chemformula	Not installed		34898	951.3 KB
chemgreek	Not installed		34896	539.7 KB
clrscode3e	Not installed		34887	5.9 KB
ghsystem	Not installed		34925	2.4 MB
komacv	Not installed		34906	461.6 KB
luatodonotes	Not installed		34908	219.3 KB
mathtools	Not installed		34868	1.0 MB
mugsthesis	Not installed		34878	359.4 KB
collection-latexextra	Update available	34735	34934	4.3 KB
collection-latexrecomm.	Update available	33855	34868	602.0 B
collection-luatex	Update available	33467	34908	546.0 B
collection-publishers	Update available	34737	34878	962.0 B

**Figure 1:** The TeX Live Utility main window, with packages listed for update. Note that the uppermost package will be automatically removed, as it has been removed on the server.

standard error back to TeX Live Utility over a Distributed Objects communications channel.<sup>4</sup>

The first and last items of the design goals have driven most of the decisions. Each `tlmgr` command that is invoked by TeX Live Utility is encapsulated in an `NSOperation` subclass. An `NSOperation` is an object which can be enqueued for asynchronous execution on a separate thread. This avoids blocking the GUI thread (“main thread”, in Cocoa parlance) while waiting for a `tlmgr` command to finish executing. Very little data is shared between threads, in order to avoid locking and reduce complexity.

### 3 Usage examples

The main window of TeX Live Utility is tabbed, and the present iteration of the GUI is intended to give a web browser-style interface to the repository.

#### 3.1 Updates

Figure 1 shows the main window and packages listed for update; this tab is an interface for `tlmgr update --list` and allows you to run `tlmgr update foo0 foo1...fooN` for specific packages, or `tlmgr update --all` to update all packages.

#### 3.2 Packages

Figure 2 shows the second tab of the main window, which lists all available packages; this tab is an interface for `tlmgr list`. When a network connection is not available, TeX Live Utility runs `tlmgr list --only-installed`. Actions available from this tab

<sup>4</sup> <https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/DistObjects/DistObjects.html>

Package	Description	Status
bguaq	Improved quantifier stroke for Begriffsschrift...	Installed
bhxcexam	An exam class designed for Mathematics Teachers...	Installed
bib-fr	French translation of classical BibTeX styles	Installed
bibarts	"Arts"-style bibliographical information.	Installed
▼ biber	A BibTeX replacement for users of biblatex.	Mixed
amd64-freebsd	amd64-freebsd files of biber	Not installed
i386-cygwin	i386-cygwin files of biber	Not installed
i386-freebsd	i386-freebsd files of biber	Not installed
i386-linux	i386-linux files of biber	Not installed
i386-solaris	i386-solaris files of biber	Not installed
universal-da...	universal-darwin files of biber	Installed
win32	win32 files of biber	Not installed
x86_64-darwin	x86_64-darwin files of biber	Installed
x86_64-linux	x86_64-linux files of biber	Not installed
x86_64-solar...	x86_64-solaris files of biber	Not installed
► bibexport	Extract a BibTeX file based on a .aux file.	Mixed
bibhtml	BibTeX support for HTML files.	Installed
biblatex	Bibliographies in LaTeX using BibTeX for sortin...	Installed

**Figure 2:** The TeX Live Utility main window, with packages listed. Note that binaries for various architectures are shown via a disclosure triangle.

Package	Date
Restore 27,227	Aug 9, 2014, 16:33:54
lua <sup>2</sup> atex-math	
Restore 31,389	Aug 2, 2014, 11:23:01
lua <sup>2</sup> libs	
Restore 33,861	Aug 2, 2014, 11:23:04
lua <sup>2</sup> mplib	
Restore 34,021	Aug 2, 2014, 11:23:07
lua <sup>2</sup> otfload	
Restore 34,131	Aug 2, 2014, 11:23:11
Restore 34,726	Aug 9, 2014, 16:49:06
lua <sup>2</sup> otfload.universal-darwin	
Restore 30,313	Aug 2, 2014, 11:23:09
lua <sup>2</sup> otfload.x86_64-darwin	
Restore 30,313	Aug 2, 2014, 11:23:10

**Figure 3:** The TeX Live Utility main window, with backups listed.

include install, reinstall, and removal of specific packages. Forcible removal can be accomplished by holding down the option key and choosing the “Remove Selected Packages” menu item.

#### 3.3 Backups

The Backups tab shown in Figure 3 gives a listing of available backups, from `tlmgr restore`. This allows the user to select a previous revision of an updated package...one of the most useful features of TeX Live Manager! The default setting for backups can be configured via an options sheet, and I recommend that users keep at least 1–2 versions of packages. Updates from CTAN can occasionally break packages, as they’re not tested at all.

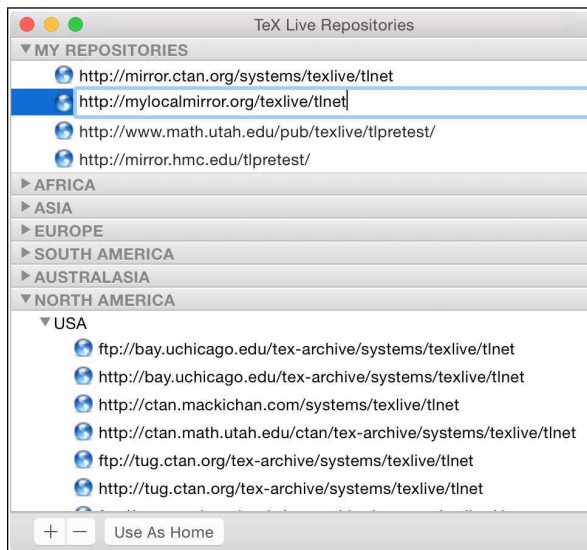


Figure 4: The TeX Live Utility repository interface, showing potential hosts.

### 3.4 Repositories

Managing and interacting with repositories is one of the more tedious parts of the program. In general, the CTAN multiplexor<sup>5</sup> works quite well, and is the default repository. It automatically selects a “good” server for your geographic location, and typically provides servers that are up-to-date. However, arbitrary servers often have different sync states with their master CTAN node, so if you run `tlmgr update --list` your next call to `tlmgr update --all` may use a different server and update a different set of packages. For this reason, TeX Live Utility uses the same server for updates as it does for listing them.

Users can also set a default repository, based on the available CTAN mirrors at the time TeX Live Utility was released.<sup>6</sup> Figure 4 shows the interface for choosing a repository; this is analogous to the “Bookmarks” interface in a web browser, and supports drag-and-drop to/from browsers. Arbitrary repositories can also be added, in the event that you have a private mirror or want to connect to an add-on repository such as `http://tlcontrib.metatex.org`.

Within the TeX Live Utility main window, you can choose a specific mirror by typing part of its name or location (e.g., `usa`) in the location bar, as shown in Figure 5. The location bar also offers standard “Reload” and “Cancel” buttons, and draws a progress bar during package installation (a user’s suggestion, inspired by web browsers such as Safari).

<sup>5</sup> `http://mirror.ctan.org/`

<sup>6</sup> This list of mirrors is developed by parsing `http://www.tex.ac.uk/tex-archive/CTAN/sites` using a Python script.

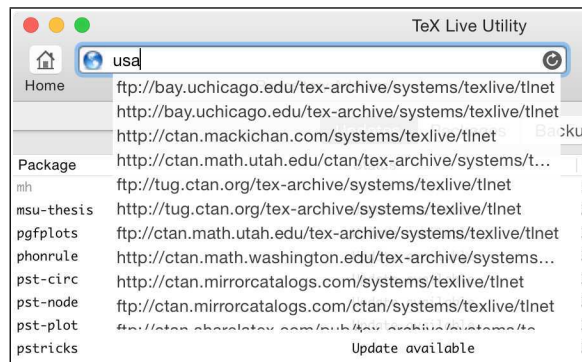


Figure 5: The TeX Live Utility address bar, showing the autocompletion feature.

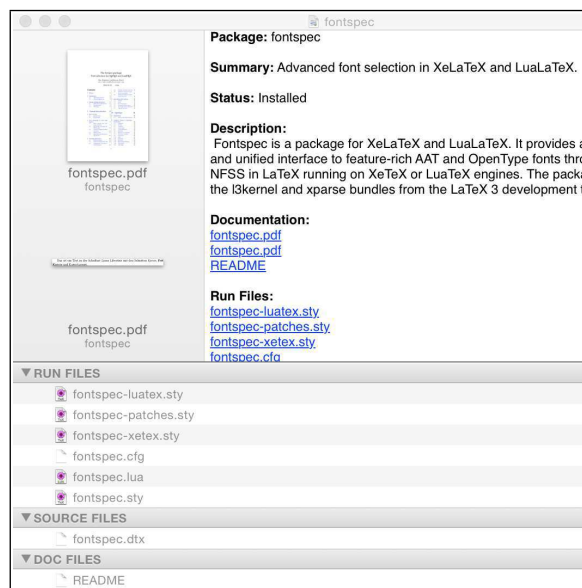


Figure 6: The TeX Live Utility Info panel, showing documentation and various associated files for a given package, along with a short description from the TeX Live Manager database.

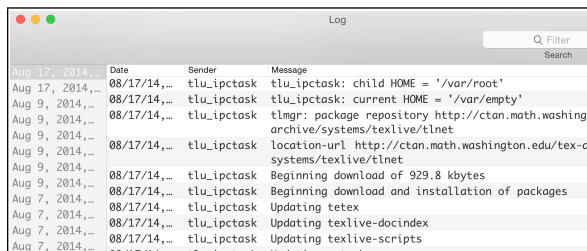
### 3.5 Info

Double-clicking on a given package in one of the main listings will show information for that package, along with the files associated with it in the TeX Live Manager database and any documentation found by `texdoc`. This is shown in Figure 6. For more advanced users, it’s easier just to type `texdoc fontspec` than to launch TeX Live Utility just for this task. However, it’s convenient if you’re browsing the package list and wonder what is (being) installed on your system.

### 3.6 Logging

I attempted to hide the log window shown in Figure 7 during a UI redesign, but that was not possible. It





**Figure 7:** The TeX Live Utility Log window, showing progress and debugging output.

turns out to be a critical feature, given that problems will occur sooner or later during an update, whether due to a problem with the TeX Live infrastructure or a bug in TeX Live Utility itself. Log sessions are saved, and you can select a previous session from the list on the left. Unfortunately, there’s a significant amount of clutter in the list, as TeX Live Utility logs many operations internally; this makes it less generally useful to users, who can easily mistake routine diagnostics for an actual problem.

### 3.7 Miscellaneous

TeX Live Utility also provides a means to schedule update checking, which ties into the Notification Center on Mac OS X 10.8 and later, and provides an alert dialog on 10.7 and earlier. The update notice lets you start TeX Live Utility and begin the update process, without having to remember to manually launch it and check for updates.

Various preferences can also be set, and there is a nascent feature for installation using the TeX Live network install script. This allows a user to install in an alternate location, and is mainly intended for users who do not have admin privileges. The TeX Live Utility Help menu describes how to enable and use it.

TeX Live Utility attempts to use the OS proxy configuration, and sets the necessary environment variables to work with `tlmgr` and other tools that it uses. This is one of the more complicated features from a code perspective (and one of the more fragile).

Obsolete versions of TeX Live can also be managed. TeX Live Utility will detect whether your installed version is older than the one on the remote repository, and switch to a historical archive as needed.

By holding down the option key, the menu item for reinstalling TeX Live Manager can also install it from the `tlcritical` repository, where test versions of TeX Live Manager are uploaded. This allows Mac users to easily test the latest version, and provide feedback to the TeX Live development team. Re-

verting to the current official version of TeX Live Manager is straightforward, via the same process without the option key.

A Spotlight importer for DVI files is included with TeX Live Utility, as is a Quick Look plugin for DVI files. These aren’t strictly related to TeX Live Utility’s core functionality, but it’s a convenient way to distribute them.

## 4 Pitfalls

This section contains a mixture of description, complaints, and advice for anyone contemplating a similar exercise, perhaps on another platform. Determining the intent of each piece is left as an exercise for the reader.

### 4.1 Error handling

Error handling is the biggest challenge in working with `tlmgr`; *viz.*, trying to communicate comprehensible errors to the user. For example, if a call to `updmap-sys` fails as part of the overall update process, there is no way for TeX Live Utility to know which command failed, or give the user any helpful instruction for resolving the problem. The standard error output from `tlmgr` contains the necessary information, but parsing it for individual errors is not a practical solution; each tool such as `fmtutil` or `updmap-sys` has its own error message style, and they are designed to be read by humans rather than parsed by a machine.

### 4.2 Asynchronous processing

An update can be hundreds of megabytes, especially if you update infrequently. This justifies the need for an asynchronous process model, in my opinion, although that adds complexity. However, running multiple operations at once could be problematic; TeX Live Utility doesn’t allow you to do a restore while doing an update, for instance, although you can show the Info panel for various packages during an update.

### 4.3 TeX Live Manager updates

The original TeX Live Manager in 2008 had a bug whereby it would delete itself during an update, and at Karl Berry’s suggestion, I modified TeX Live Utility to download and execute the standalone shell script updater from CTAN. This has turned out to be a good thing in the long run, as it also allows easy testing of the `tlcritical` version, and recovery from future installer problems is trivial.

### 4.4 Environment variables

Environment variables have caused the most painful support issues. Some users alter their `PATH`, `TEX*`, or

PYTHON\* environment variables using various obscure and poorly documented techniques.<sup>7</sup> Unfortunately, they typically forget having done it, so T<sub>E</sub>X Live Utility checks for these settings, logs a mildly rude warning message, and unsets their environment variables in its process space. Another clever user set his `umask` to 077 using these techniques, in the name of security. This meant that T<sub>E</sub>X Live Utility could not install updates, even as root, and was extremely puzzling.

## 5 Future plans

Future development plans for T<sub>E</sub>X Live Utility are limited at this time, since it works pretty well, and has only a single, very lazy, developer. However, a few things are in the works, presented here in probable order of appearance.

### 5.1 BasicT<sub>E</sub>X documentation

The BasicT<sub>E</sub>X distribution provided by Dick Koch is a small subset of T<sub>E</sub>X Live, provided for users with limited bandwidth or storage. One of the ways it saves space is by not including documentation, but some users have requested the capability to install documentation for packages on a case-by-case basis.

### 5.2 Interface updates

A few cosmetic problems exist on Mac OS X 10.9 and later; in fact, it's surprising that users have not yet pointed out that the progress bar should be drawn differently. More importantly, I plan to add toolbar buttons for the update action once again.

### 5.3 Network install

The network installer (a hidden feature) needs to be refined, and options simplified so that it's more usable; manually editing cryptic shell script variable names, needed at present, does not meet T<sub>E</sub>X Live Utility's design goals! It should also integrate with the T<sub>E</sub>X Distribution structure on Mac OS X [3].

---

<sup>7</sup> This used to be via `~/MacOSX/environment.plist`, and can now be accomplished via other means. I strongly advise against this, as it affects all programs spawned by the `loginwindow`, but is overridden in the Terminal by shell init files. It can literally take days of email exchanges to sort these problems out.

## 5.4 Privileged code

Apple has deprecated the `AuthorizationExecuteWithPrivileges()` C function that T<sub>E</sub>X Live Utility uses to run tasks as root, and is currently recommending that developers use `launchd` to run their privileged process. This will require significant effort, mainly in rewriting the communication code between the two processes.

## 6 Acknowledgments

This program would not be as effective as it is without the testing and feedback of Bruno Voisin, Herb Schulz, Justin C. Walker, Dick Koch, Will Robertson, and other members of the MacT<sub>E</sub>X group. The icon was drawn by Jérôme Laurens, and I thank him for letting me use it for this project.

Thanks to the T<sub>E</sub>X Live team for being supportive, especially Karl Berry in suggesting which features to include (or not!). Norbert Preining added the `--machine-readable` option to `tlmgr`, so I could do away with my gruesome ad-hoc parsing code, and has made numerous other improvements as maintainer of T<sub>E</sub>X Live Manager. Hopefully T<sub>E</sub>X Live Utility makes their lives a bit easier, in that they have to deal with fewer obnoxious Mac users like me!

## References

- [1] Pavneet Arora. YAWN — A T<sub>E</sub>X-enabled workflow for project estimation. *TUGboat*, 33(2):196–198, 2012. <http://tug.org/TUGboat/tb33-2/tb104arora.pdf>.
- [2] Karl Berry, editor. *The T<sub>E</sub>X Live Guide—2014*, 2014. Available online: <http://tug.org/texlive/doc/texlive-en/texlive-en.pdf>.
- [3] Richard Koch. Support for multiple T<sub>E</sub>X distributions in i-Installer and MacT<sub>E</sub>X. *TUGboat*, 28(3):329–334, 2007. <http://tug.org/TUGboat/tb28-3/tb90koch.pdf>.
- [4] Norbert Preining. T<sub>E</sub>X Live 2008 and the T<sub>E</sub>X Live Manager. *ArsT<sub>E</sub>Xnica*, 6:67–75, 2008. <http://www.guitex.org/home/numero-6>.

◇ Adam R. Maxwell  
 Port Angeles, WA 98362  
 USA  
[amaxwell \(at\) mac dot com](mailto:amaxwell@macdotcom)  
<http://github.com/amaxwell/tlutility>

---

## On tracing the trip test with JSBox

Doug McKenna

### Abstract

A new  $\text{\TeX}$  language interpreter library (currently called `JSBox`) I've been writing and debugging can accurately execute and completely trace the `trip` test, including recursive expansion, error interrupts, alignment table processing, and more. Relying on the log file that `JSBox` creates during tracing, this article explains how `JSBox` differs from  $\text{\TeX}$  in tracing implementation and formatting philosophy, and reveals what's going on in a few of the many deliberately puzzling areas of the `trip` test.

### Introduction

As is well-known in the  $\text{\TeX}$  community, Knuth's self-described “diabolical” `trip` test helps validate any new or extended  $\text{\TeX}$  language interpreter. Absent some similar test, the `trip` test is necessary — though not quite sufficient — to guarantee that one's interpreter is faithful to the core  $\text{\TeX}$  language and the myriad lines of  $\text{\TeX}$  code that have been written and relied upon for the last three decades. Knuth deliberately designed his test to be very difficult for both a non-conforming  $\text{\TeX}$  interpreter and a human (conforming or not!) to understand.

The  $\text{\TeX}$  code in `trip.tex` creates 16 pages of nothing useful typographically. But processing the convoluted input invokes nearly all the language's primitive commands, and most of the code paths each one depends upon. The test also relies on a font metric file, `trip.tfm`, with absurd ligature and kern programs in it. Many boundary conditions, where most errors occur, are deliberately triggered.

By design, there is almost no commenting nor documentation on what the `trip` test does. Validation generally means finding no non-trivial differences between an output log file and a reference log file. But this means that validation can occur without really understanding what the test does.

Worse, regardless of the job that  $\text{\TeX}$  is performing,  $\text{\TeX}$  only traces a portion of what's going on under its hood anyway. All of which is to say that the job of creating a conforming  $\text{\TeX}$  language interpreter is not an easy or pretty one. Nonetheless, the `trip` test is invaluable in ferreting out bugs in any  $\text{\TeX}$  language interpreter's implementation.

### Tracing vs. hidden state

Basic user-interface theory teaches that modes foster human error. And all hidden state represents a mode of some kind. Hence, revealing hidden state —

such as inserting individual temporary `print` statements in one's code — is always a key component of debugging. The  $\text{\TeX}$  virtual machine and its quite complicated typesetting algorithms represent a great deal of hidden rules and state. So any  $\text{\TeX}$  language interpreter must be pre-infused with special `print` statements to reveal (if asked) what is happening. The purpose of tracing is not only to create a record of what the macro interpreter has done on the user's behalf, but also to reveal what the machine is *not doing* that a user thinks it should be doing, because of some mode-induced user error.

To save code space, in one place (its inner execution dispatch loop)  $\text{\TeX}$  generically traces the meaning of a primitive command that it is about to execute, on behalf of the upcoming snippet of code that implements that command. But because of this, there are still lots of holes in  $\text{\TeX}$ 's tracing that regularly cause user confusion. For instance,  $\text{\TeX}$  only traces the first in a sequence of characters (i.e., a word), because characters after the first are handled by a separate inner loop looking for ligatures. Unfortunately, this violates the user's idea of the world, not to mention the reasonable expectation that if tracing is good enough for one character it ought to be good for them all.

$\text{\TeX}$  was designed to output quite short lines, and to break longer lines at arbitrary points, without regard to content. This in turn means there is no indentation to indicate in the log file where any subordinate set of executed commands start and end. All of which makes reading log files unpleasant. Regardless, often there is ensuing recursive expansion that is not (or only partially) traced.

Other examples abound: for instance, a large amount of complex behind-the-scenes processing occurs during any `\halign` or `\valign` command, yet much of it remains hidden during tracing. Many group contexts are unlabeled, e.g., when typesetting math formulas. `\global` definitions are not traced well, due to the internal design of how the prefix is processed. The places where a file is not found prior to being found are important to know when things go wrong. That missing information is a constant source of confusion ever-addressed by questions on various  $\text{\TeX}$ -support web sites or mailing lists.

All of this hidden state represents a significant cognitive load on anyone reading a job's log file, because every lacuna violates the user's view of things, which is formed primarily by the sequence of commands and characters in his or her source code. There have been a variety of extensions made to  $\text{\TeX}$ 's tracing over the years, but as a relatively

recent user I find the results unsatisfactory: overly generic and/or still incomplete.

### Redesigning tracing

As a self-defensive tech-support measure, I wanted my  $\text{\TeX}$  language interpreter to be as communicative as possible to myself and any other user in explaining what it is doing, or not doing. So my goal has been to accomplish 100% tracing in a more complete and understandable format than what the classic  $\text{\TeX}$  (or  $\varepsilon\text{\TeX}$ ) engine does, without any significant hit on efficiency when not tracing. This means creating long-lined log files that no longer can be compared (e.g., with `diff`) to  $\text{\TeX}$ 's log files. The downside is that passing the ill-defined `trip` test becomes a tedious manual exercise, and either problematic or impossible, depending on what it means to be “the same”. I don't really care, because I'm willing to label this new interpreter with something other than “ $\text{\TeX}$ ”. The goal is to faithfully execute  $\text{\TeX}$  source code and get the same typeset results.

In the `JSBox` library, each class of related primitives is implemented by a subroutine that is responsible for tracing each variant's operation, using a common set of tracing utilities and formatting rules. The utilities include a stack of output staging buffers in which to construct lines of text. The bottom buffer in the stack is always used for tracing. Higher buffer levels can interrupt tracing in the service of constructing strings, error messages, or other formatting. The stack is usually one level deep, and almost never more than two levels deep.

The start of any line of output from a program is usually better-defined in both time and space than the line's end. This means that it is the responsibility of any tracing or other output code initially to flush the last tracing line, then to start a new line, and to never worry about terminating that line. Different code paths can append to the line as needed, without being responsible for knowing the line needs to end. This is not dissimilar to what  $\text{\TeX}$  does, but in `JSBox` we allow the end of the current text buffer to be trimmed prior to flushing. As we will see, interruptions can then be unambiguously formatted in a nice way. (It also allows `JSBox` to coalesce sequences of input characters so that all characters in a spacer-separated word are traced in one readable, delimited group of characters per trace, without doing any internal lookahead.)

Lines in a `JSBox` log file are not length-limited; the library can indent tracing to indicate different execution or nested group context stack levels. So every trace of a command or character(s) not only starts with a newline, but is then followed by inden-

tation representing execution, group nesting depth, or trace-continuation status. (If occasionally indentation gets excessive, it is pinned.) For nearly all normal execution I find the clarity to be worth it. I place a high value on vertical alignment and white space. So a log file is best viewed in a fixed-width programmer's font.

The name, not the meaning, of the command being executed is enclosed by a pair of matching braces, as in `{\indent}`. A sequence of letter or other characters is also collected and placed at the start of the trace line, e.g., `{xyzzy}`, even though each input character is processed one at a time as it is read. If there is any further information or commentary about the command, a colon follows, and a description of what's going on, any special meaning, or what any collected argument value is, is appended next. Extra commentary is placed inside a pair of matching brackets. Unicode characters are presented as is (converted to UTF-8) and usually with added commentary showing both a base 10 and a hex integer value (and even more information for math characters). If a large amount of information is needed, any number of extra lines can be used in a single trace. All information on subsequent lines is indented to the same position as the line's initial information, *after* the announcement of the name of the primitive being executed or the word of characters being appended.

Internally, every new (multi-line) trace is assigned an integer code that immediately increments to guarantee uniqueness. If there is any chance that tracing might be interrupted by expansion, recursion, paragraph/page building output, an error message issued, or any sub-system tracing (such as a macro stack frame popping while looking for the next token), then the partial trace's text line has `"..."` appended. Later tracing then checks to see if there was an interruption. If not, the `"..."` at the end of the still-unflushed trace text is erased, and further tracing information is appended to the current line. But if there was any sub-tracing, error messages, or other output, then the trace buffer with its trailing ellipsis was flushed, and we create a repeated trace continuation line that *starts* with an ellipsis and re-traces the command again, at its usual indentation level, so that newly collected information can be presented to the user.

This is a lot of work, but clarity, not efficiency, is the user's focus during tracing. The  $\text{\TeX}$  language's peculiarities make it really important to “go the distance” on this. Indeed, a significant portion of the `JSBox` library's code is devoted to tracing its own operation, in order to reveal hidden state.

**Tracing examples from `trip.tex`**

Consider the following line of nonsense code, from line 288 of `trip.tex`, but treated as line 1 here:

```
| \raise1pt\hbox{\special{\the\hangafter} } \penalty-10000
```

This is a single `\raise` command, operating on a horizontal box with “embraced” contents, followed by a space, and then a `\penalty` command with its trailing integer argument. The `TeX` log file tracing this would contain (again, changing line 288 to 1):

```
{\raise}
{entering hbox group (level 2) at line 1}
{restricted horizontal mode: \special}
{blank space }
{end-group character }}
{leaving hbox group (level 2) entered at line 1}
{horizontal mode: blank space }
{\penalty}
```

This is concise, but unfortunately too concise. Important information remains confusingly hidden.

Here, on the other hand, is how `JSBox` traces the same code:

```
|
|   {\raise}: by 1.0pt ...
|       {\hbox}
| >>> restricted horizontal mode
|       {}: entering \hbox group [level 2 at line 1]
|           {\special} ...
|               {\the} ...
|                   {\hangafter}: -12 [parameter]
|                   ... {\the}: Pushing {-12} [3 chars] onto input
|                   ... {\special}: {-12} [appending external command]
|                       { } : appending font \rip's inter-word glue [4.0 plus 2.0 minus 1.0]
|                       {} : leaving \hbox group [level 2 at line 1]
| .. {\raise}: appending hbox : [id=581] (0.0 + 0.0) x 4.0 [rigid] [2 items] shifted by -1.0pt [upward]
| >> horizontal mode
|     { } : appending font \rip's inter-word glue [4.0 plus 2.0 minus 1.0]
|     {\penalty}: -10000 [always] [appending to horizontal list]
```

It's twice as many (longer) lines of tracing (and I've asked this journal's editor not to reformat the above to fit in a narrow column), which helps the reader discern pretty much everything that's gone on. So let's explain some of the design decisions that went into formatting the foregoing.

To start, the final (usually first) line of any trace that is interrupted ends in an ellipsis. And every trace so interrupted is re-traced (using the same indentation) after the interruption ends, showing the final information collected by the command's end. In the foregoing, the `\raise`, `\special`, and `\the` commands are interrupted and therefore re-traced using the latest state and argument information.

Every trace contains the command or a character or set of characters, enclosed in braces. The rule I try to follow is that, whatever the item is, it should

be the same as what is in the user's original source code (`TeX`'s tracing violates this in several ways).

Unlike `TeX`, we don't integrate changes to the layout's current typesetting mode as a modifier to the brace-enclosed item being executed or appended to the layout. Each such change is traced on its own line, with an indication (e.g., "`>>>`") of semantic nest stack depth, which is independent of execution stack or group context indentation. More importantly, this makes it easy to find, or easy to ignore, the state of the layout mode. And it doesn't violate the rule that the first and only thing executing is what's enclosed by a pair of braces at the start of each trace. That information is more important, and needed to synchronize source code with log file.

We strive to place a bracketed hint mentioning the internal meaning of numeric values. A “penalty” of `-10000` is really an (infinite) incentive to “always” do something. A negative `\raise` is upward on the page (unlike, say, `PostScript`, `TeX`'s page coordinate system has its origin near the page top).

When a primitive appends a new item to the current layout list, it says so. For instance, when a spacer is processed in a horizontal mode, it appends a particular glue value, which is announced. And the `\raise` command, after all forward-looking recursive expansion is finished, is left with a horizontal box with two items inside that is appended to the current layout list after shifting. Another

hint reminds the user that this particular box cannot stretch or shrink. The library assigns a unique ID number to each box it constructs—searching a log file for a particular box is thus much easier.

There are other subtle formatting issues going on, in the service of maintaining vertical alignment of information. For example, the ellipsis in front of the re-traced `\raise` command is truncated by one dot, because there's one column too few in the indentation area in which to place three dots followed by a space (here, standard indentation is three columns per level).

Another difference from  $\TeX$  is that nodes in a layout list (boxes, ligatures, kerns, penalties, glue, math, output nodes, etc.) are always described with a four-character identifier that never begins with a backslash. This regularizes vertical alignment of layout list dumps, which  $\TeX$  minimally indents with visually noisy sequences of dots, sometimes followed by pseudo-commands (e.g., `\glue`) that don't occur in your source code. So the description of the box being appended by the `\raise` command is not `\hbox`. It is `hbox`, followed by a colon. The former is a command, the latter a type. Notice also that in talking to the user, we avoid the internal implementation and graph-theoretical term *node*. The generic word *item* suffices and is more user-friendly.

The `trip` test essentially consists of two parts. The first executes if one's virtual machine is uninitialized. The second runs if the interpreter is initialized from a format file, `trip.fmt`, created when the `\dump` command is issued at the end of the first part. The second, more substantive, part is where the `trip` test does most of its work. Because I desire this interpreter to be able to avoid using format files, `JSBox` currently treats a `\dump` as a no-op. Fortunately, by virtue of the trickery in `trip.tex` (see the definition of `\next` on line 90), this merely results in executing both parts of the `trip` test in one run.

In the first (format-creating) clause, `trip.tex` turns tracing off. Only various error or other messages are issued. Because `JSBox` can be compiled to suppress all “turn tracing off” commands that arise from source code, we can trace all parts of `trip.tex` without adding extra trace commands at the start of the input file.

On line 2 of `trip.tex`, the very first primitive is an `\immediate` command, followed by a `\catcode` command. The former's use is not an error, but is deliberately incorrect and/or unnecessary. That's because `\immediate` modifies only output-related commands that follow it (the `\catcode` command is not subject to the immediate vs. delayed execution distinction). An interpreter created by `JSBox`

can be configured at run-time to comport with the constraints of a classic  $\TeX$ 82 interpreter, for which the `trip` test was designed. But `JSBox`'s client program can enable warnings for situations like this, because a command that does nothing when misused may still be worthy of the user's attention. So, the log file contains both the trace and the warning message:

```
{\immediate}: [ignored]
Context : "trip.tex" [Line 2]
Warning : Ignoring \immediate. It only modifies
          output-related commands (e.g., \write),
          not \catcode.
Line 2   : \immediate\catcode '{ = 1 \endlinechar=13
          ~~~~~
```

Trace lines are usually indented one level from the left margin, because most jobs start with the inclusion of a  $\TeX$  source file or memory string. This pushes an input stack frame, to which trace indentation pays attention. While some might consider this a waste of space, it has the salutary effect of making non-trace messages (errors, warnings, etc.) easier to pick up in the log file, where they start at the left margin. And the extra space leaves room to signify re-tracing with a (partial) ellipsis, as our earlier example shows.

Because `\immediate` can be completely traced prior to the warning message being issued, there is no need for any trailing ellipsis: a re-trace wouldn't be able to add any new information. But in case warnings are disabled, the trace still provides a hint to the user that the command is useless and ignored.

Subsequent lines of interpreter-generated messages like the above are indented. This makes visual parsing of the log file much easier. Furthermore, the message itself is not generic. It has been tailored to include mention of the command (or character) which rendered the `\immediate` worthy of flagging. Non-generic messages are more work to create, but they keep the user grounded, preventing bad assumptions. Again, the goal in debugging is to reveal as much hidden state as possible.

Unlike  $\TeX$ , the input line (or for longer lines, a portion thereof) is not broken into two pieces to implicitly show the position of the scanner when the message was issued. To do so violates the user's view of his or her source code, and thereby unnecessarily adds to a cognitive load at an inopportune time. So `JSBox`'s scanner maintains the starting and ending position of each item parsed on an input line, and preserves that information for the benefit of any

formal message reporters. In the case of writing the error to a fixed-width format log file, we underscore—as best we can in a fixed-width log file font—the command (or character) responsible for the message. This is sometimes difficult to do in a manner that doesn’t confuse the user, even though it might be internally accurate. Non-generic error messages can alleviate the problem somewhat.

Here is another traced snippet from line 4 at the start of `trip.tex` that illustrates more of JSBox’s tracing philosophy and formatting. This code temporarily changes the category code of the math formula shift character (`$`) inside a group context.

```
{\catcode}: '$ <- 3 [math shift] [no change]
{ }: entering simple group [level 1 at line 4]
  {\catcode}: '$ <- 13 [active] [was 3 = math shift]
{ }: leaving simple group [level 1 at line 4]
  restoring [mapping] \catcode of '$ to 3 [math shift]
{ } : [ignored in vertical mode]
```

In this case, it is now `TeX` that would trace the above using nearly twice as many (shorter) lines:

```
{\catcode}
{reassigning \catcode36=3}
{begin-group character { }
{entering simple group (level 2) at line 4}
{\catcode}
{changing \catcode36=3}
{into \catcode36=13}
{end-group character { }}
{restoring \catcode36=3}
{leaving simple group (level 2) entered at line 4}
{blank space }
```

I invariably can’t recall the implicit (hidden) meanings of numeric codes, such as 36 or 3. So when JSBox traces the `\catcode` command, adding commentary (i.e., `[math shift]`) on the syntactic meaning of the numerical argument 3 saves mental energy and prevents errors, and the character itself is used, not just its ASCII code 36. We also note actual value changes, but unlike `TeX`, we don’t use two more lines to accomplish this trace generically. And in the main trace we strive to inform the user of the new value *prior* to what the old value was, because the new value is what the user is nearly always interested in.

JSBox internally traffics in full 21-bit Unicode code point integer values, with all of them above the initial ASCII range initially classified as characters of type “other”. And any Unicode character (code point) can have a syntactic catcode assigned to it. For printable ASCII characters in the initial 7-bit plane, such as the `$` above, we don’t output the character’s integer code (for arbitrary Unicode, we

would). In extended mode (i.e., when not limited to just `TeX82` features) the JSBox library can handle non-UTF-8 Unicode using `~^uxxxx` or `~^Uxxxxxx` extended escape sequences to specify Unicode code points in hex, or via the usual `\char` command, which will take any 21-bit integer argument (but limited to 8-bit values in `TeX82` emulation mode).

Notice also that the information about popping a group context stack frame, and restoring any non-globally changed values, is properly traced solely by the recognition of the `}`. Restoration is indented to indicate its subordinate status to the closing brace, and there can be an arbitrarily long list of lines an-

nouncing each restoration. Because it’s hard to distinguish between various control sequence names, a hint as to whether a name is a parameter, register, mapping, etc., is also inserted for good measure.

Finally, by announcing that a space in vertical mode is ignored, there is no need to label the space character as a blank space; `{ }`: does a perfectly fine and unambiguous job.

Here is another example, from line 10 of `trip.tex`:

```
| \defaultthyphenchar='-
```

JSBox traces this as

```
| {\defaultthyphenchar}: - [was ^^@] ['- = "2D = 45]
```

The new character code for the parameter is shown, followed by bracketed commentary on the parameter’s previous value, which changed. Then further commentary on the integer and hex value of the new character is added in case it might be useful. Notice that the old value was 0, a null. Unlike `TeX`, JSBox strives never to write a null byte to a log file or to the terminal. Programs that display text files do not treat nulls uniformly. So unless it’s writing a data file, JSBox converts each null byte to a printable `^^@`.

The general philosophy I’m guided by is that too much information is a lesser evil than too little. For commands that expect character code point values, the numbers are there, but on the right, in commentary, where it can be easily ignored.

Consider lines 59–60 of `trip.tex`:

```
\def\weird#1{\csname\expandafter\gobble\string#1 \string\csname\endcsname}
\message{\the\output\weird\one on line \the\inputlineno}
```

$\epsilon$ -TeX's trace of this peculiar code would be

```
{\def}
{changing \weird=undefined}
{into \weird=macro:#1->\csname \expandafter \gobble \ETC.}
{blank space }
{\message}

\weird #1->\csname \expandafter \gobble \string #1 \string \csname \endcsname
#1<-\one
{\csname}
{\expandafter}
{\string}

\gobble #1->
#1<-\
{\string}
{changing \one \csname=undefined}
{into \one \csname=\relax}
\one \csname on line 60
{blank space }
```

whereas JSBox can trace the same input as follows:

```
1  {\def}: \weird#1->\csname \expandafter \gobble \string #1 \string
2          \csname \endcsname
3  { }: [ignored in vertical mode]
4  {\message} ...
5  {\the} ...
6  {\output}: [parameter] ->
7  ... {\the}: Pushing {} [token list] onto input
8
9  Calling \weird #1->\csname \expandafter \gobble \string #1 \string
10         \csname \endcsname
11         #1: \one
12         {\csname} ...
13         {\expandafter}: postponing {\gobble} until after expanding {\string}
14         {\string}: Pushing {\one} [4 chars] onto input
15
16         Calling \gobble #1->
17         #1: \
18         Returning from \gobble [empty body]
19         {\string}: Pushing {\csname} [7 chars] onto input
20     ... {\csname}: 11 characters collected
21         {\endcsname}: \one \csname constructed [= \relax [internal]] [{\one \csname} has a space in it]
22     Returning from \weird, resuming reading from file "trip.tex"
23     {\the} ...
24     {\inputlineno}: 60
25     ... {\the}: Pushing {\one \csname on line 60} [23 chars] onto input
26 .. {\message}: [to "trip.log" and terminal]
27
28 \one \csname on line 60
29
30 { }: [ignored in vertical mode]
```



In lines 1–2, we wrap a longer token list onto a new line without breaking any command name internally, and we continue the token list indented to the same column as it started on, just to the right of the `->`. A longer list will be truncated, but JSBox’s threshold is larger than  $\TeX$ ’s.

Like  $\TeX$ , we insert a blank line in front of each macro call, but unlike  $\TeX$ , we label each macro call with `Calling` to distinguish macros from primitives (or `\let`-created synonym names). Arguments collected are indented further—their collection is all part of the same trace of the macro call. We additionally can trace the end of the macro, when its stack frame gets popped, and announce later (at line 22) where the next input will come from (either a file, or a previously called macro in which execution was nested). In between, we’ve entered a new nested and indented command execution level.

At line 13, the often confusing `\expandafter` command is traced non-generically. At line 18, the `\gobble` macro expands to nothing, so we add a commentary hint saying `[empty body]`. At line 21, we specifically add commentary for any constructed control sequence name that (here deliberately) has a space in it, which can otherwise be very confusing.

Finally, we place a blank line on either side of any non-tracing text being output to the same destination (log file or terminal) that interleaves with traces. This sets the output off and makes it much easier to find by scanning down the page, especially as most such non-tracing, internally generated message output is not indented. (JSBox can insert an extra blank line automatically between any two different classes of text in log/terminal output.)

### Redesigning the `\show... commands`

Each of the  $\TeX$  language’s `\show...` commands formats and prints the value of some internal variable or list. But  $\TeX$  re-uses some of its error reporting machinery to do so, which I find confusing. For example, lines 29–30 of `trip.tex` seem to me to result in a hard-to-read formatting mess:

```
> \errorstopmode=\errorstopmode.
1.29 ...=256 \show\errorstopmode

> \rip .
<recently read> \font

1.30 \showthe\font
\showthe\pageshrink \showthe\pagegoal
> 0.0pt.
1.30 ...font \showthe\pageshrink
\showthe\pagegoal
```

There’s too much extraneous information, and it’s a cognitive load to be presented with the effect of the command *prior* to seeing the command causing that effect. So JSBox avoids displaying non-relevant parts of the input line and/or breaking it apart to show where the scanner is, puts cause and effect back in order, skips the unnecessary detail about what was `<recently read>`, and displays just the answer on its own line. Also, we label lines with `Line`, not `l.`, because of the time-honored principle that a lowercase `l` in a fixed-width font will invariably be confused with the digit `1`. A new trace of lines 29–30 “shows” the difference:

```
{\show}: \errorstopmode

Line 29 | \show \errorstopmode
\errorstopmode

{\showthe} ...
{\font}: \rip
... {\showthe}:

Line 30 | \showthe \font
\rip

{\showthe} ...
{\pageshrink}: 0.0pt
... {\showthe}:

Line 30 | \showthe \pageshrink
0.0pt
```

Without interleaved tracing, this would simply be:

```
Line 29 | \show \errorstopmode
\errorstopmode
Line 30 | \showthe \font
\rip
Line 30 | \showthe \pageshrink
0.0pt
```

The delimiters `>` and `.` are not used, because I find that they add more ambiguity/noise to  $\TeX$ ’s output than they resolve. Also, JSBox doesn’t insert blank lines between these similar `\show...` commands because it knows that each result fits on one line. For other `\show...` commands that result in multiline answers, such as `\showlists` or the extended JSBox `\showfont` command (which shows the metrics and all other data of an entire loaded font), blank lines are used to help the user understand where the command’s group of output lines ends.

## Tracing alignments

Perhaps the most complicated primitive commands in the  $\TeX$  language are `\halign` and `\valign`, each of which converts a one-dimensional stream of commands and text into a two-dimensional table on the page. Both commands work almost identically, by swapping horizontal rows with vertical columns. And they're recursive, since an element of a table's cell can be a sub-table. Material that looks like it might be executed is recorded, and material that looks like it might be recorded is executed. Expansion can occur. There are hidden contexts. The purpose is to allow the entire power of  $\TeX$  to be applied to any cell in a table. Knuth describes them as working almost magically.

Consider line 120 of `trip.tex`. It contains a curious empty table, as part of the `\output` routine for the page executed later at line 150:

```
\globaldefs1\halign{#\tabskip\lineskip\cr}
```

Among other things, this tests whether one has executed a `\tabskip` command, with attendant expansion and implicit global definition, in the alignment's preamble, rather than recording it into the preamble's token list(s), as would be nearly all other commands and characters.  $\TeX$ 82's trace of this, as taken from `trip.log`, is about as minimal as can be:

```
{\globaldefs}
{\halign}
```

$\epsilon$ - $\TeX$  does a little better with its tracing extensions turned fully on, but here's what's really going on under the hood, as traced by JSBox, using its interruption and indentation rules:

```
1      {\globaldefs}: 1 [was 0]
2      {\halign}: building successive rows, each containing entries in horizontally tabbed columns ...
3          {}: entering \halign group [level 3 at line 151]
4      ... {\halign}: [preamble] recording templates for each column ...
5          {}: end of prefix material for column 1's template; collecting suffix
6          {\tabskip}: [not recorded in template] ...
7              {\lineskip}: 0.0pt plus 40.0pt [parameter]
8      ... {\tabskip}: changing \tabskip [inter-column glue] to 0.0 plus 40.0 [global]
9          {\cr}: end of column 1's suffix and template
10     ... {\halign}: preamble has declared template for 1 column ...
11         \tabskip [= 0.0]
12         column 1: [no extra material to insert]
13         \tabskip = 0.0 plus 40.0
14         {column entry}: entering hidden alignment item group [level 4 at line 151]
15         {column entry}: leaving hidden alignment item group [level 4 at line 151]
16         {}: leaving \halign group [level 3 at line 151]
17     ... {\halign}: [done] appending 0 rows of aligned material
```

As `\halign` fires up (see line 2 above), exactly what is about to happen is announced, because there's

simply not enough information in the name of the command to disambiguate what the command does, should the user not be sure. As processing of input proceeds, the `\halign` will retrace itself three more times (with ellipses as appropriate), at the same indentation level, even though the execution stack level is changing up and down at the same time. Notice that every token in the source code is represented as the start of a full trace line or lines—tracing should not break the user's mental model of what's going on. Because there will be one last trace line announcing the final result of the command, we indent the opening `{` and closing `}` of the `\halign` group, to make it easier to scan down the log file looking for the start of the final trace line.

At line 4, the alignment's preamble starts being recorded. When the preamble material ends with the first `\cr`, we trace again, and then synopsize the template material for each column. Unlike  $\TeX$ 's opaque and overly mathematical *u*-part and *v*-part terminology, I've used the more descriptive and user-friendly terms *prefix* and *suffix*.

$\TeX$  processes each column's material inside a hidden group context, as if surrounded by a pair of braces, to prevent changes to registers and parameters from leaking to the following column's material. In this example, there is no material at all, so after all is done and the final `}` is "executed", the column is empty, leaving the table with no rows at all. If a sub-table is processed, the alignment stack's current level (other than 1) is also traced, which makes searching for matching traces easier.

If that's what goes on in an empty table, imagine how important full tracing is in a non-trivial example, of which there are many more in `trip.tex`.

## Tracing synonyms and conditionals

$\TeX$  traces the *meaning* of the executing control sequence, not its name. So if the control sequence is a synonym for a register, or a primitive command, or other name (e.g., created by a `\let`), it can get confusing. For example, here's line 161 from `trip.tex`:

```
\dimendef\varunit=222\varunit=+1,001\ifdim.5\mag>0cc0\fi!pt
```

This defines a name `\varunit` for the dimension register at index 222, and assigns a value to it. The value's digits are conditionally expanded using an `\ifdim` conditional smack dab in the middle of collecting the value's digits. The test asks if half of `\mag`, whose value is 2000, is greater than 0 (Cicero) points, which is true. But there's an implicit conversion from an integer `.5\mag` (1000) to a fixed-point dimension (0.01526pt).

$\varepsilon\text{-}\TeX$  would trace some of the statements of this line generically as

```
{\dimendef}
{changing \varunit=undefined}
{into \varunit=\relax}
{changing \varunit=\relax}
{into \varunit=\dimen222}
{\dimen222}
{\ifdim: (level 1) entered on line 161}
{true}
{\fi: \ifdim (level 1) entered on line 161}
{changing \dimen222=0.0pt}
{into \dimen222=1.001pt}
```

whereas JSBox traces the same code, in both shorter and more faithful-to-the-source fashion, as

```
{\dimendef}: \varunit = \dimen222
{\varunit} ...
{\ifdim} ...
{\mag}: 2000 = "07D0 [parameter]
... {\ifdim}: 0.01526 > 0.0 ? true [line 161]
{\fi}: [end of \ifdim on line 161]
.. {\varunit}: \dimen222 = 1.001pt [was 0.0pt]
```

Notice that the extra definition to an intermediate `\relax` is an arcane internal  $\TeX$  implementation detail (see the comments in "`tex.web`") that is of no interest to 99.999% of users, but  $\varepsilon\text{-}\TeX$  traces it anyway due to the generic nature of how it reports changes to interpreter values. JSBox's focused tracing during name definition elides this extra internal reference management step.

We announce `{\varunit}` at the start of its trace (as opposed to `{\dimen222}`), because it comports with the source code. When the command has finished recursively expanding its argument, the final re-trace shows the value assigned, and the register index, and the former value, all in one line.

When JSBox traces a conditional, it shows the values used in the test as such, followed by a `?`, followed by the answer, either `true` or `false`, all in one line. A hint can be added, followed by the test's line number as final commentary. If a `\fi` ends a multi-line conditional, the line range is used.

The only other information that might arguably be brought to the user's attention is what happens to the single character `0` collected as another digit in the dimension's value, and the fact that the original source tries to present (after the conditional is through mucking with things) the value `1.00101pt` as `\varunit`'s value. But the final `1` is insignificant as there is round-off to `1.001pt`, which is what's traced.

Line 360 of `trip.tex` has a crazy `\ifcase` conditional statement, with nested `\ifcases`:

```
\ifcase\iftrue-1a\else\fi
\ifcase0\fi\else\ifcase5\fi\fi
```

that JSBox traces fairly cogently and concisely as

```
{\ifcase} ...
{\iftrue}: true [line 360]
.. {\ifcase}: looking for case -1 [only matches
an \else clause, if any] [line 360]
{\else}: false
{\fi}: [end of \iftrue on line 360]
{\ifcase}: looking for case 0 [matched]
[line 360]
{\relax [internal]}
{\fi}: [end of \ifcase on line 360]
{\else}: true [no previous case matched -1]
{\ifcase}: looking for case 5 [line 360]
{\fi}: [end of \ifcase on line 360]
{\fi}: [end of \ifcase on line 360]
[no case matched]
```

A  $\TeX$  language interpreter must insert an internal `\relax` to enable expansion to parse an inner nested conditional involved, as here, in constructing the condition of an outer conditional. The only remaining information that might be traced in some way, but isn't, is which commands or characters in the source code are skipped over for false matches, and why. This would help explain the mysterious disappearance of the letter 'a' when executing the foregoing code (it ends up being part of the outer case 0, which is skipped while scanning for case -1).

The `trip` test has another strange boundary condition test: what happens to an `\if...` statement with more than one `\else` clause. Peculiarly, sometimes it's an error, and sometimes it's not. So

I had fun ensuring that JSBox's trace of line 389, where multiple `\elses` are not an error,

```
| \ifx T\span\else\par\if\span\else\else\else\fi\fi
```

would be enlightening to the user:

```
{\ifx}: T = \span ? [Chr = Cmd] false [line 389]
{\else}: true
  {\par}: [building more page]

% t=30.0 plus 42.0 plus 1.0fil minus 8.0    g=16383.99998 b=0 p=0 c=0#    [# = best break so far]

  {\if}: \span=\relax [internal] ? [neither is a character, treated as equal] true [line 389]
  {\else}: false
  {\else}: still false [ignoring extra \else clause]
  {\else}: still false [ignoring extra \else clause]
  {\fi}: [end of \if on line 389]
{\fi}: [end of \ifx on line 389]
```

Note that when tracing an `\else` clause, its line number is suppressed if it is the same as that of its initial `\if...` condition. This makes tracing short conditionals like the above less noisy. And again, notice in the above trace the paragraph building data that's caused by executing the `\par` command. When output of a different class is created, we strive to set it off with blank lines from the surrounding indented trace lines, so as to stand out as non-tracing.

### Other tracing niceties

Every hundred or so traces, JSBox inserts a short context announcement that shows the current set of included files and the line number in each from which the scanner is reading. For example, if we were including `trip.tex` from another file, such as `test.tex`, the announcement might look like:

```
>> display math mode
  {\^}: [superscript]
  {\mathop}

"test.tex"[Line 4] > "trip.tex"[272]

  {b}: letter maps to "7162
      [class 7 (inner);
      \fam 1 \char "62 = 98 = b]
  {\nolimits}
```

This makes it easier for the reader to understand where nearby tracing in the log file is arising from in an included source file. It also lessens the need to include line numbers in individual traces.

Formal error reports are usually preceded by a full execution stack dump, showing included files, the macro call stack, and an indication of empty stack frames, deleted to allow tail recursion.

When executing `trip.tex` the execution stack is never very deep when errors are reported. So the following is an example of a stack dump showing nested macro calls while executing a test file called `plainstory.tex` that relies on footnote macros defined in the `plain` format, whose non-dumped source code is read in at the start of the job:

```
Context | "plainstory.tex"[Line 112] >> \story >
        | \rhubarb > \fubaru > \bar > \foo >
        | \testparagraph > \note >> \footnote >
        | \vfootnote
```

The `>>` indicates that there were empty stack frames that were deleted during regular stack cleanup (the last two macro names are defined in `plain.tex`).

Another nicety in JSBox, tracing the resolution of input files, is not as demonstrable using the `trip` test. JSBox is a system-agnostic library that can be linked into a client program. The client is responsible for mediating between the interpreter and the system, and must install a callback function with which the interpreter asks the client to do various tasks.

In particular, when JSBox executes the `\input` command, it first asks the client to vet each character in the file name. This lets the interpreter issue an error message at precisely the right time for any illegal or unwanted character. Once the file name is collected and analyzed a bit more, the interpreter then asks the client to construct a list of directories in the client's file system where to look for that file. As the list is iterated, a back and forth between the client and the interpreter allows all the unsuccessful folders to be traced as well as the final one where the file is found. To gate this, JSBox implements another integer parameter, `\tracingfiles`, that can

be set to 1. For example, when a source file inputs `plain.tex`, the trace might look like:

```
{\tracingfiles}: 1 [was 0]
{\input}: plain.tex
      [not found at ./Projects/TUG Article/plain.tex]
      [not found at ./JSBox/Projects/plain.tex]
      [aha! it's at ./JSBox/Library/Formats/plain.tex]
```

Thus, the input resolution strategy is up to each individual client program, but the interpreter can reveal and record some of that strategy for the user at the right time and in the right place. This is particularly important when, contrary to the user's expectation, no file is found, or when an incorrect file of the same name but with a different path is found first (e.g., the wrong version of a file).

### The complete JSBox trip test trace

The foregoing should give the seasoned T<sub>E</sub>X log file tracer/reader a good idea of how the execution of a piece of T<sub>E</sub>X source code can be traced in a much nicer and more useful manner than what the T<sub>E</sub>X engine does, even considering the  $\varepsilon$ -T<sub>E</sub>X extensions.

The entire and latest complete JSBox trace of the trip test can be found as a PDF at <http://www.mathemaesthetics.com/JSBox/triplog.pdf>. It's about 160 landscape pages long. I continue to refine and add to the library's tracing, so the log file will be updated occasionally. Indeed, I tweaked at least one feature (eliding line numbers in `\else` traces when on the same line as the initial `\if...`) as a result of creating examples for this article.

### Conclusion

Over the course of this multi-year project, I have perhaps four times re-designed and re-written how tracing should work, each time realizing that what I was doing in the previous iteration wasn't complete enough. There are still a few indentation bugs in the code, and some older cruft that deserves to be re-written or cleaned up.

The JSBox library is a work-in-progress. Over 2000 pages of portable C code, half of it English comments, it supports plenty of other interesting features, a description of which can be saved for

another time. The library is not yet ready to be deployed as a new T<sub>E</sub>X language engine. When I started on this project in 2009, I had very little idea of the complexity of the task of being compatible with the T<sub>E</sub>X language, which I barely knew. Its many quirks as a Turing-complete macro language seem as closely tied to its program's implementation details as they are to any overarching language syntax specification. And the greater T<sub>E</sub>X ecosystem is even more complex. Indeed, my goal of complete tracing is to make executing T<sub>E</sub>X code as self-documenting as possible.

Debugging the interpreter so that it would execute the trip test correctly — i.e., mathematically and functionally equivalent to what T<sub>E</sub>X does — required many months of work. Each bug it revealed had to be fixed prior to moving on to the next, due to the cascading nature of layout calculations. In several cases, I had to completely re-implement how certain primitives worked internally, after my initial assumptions proved invalid.

There's a lot of work left to do. But at the very least, this interpreter can — in full and gory detail — now tell the world nearly all of what it's doing under the hood. Which is why the chapter on tracing utilities in JSBox's source code has the title

#### Understanding Interpreter Execution Vanishes Without A Trace!

- ◇ Doug McKenna  
Mathemaesthetics, Inc.  
PO Box 298  
Boulder, Colorado 80306, USA  
doug at either mathemaesthetics  
dot com or dmck dot us

Editor's note: An overview of JSBox is available via the slides at <http://tug.org/tug2014/slides/mckenna-JSBox.pdf>. We hope to publish additional related papers on this work as it progresses.

## Creating (mathematical) jigsaw puzzles using T<sub>E</sub>X and friends

Julian Gilbey

### Abstract

The jigsaw puzzles considered here are constructed from shapes such as triangles, squares and so on, with questions and answers written along their edges. The aim is to match them up correctly. Related puzzle varieties are card sorts and dominoes. We describe a T<sub>E</sub>X- and Python-based system designed to author such puzzles. The input is a text-based YAML file; the output can include both printable PDFs for cutting up and Markdown files for potential conversion to HTML.

## 1 Background

### 1.1 History and benefits of (mathematical) jigsaw puzzle software

In 2005, Hermitech Laboratory created *Jigsaw 2005*, software designed to create mathematical jigsaws. These are puzzles consisting of square and/or triangular pieces which fit together to make a hexagon or other shape. Two edges match if a question on one matches the answer on the other; figure 1 shows an example of a completed jigsaw. These can be used within classrooms to help make learning more engaging and enjoyable. For example, some students feel disengaged by having to continuously write in mathematics lessons. Students may well attempt many more questions than normal and participate more willingly when solving a jigsaw puzzle than when answering textbook questions. It is also different from working with a computer-based activity in that it is more physical and can easily involve an element of collaboration with classmates.

Jigsaws can also be used to develop certain logical thinking skills. For example, some questions could have the same answer, so that students would have to realise this and determine which one of the possible pieces fits all of the constraints. There can also be blanks or ‘?’ symbols used to indicate that an answer has been left out. Deliberate mistakes could be introduced to raise the level of challenge. Finally, the edges could be used to introduce distractors (as in the example shown), or they could be left blank, or they could be used to spell out a relevant word, phrase or sentence.

While the *Jigsaw 2005* software was originally designed for use in the mathematics classroom, the same puzzle structure could easily be used to assist in learning languages, scientific facts, and so on.

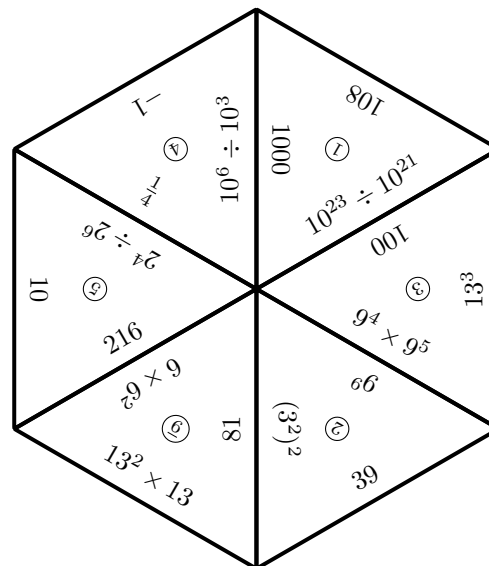


Figure 1: A small completed jigsaw

In addition to jigsaws, the software also offered dominoes, where each card has an answer and another question; these then match up to create a domino chain. Another type is a card sort, where a set of cards is to be sorted either into order (for example, a proof of the irrationality of  $\sqrt{2}$  or the derivation of the formula for solving quadratic equations), or into groups (for example, “Which of these statements are always true, which are sometimes true, and which are never true?”).

The *Jigsaw 2005* software was distributed to all UK schools and colleges with 16–18 year-old students as part of a project to improve the quality of mathematics learning. Since then, the software has been further developed, and is now freely available as *Formulator Tarsia* [3] (though it is not open source). Hundreds of classroom activities have now been written using this software.

The *Formulator Tarsia* software is Windows-based, and every question–answer pair is entered on a separate input screen. There are keyboard shortcuts available, but even having learnt them all, there is still a fair amount of mouse-work required to enter the questions. One also has to switch to different ‘tabs’ to see an overview of all entered pairs. On the other hand, the software learning-curve is very gentle, which is a huge bonus for busy teachers: though it may take time to create each jigsaw, there is very little up-front time investment required. Also, the data storage format is XML, with MathML for the mathematics, so it is potentially possible to edit the files outside of the software or to export the data.

## 1.2 The desire for a $\text{T}\text{E}\text{X}$ -based system

I am currently developing mathematical resources for the Cambridge Mathematics Education Project (CMEP) [2]. We are aiming to provide innovative resources to help support and inspire teachers of advanced mathematics for 16–18 year-old students. In the UK, most of these students will study an A-level mathematics curriculum, but the material we are producing can certainly be used in other teaching situations and for other curricula.

Our build system involves authors writing the original content in Markdown, which is then converted using `pandoc` [4] to HTML for viewing and to  $\text{L}\text{A}\text{T}\text{E}\text{X}$  for producing a printable PDF document. (There are more steps in addition, but this lies at the heart of the conversion process.)

Among the resources we are developing are a variety of card-sorting activities, and there is a possibility that we may also offer some jigsaw activities. The *Tarsia Formulator* software does not meet our needs for a variety of reasons:

- All of our developers use either Mac OS X or GNU/Linux, so it would be hard for us to use Windows-based software.
- The resources are going to be made available via the web, so we require a way of having the content of our cardsorts available both for easy printing and for viewing online. It would be painful to have to enter the content from scratch into both a card-sort creator and a Markdown document.
- The WYSIWYG input system is quite laborious.
- We like the flexibility of  $\text{L}\text{A}\text{T}\text{E}\text{X}$ 's mathematical typesetting abilities (in spite of the limitations imposed by using `pandoc` and *MathJax* [5]).
- It is desirable to have more fine-grained control over the output, should that be desired; without access to the source code, it is very hard to make any systematic changes to *Formulator Tarsia*'s output.
- When an activity has been created in *Formulator Tarsia*, it seems to be impossible to change the activity type (say from a triangle-based jigsaw to a domino puzzle) without either editing XML files by hand or re-entering all of the data; it would be much nicer to have a text-based input file which can simply have its header information modified appropriately or the content cut-and-pasted into another document.
- The output quality of the various  $\text{T}\text{E}\text{X}$  engines is superior.

For all of these reasons, I decided to create a  $\text{T}\text{E}\text{X}$ -based solution to address our needs and produce beautiful resources.

## 2 The new software

### 2.1 Design goals

My aim in creating this new software was to build on the great work that the creators of *Formulator Tarsia* had done, while addressing some of the shortcomings that we had identified above. The audience of this new software is also crucially different, in that it assumes its users have both a working knowledge of  $\text{L}\text{A}\text{T}\text{E}\text{X}$  and a  $\text{T}\text{E}\text{X}$  system installed. They will also need to have Python 3 and be able to install some standard Python modules. (I might endeavour to make the software compatible with Python 2.7 at a later stage if I have time and people express an interest in this.)

The primary design goals were:

- The content should be easy to input using a text editor (assuming they have a basic knowledge of  $\text{L}\text{A}\text{T}\text{E}\text{X}$ ).
- The puzzle layouts should be editable if desired.
- New puzzle types should be easy to create, ideally without having to modify the source code.
- It should be easy to create the printable jigsaws from the input files.
- There should be sensible defaults which can be overridden if desired.
- It must be possible to include images in the puzzles.
- The software must be able to output a Markdown version of the puzzle content for including into our CMEP build process. The precise format of this must also be customisable.
- The software must be able to run on at least Mac OS X and GNU/Linux, and ideally on Windows too.
- It should be easy to install and use.
- The dependencies should be kept to a minimum (beyond a working  $\text{L}\text{A}\text{T}\text{E}\text{X}$  system).
- It should be easy to maintain (time is always in short supply).
- It should be possible for the jigsaw pieces to be automatically numbered.

For the last item, I often found as a teacher that it was very time-consuming to check whether a jigsaw puzzle or card sorting activity had been completed correctly. I would therefore number the pieces before photocopying a puzzle, to make this task more efficient. However, I still had to go to the effort of identifying the pieces in the solution to

determine where the numbers would end up. So the ability to automatically number the pieces, both in the puzzle and solution, is very desirable. To make this effective, though, the cards had to be shuffled randomly for each different puzzle, so as to prevent the students from ‘solving’ a puzzle just by arranging the card numbers in some standard order.

A possible additional feature would be to create some form of graphical data input system, but I have no immediate plans to do so.

## 2.2 Implementation overview

The most important files from a user’s perspective are the data input files. These specify the type of puzzle (a hexagonal jigsaw or a card sort, for example), any options for the jigsaw (such as whether to number the cards or not) and the text to appear on the puzzle. Sensible defaults are provided for all of the options if they are not specified. We decided fairly early on that YAML was a suitable markup language for writing these files, as it is a very easy format for humans to read and write, with little “noise”. It also made more sense to use a standard, well-known markup language than to create a new one specifically for this project: it will then be easy for other software to read the input files or to change the files to a different format (such as JSON) should this ever be desired.

The puzzle templates are  $\LaTeX$  files with templating marks. For the templates I have created, I have used PGF/TikZ for the graphics and to place the text items. My initial templates were written when I was using version 2.10 of PGF/TikZ, as distributed with  $\TeX$  Live 2013. However, it turned out that I ran into a bug related to the placement of text along cyclic paths, which has been fixed in version 3.0.0 (distributed in  $\TeX$  Live 2014). It is therefore necessary to have an up-to-date  $\TeX$  Live distribution (or at least an up-to-date PGF) for these templates to work correctly. Alternative templates can be written if this is desired. For example, someone might prefer to use Asymptote or another graphical package, or they may wish to modify the existing templates in various ways.

There is also a template description file (again written in YAML) for each puzzle type: this specifies various parameters required to create the puzzle.

Both of these template types (the puzzle templates and the description files) are described in detail in the software documentation.

I wrote the program itself in Python. This was for a few reasons. Firstly, it is a well-known, popular language, so if other people wish to become involved in developing this software, it will be relatively easy

```

type: smallhexagon
title: An example puzzle
note: 'You will have to work out the
      missing number shown as `?'`

pairs:
- ['$10^6\div10^3$', '$1000$']
- ['$10^{23}\div10^{21}$', '$100$']
- ['$9^4\times9^5$', '$9^9$']
- - '$(3^2)^2$'
  - puzzletext: '?'
  - solutiontext: '$81$'
- ['$6\times6^2$', '$216$']
- ['$2^4\div2^6$', '$\dfrac{1}{4}$']
edges:
- '$-1$'
- '$10$'
- '$13^2\times13$'
- '$39$'
- '$13^3$'
- '$108$'

```

Figure 2: Example small hexagon puzzle

for them to do so. Secondly, the Python interpreter is easy to install on the major platforms people will be using. Furthermore, since the software is written in pure Python, it does not require compilation, making things a little simpler to install. Finally, it provided me with an opportunity to learn more about this language: as I have learnt more about Python, I have improved my code and made it more idiomatic. The code is currently written in Python 3.x; if I have time and people express an interest, I will endeavour to make it compatible with Python 2.7.

To create the jigsaws, the jigsaw generator program is run over the data file. It reads this file along with the relevant template description file and puzzle template files. It then fills in the puzzle templates with the puzzle data to create  $\LaTeX$  files (and Markdown files too, if requested). The  $\LaTeX$  files are then processed with pdf $\LaTeX$  (or some other  $\LaTeX$  variant) to create PDFs for printing.

## 2.3 An example data file

Figure 2 shows the YAML puzzle file which was used to create the example shown in figure 1 above (with some small modifications).

The file begins with some metadata:

- the type of the puzzle: in this case, it is a ‘small-hexagon’ puzzle, which consists of six triangles;
- the title of the puzzle, which is optional, and
- an optional note, which is printed above the puzzle.



The existing jigsaw types at the time of writing are `hexagon`, `smallhexagon`, `triangle` and `parquet`; there are a few more in the pipeline, too. It is also possible to write one's own jigsaw types, as long as they consist of equilateral triangles and squares. (To write templates using different shapes would require extensions to the software itself; I may do this in the future.) In addition, there are three more types, `cards`, `cardsort` and `dominoes`, which are described briefly later.

The next section of the file specifies the data. There are two parts here: the pairs data, which lists question and answer pairs, and the edges data, which specifies the text to appear on the edges in an anticlockwise direction. Each pair is a sequence of two items, the question and the answer, whereas each edge consists of just a single item.

Each item is usually just a single string. However, there is a possibility of 'hiding' text in the puzzle. This means either leaving a blank where the text should appear or writing something else in its place. The fourth question–answer pair in the above example illustrates this. This pair is written using YAML 'block style' for clarity, that is, the question and answer are written on separate equally-indented lines. The question is just a plain string ('\$(3^2)^2\$'), whereas the answer is a mapping with two entries: the `puzzletext` appears in the puzzle, while the `solutiontext` appears in the solution. This feature could be used as in the example shown, or it could be used to introduce deliberate mistakes in the puzzle, increasing the level of difficulty for the students. There is also an alternative notation available to simply hide the text in the puzzle, which is described in the documentation.

If a puzzle does include such hidden text, then the solution highlights these occurrences; in the default templates, these are shown with a yellow background.

It is also possible to include images in the items or to change the text size of individual items or all items. The details are described in the documentation.

### 2.3.1 Some notes on YAML syntax

For a full, precise description of YAML syntax, see the YAML Specification [1]. What follows is a very brief summary of some of the basic parts of the specification which should be sufficient for most people's needs when using this software.

Although YAML allows unquoted strings in general, there are a number of restrictions on what is permitted in them. For this reason, it may be simplest to single-quote all value strings. (YAML also

offers double-quoted strings, but these interpolate backslash-escapes; this is probably undesirable in this context, since many  $\text{\TeX}$  expressions include backslashes.)

Within a single-quoted string, a single quote is written as a doubled quote mark (''), hence the three quote marks in a row at the end of the note field in the above example (two to indicate a quote, and the third to end the string).

For collections ('sequences' and 'mappings' in YAML's terminology, each of which consists of a number of 'entries'), YAML allows two different notations: either a flow style, which is similar to JSON notation, or a block style. With the flow style, the entries of a sequence, such as a question–answer pair or the list of edges, are enclosed in square brackets and separated by commas; for the block style, each entry appears on a new line preceded by a vertically-aligned hyphen.

Similarly, for a mapping the entries consist of key–value pairs, with the key and value separated by a colon. They can be written either using a flow style as a comma-separated list enclosed in braces, or with a block style by writing each key–value pair on an identically-indented new line.

Both of the sequence styles appear in the example here, though only the block style is used for mappings. Note also that the top-level structure of the file is itself a mapping. This means that the entries (type, title, pairs, and so on) can appear in any order. However, for the benefit of the human reader, it is wise to maintain a meaningful order to these entries.

## 2.4 Card sorts and dominoes

As mentioned above, this software also offers some other types of activities: card sorts and dominoes.

A domino puzzle is just a collection of question–answer pairs which are laid out on a series of dominoes. Each domino consists of an answer and a new question. The dominoes can then be laid out to create a complete chain (beginning with 'Start' and ending with 'Finish') or loop (if 'Start' and 'Finish' are not present), with each question matching its corresponding answer. The data file used to create this is very similar to the example shown above, only the type is now `dominoes`. Within the data file, it is also possible to specify various options such as how many dominoes should appear on a page and whether the cards should form a loop or chain; these are described in more detail in the documentation.

A card sort is simply a collection of cards which are to be sorted in some way. For the `cardsort` type, the aim is to sort the cards into the correct order, and so the cards are shuffled for the puzzle

and a solution is also produced by default. For the `cards` type, on the other hand, there is no canonical order (for example: “Arrange these cards into order of importance to you” in a politics lesson), and so no solution is produced, nor are the cards shuffled by default. Again, the data file is very similar in structure, and details can be found in the documentation.

Both cards and dominoes offer the possibility of having some form of title on the cards, and cards have a further option of having labels on individual cards. (This was introduced into the software when we wanted to create an activity which had different categories of cards; the categories were then written on the cards.) Additional options are discussed in the documentation.

## 2.5 Markdown output

As explained in section 1.2, our requirements include the need to output a Markdown version of cards data for inclusion into our build system; from there, it is translated into HTML. Our current system requires each card to be embedded in an HTML `<div>` element. The cards are then displayed in an appropriate way using some simple CSS. Since `pandoc` passes any HTML `<div>` elements in a Markdown file to the HTML output unchanged, we simply need a Markdown file with the `<div>` elements already present for our card sorts. I have therefore created a Markdown template file which places the card content within these elements.

For other needs, it is perfectly straightforward to create alternative Markdown templates, which could then be converted into the required HTML. For example, with the appropriate JavaScript and supporting CSS, it would be entirely feasible to create an interactive version of the card sort or jigsaw from the same data file. While we have not yet done this, it would be a very interesting next step.

## 3 Status of the software

As the time of writing, the software is in alpha state. It is currently able to produce jigsaws and card sorts in all of the ways discussed in this article. There are a few key issues outstanding, which should be resolved in the very near future, including:

- creation of an installable Python package;
- handling command-line options;
- the ability to read a configuration file, specifying such things as the flavour of  $\text{\LaTeX}$  to use;
- offering the ability to read user-defined template files, and
- writing the full user documentation.

Once these are done, I will consider the software to be in either beta or release-ready state. I welcome feedback and any suggestions for improvements or enhancements, as well as stories of how it has been used.

### 3.1 Obtaining the software

The software can be downloaded from GitHub; the repository is <https://github.com/juliangilbey/jigsaw-generator>. At the time of writing, the simplest way to obtain it is to use git to clone the repository. As mentioned above, I intend there to be an installable Python package by the time this article is published. It might then be appropriate to upload this package to CTAN. Information about this will be posted on the GitHub site.

### Acknowledgements

I thank those who offered very useful ideas and feedback at the  $\text{\TeX}$  Users Group conference; these have helped me to solve some of the thorny issues I had been facing.

### References

- [1] Oren Ben-Kiki, Clark Evans, and Ingy döt Net. YAML Specification. Available from <http://www.yaml.org/spec/1.2/spec.html>.
- [2] Cambridge Mathematics Education Project. <http://www.maths.cam.ac.uk/cmep/>.
- [3] Hermitech Laboratory. Formulater Tarsia. Available from <http://www.mmlsoft.com/index.php/products/tarsia>.
- [4] John MacFarlane. The Pandoc universal document converter. Available from <http://johnmacfarlane.net/pandoc/index.html>.
- [5] MathJax Consortium. MathJax. Available from <http://www.mathjax.org/>.

◇ Julian Gilbey  
 Department of Pure Mathematics  
 and Mathematical Statistics  
 University of Cambridge  
 Wilberforce Road  
 Cambridge CB3 0WB  
 England  
 J.Gilbey (at) maths dot cam dot  
 ac dot uk  
 jdg (at) debian dot org

---

SUTRA —

## A workflow for documenting signals

Pavneet Arora

### Abstract

Modern home design increasingly emphasizes electronic signals to carry not only data and entertainment, but also basic comfort and security functions throughout the house. The connected devices along these signal paths are almost never static, but are replaced or augmented regularly depending on the homeowners' needs or whims. This proliferation of signals, often implemented on an *ad hoc* basis, creates difficulties when it comes to effective implementation, but even more importantly, for diagnoses when things inevitably go wrong. The need for some form of documentation is clear; but what representations make sense both for capturing the implementation details as work progresses, and then as an aid in the discovery phase of diagnosis? SUTRA is a documentation workflow that builds upon the earlier work introduced in the YAWN framework. It relies on YAML for data representation, the Ruby language for processing, and, in this case, uses ConTeXt's natural tables mechanism to collate and present this information in a useful way to round out YAWN's model-view-controller projection onto the problem space.

### 1 Introduction

The humble abode has evolved from an unserviced shack bereft of either electricity or plumbing to an inter-connected set of systems — power, heating and cooling, electronic controls — that wind their way behind walls and through floor joists. Their scope is both dazzling and staggering in its complexity, and yet we assume they will instantaneously and reliably function.

On top of such basic services are layered audio and video distribution throughout the house, surveillance, alarm, data networking, and telephony. Not to mention that all of these services, primary and ancillary, are also being pushed outside into the garden and all the way to the lot perimeter. And these are just the elements that are the responsibility of the homeowner. They are in turn “lit” by the utility providers that have interfaces to the house systems.

Control and automation add yet another layer in an effort to bring these disparate systems under some semblance of manageability for the homeowner, but naturally each new layer adds even more complexity to the underlying infrastructure.

Plugging into these various systems can be fixed wire devices such as traditional thermostats, alarm

keypads, computers, etc., as well as transient devices such as mobile phones, tablets, wand remotes, and others — both trusted and known, as well as interlopers brought in by guests who might be given temporary access.

Basic design documents used for construction or renovation almost always excludes any information about several of the layers mentioned above, as they do not come under the governance of building permits, or will leave the details of the implementation to the supplying vendor in a form of *late-binding* where details are expected to be discussed with the homeowner at some time in the future. As a result, it is common to go to tender for a generic “pre-wire” package with trust placed in the cabling vendor, often lowest-cost, without much attention paid to what this wiring will afford in terms of connectivity. Once the equipment selection is complete it is left to the implementation team to install the equipment using whatever wiring they find.

It is the rare consumer that even bothers to understand what is underneath the surface, satisfied as they are to interact through the various interfaces. And rightly so, as it should not concern them beyond being able to access the service they seek, any more so than a driver getting into their car and expecting it to start, move and turn under their command without requiring to know if their steering is driven hydraulically or electrically. Unfortunately, it is the equally rare installer that takes care to note down what they have done on site so that there might be an entry point of understanding in the future for either their next visit, or that of the next technician.

The end result is that there soon grows a vast gap between the actual as-built state of the wiring and the systems built upon it, and what is known about that state. This lack of information makes the systems fragile and susceptible to failure, a most unwelcome outcome given their increasing complexity and the utter dependence of the homeowner on them.

### 2 A specific example to illustrate the problem

To illustrate the rise in complexity of signal propagation in a residential setting, let us consider, in isolation, the example of music playback. The arrival of consumer stereos in the 1960s soon resulted in a new essential fixture in many homes, alongside televisions. However, even its more complex forms, e.g., a pre-amplifier driving a pair of mono-block power amplifiers which, in turn, would drive a pair of loudspeakers now seems laughable in its simplicity. A cursory visual inspection would suffice to make sense of which wire went where, and which did what.

The integrated amplifier which followed, incorporating the pre-amplifier and the power-amplifier in the same chassis, and then its successor the audio receiver which also included a tuner, simplified the wiring further, now reduced to just the speaker wiring and, perhaps, wiring from an additional source such as a turntable.

Now consider the modern audio/video receiver, which is tasked with not just taking source audio signal to speaker, amplifying it along the way, but also takes responsibility for video switching, sometimes transforming and scaling the video signal, processing digital audio signals into multi-channel audio, handling output to multiple zones while allowing repurposing of its amplifier channels to be assigned to these secondary zones. Accepted video signals can be selected from amongst a variety of interfaces, both analogue, e.g., component video, and digital, e.g., HDMI. The control interface handled by the receiver almost always includes infrared, RS-232, 12V trigger, and increasingly IP-based control. Add to this the variety of source equipment such as set top boxes, dedicated media players, games consoles, mobile devices, and dare I say, for some die-hards, a turntable and CD-player. Source signals need not even be local to the unit, but may also simply be connected via the data network. All of this capability and complexity — just for a single device! Now let's consider that this is just one such unit amongst many in a house, and I think that one can soon appreciate the need for both a way of capturing this information, and equally for representing it in a way that is both digestible and easy to navigate.

### 3 How to capture the information?

#### 3.1 What information is needed?

In order to look at methods of information capture we must first understand just what information would be useful. The emphasis on the word *signals* in the present title gives a clue, in that what is of primary importance are the signals themselves, i.e., the connections between the different components that are the conduits for the signals. Components may act as either the source, destination, or intermediate processor of a signal but it is the connections that reveal the topology of the implementation, and are essential in understanding how the system behaves overall and how pieces within the system interact.

Of course, identifying equipment is also necessary, but we can think of equipment as a set of connection termini that happen to be physically grouped together.

The attributes of each connection that I feel essential to reveal its purpose are:

**Identity** Name of the connection. This could be just the physical port name on the component chassis or the assigned port name in the component's software configuration. Using the example of the A/V receiver mentioned earlier, this could be, for example, *Front Right Speaker* or *HDMI 1*.

**Interface** The type of physical connection.

**Intent** What is the purpose of the signal that the connection is carrying?

**Implementation** Details about the connection. An example might be: in connecting to the infrared interface, we might be using the *blue* and *green* pairs for *signal* and *ground* respectively of the pulled UTP cable between the location IR connecting block and the mouse emitter. Such implementation details should be captured as well.

**Interaction** To what component(s) does the signal travel? And to which port(s) of that destination equipment does it connect?

I call this set *5i* in counterpoint to the exclamation of frustration, “Ay-Ay-Ay-Ay-Ay”, that usually accompanies confronting an undocumented equipment closet under time pressure, trying to fix a problem that is unknown in scope.

#### 3.2 Layers

Another aspect that is important to note is that of *layers* or *buses*. From the audio example above, it is clear that an individual component can operate along different buses and may be involved in transforming a signal so that it moves from one bus to another. So the *interaction* attribute needs to encompass the movement of a signal along a single or multiple layers.

**Sharing** A connection operates within a single layer or bus.

**Traversal** A connection may switch or split a signal onto multiple layers. An example of this would be an HDMI audio extractor which decomposes the combined signal into its requisite video and digital audio components.

**Isolation** The component works in isolation. An example of this might be a power line conditioner that is IP-enabled to carry out scheduled or triggered reboot functions, but on the other hand it can also be left as a standalone unit.

#### 3.3 A simple method to capture signals information

For the information capture I took inspiration from Kent Beck's and Ward Cunningham's seminal work

on CRC (*classes, responsibilities, and collaborators*) cards [7].

I rely on 3" x 5" index cards, and each time I or my team would come across a piece of equipment we would jot down whatever we found out about its connections using the *5i* framework. Our field experience indicated that discovery of a system's function is more effectively done bottom-up for the simple reason that there are rarely any maps that would facilitate a top-down approach. In other words, when *step-wise refinement* is not an option then the alternative is to turn to *step-wise abstraction* [2]. For that reason, we chose to put our attention on a single piece of equipment and its *outgoing* connections.

Our focus on outgoing connections exclusively came from our belief that much of the intent of the connection is revealed from its source. Once this informal information capture was processed, the software could resolve and illustrate the requisite connections on the corresponding input ports of the destination equipment.

#### 4 How to present information about signals

When seeking to map out electronic connections, it is an easy assumption that a schematic is the natural and best representation. But is it always appropriate; moreover, is it even an effective representation for the domains described in this paper? There are several problems associated with schematics:

- They work best when there is a close mapping between the physical wiring layout and the logical design, e.g., a circuit board. When the critical information relates to the logical connection, there is a great deal of wasted real estate on the wire traces, all of which distracts from the essential information about the connection, and also consequently relegates this information to distant appendices.
- The above point relates to their use when the design goes through repeated refinements, and the results of this "stable" design are then produced in quantity. Unfortunately, almost every installation of the nature described in the paper is unique in that it is built up on an *ad hoc* basis, over considerable time.
- As the number of connections grows, the required size to present the layout soon becomes unwieldy. The schematic for the RaspberryPi, a relatively simple device, requires five pages, and this doesn't begin to include the underlying ARM Application Processor details.
- They also don't easily allow us to distinguish between the *physical* and *logical* location of components.

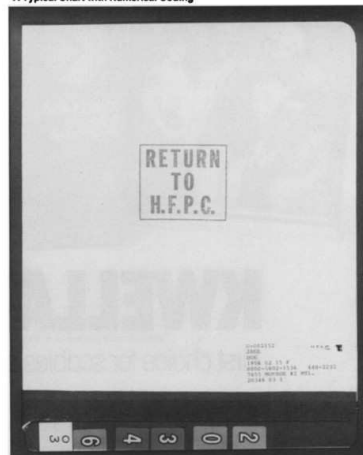
For some years we made a concerted effort to use Dia drawing software [5] under GNU/Linux to capture our own wiring installations. However, we struggled to keep our diagrams updated, and we still didn't have a methodology in place to quickly capture the information about the connections we found in the field. Upon analysis, the great shortcoming of this type of tool is that one spends an inordinate amount of time on layout, when one really wishes to enter the signal information in a raw form and have a suitable presentation generated for them. It was this need that led to the development of SUTRA.

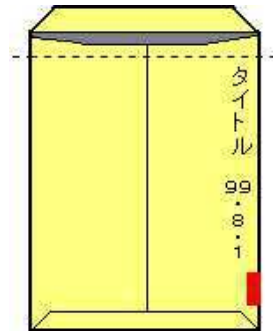
When seeking archetypes for data representation the following characteristics seemed desirable:

- The ability to group connections into representative components.
- The ability to identify the layers on which a particular component might operate. This would also help in narrowing to a subset of components that operate on a layer or bus of interest during a diagnostic exercise.
- A clear differentiation between the physical location of a component and its logical location, i.e., the zone or area on which it operates.
- When reviewing a component and its constituent connections—both outputs and inputs—the ability to have enough information about the source and destination without having to flip to different parts of the document.
- Additional information about the component, e.g., photographs, or a manufacturer's wiring diagram that identifies terminals not labelled on the component itself.

I have always been intrigued by how physicians file patient records: the racks of colour-coded files [6] from which could be retrieved the file of a single patient or all the files of a family with seeming ease:

Figure 1  
A Typical Chart with Numerical Coding





**Figure 1:** Sample of filing system by Prof. Yukio Noguchi.

More recently, I came across a filing system developed by Prof. Yukio Noguchi [3]; sample image shown in fig. 1.

From these sources, I experimented with designs and eventually settled on a prototype constructed using ConT<sub>E</sub>Xt's natural tables mechanism [4]. An early but representative version of the output is shown in fig. 2. One can see the coloured tabs around the perimeter of the table act as visual aids to classify the component. On the left are indications of input and output. On top is shown the *physical* location by *area*, *room*, *place*. At right is the analogous logical location (which defaults to the physical location unless overridden) and component identity. At bottom right are the layers on which the component operates.

The word *sutra* (सूत्र) is the Sanskrit word for a string. It seemed an apt name for our workflow since, unlike in a schematic where the physical layout dominates the presentation, it instead places emphasis on the source and destination, i.e., the connection. Of course, a large collection of strings can also invoke the image of a complex tapestry, which applies when reviewing large scale wiring diagrams. So the acronym SUTRA stands for its constituent components:

- Signals
- Unmasked using
- Table
- Representations with
- Annotations

## 5 What constitutes a signal?

While SUTRA's development initially arose from demands of electronic signals, its application has been successfully extended to domains where the definition of a signal can be a hybrid of connection types. There is no limitation within the workflow as to what constitutes a signal, and part of the appeal of its design is that it concerns itself only with connections

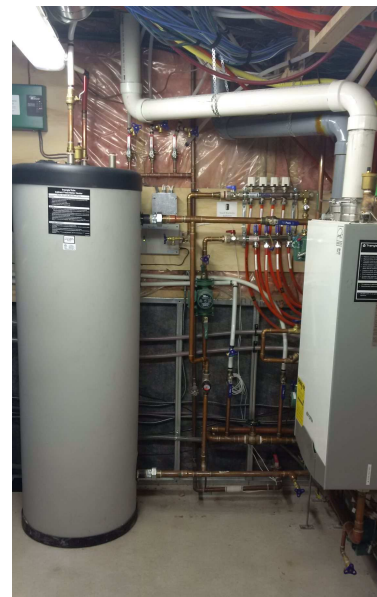
Physical Area		Physical Room		Physical Location	
PORT	SRC	INTERFACE	PORT DESC	INTENT	
INPUTS	Port	Interface type	Interface details	Connection's intent	
		Src System Area			
		Src System Room			
		Component Type			
	Component ID				
OUTPUTS	Port	Interface Type	Interface Details	Connection's intent	
		Dest System Area			
		Dest System Room			
		Component Type			
	Component ID				
		Layer 1	Layer 2	Layer 3	

**Figure 2:** Hand-crafted prototype using ConT<sub>E</sub>Xt's Natural Tables.

and components, however the user might need to interpret those.

## 6 SUTRA example

For this example I have purposely chosen a more generalized application of SUTRA. Upon my return from the 2013 TUG conference, I arrived to find our hydronic heating system misbehaving. Parts of it worked, but the in-floor heating did not, in spite of the boiler functioning as expected and a continuous call for heat from the thermostats.



To reinforce the point made earlier of the unwitting homeowner, I myself had not paid much attention to the details when the system was installed, and the installing technician could not recall the specifics of the installation. So I was faced with the ubiquitous discovery and diagnostics steps taken in such situations.

As is common, this system operates on multiple layers:

1. Electronic low-voltage. In reality this represents several layers. It is used for the controls, and in some cases the sensors. Thermostats, water temperature loop sensors, ambient air sensors, etc., all use low-voltage signals, but they aren't uniform: both AC and DC are used, as are dry contacts feeding relays.
2. Line (mains) voltage. In some cases pumps are controlled indirectly by connection to a control board on the pump, e.g., variable speed pumps, while in others they are controlled directly by mains voltage.
3. Water. This drives domestic hot water, in-floor heating, towel warmers, area radiators, and a heat exchanging air handler in staged fashion.

So SUTRA came to the rescue.

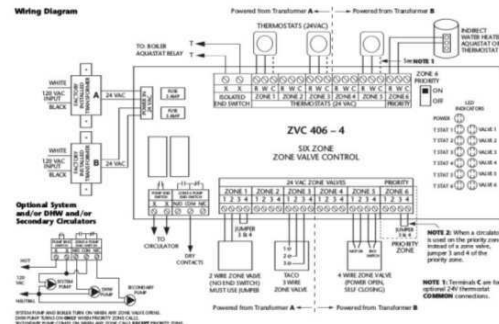
The implementation uses YAML for the input representation, following YAWN [1]. Here is some of the YAML for defining a single thermostat in the system:

```
:locAreas:
- :locArea:
  :locAreaID: F1
:locRooms:
- :locRoom:
  :locRoomID: Master Bedroom
:components:
# THERMOSTATS
- :component:
  :componentID: Tekmar 508
  :componentType: Thermostat
  :locPlaceID: Bottom of Steps
  (East Wall)
:layers:
- :layer: thermostat control
:connections:
- :connection:
  :type: thermostat
  :from:
  :port: 4-wire control
  :to:
  :sysAreaID: F0
  :sysRoomID: >
  Mechanical Room
  :componentID: Taco ZVC406
  :port: 4-wire control
  :portDesc: >
  Thermostat/Zone 1
:desc: >
  F1 Master Bedroom in-floor
  heating
```

In a multi-pass operation, SUTRA builds a database of connections, and then resolves forward references to the destination components. In this case, the destination of the thermostat connection is the “Taco ZVC406” zone valve control.

The SUTRA output is shown in fig. 3.

With the `--showresources` flag, SUTRA includes any ancillary material associated with a component. Here, the wiring diagram for the zone valve control is included in the generated output:



To bring our story to a resolution: in the end it was discovered that the thermostat in the hot water tank had failed open, resulting in a continuous call for heat. Since this is the primary circuit, all hot water generated by the boiler was directed to the heat exchanger of the hot water tank, with none for the floor heating.

## 7 ConTeXt considerations

Turning now to the ConTeXt implementation of the output we have seen, here are a few of the cases that the generated code must consider:

- How to handle the case when there are no output connections, only input connections? The layers operated upon need to shift from the bottom of the output connections to the bottom of the input connections, if any.
- How to handle the case when there are no input connections, but there are output connections? The component’s logical location and identity would shift to the output connections.
- How to handle the case when there are neither input nor output connections? An empty table needs to be generated so that the tabs can be affixed to it.

Some illustrative fragments of the ConTeXt code that generated fig. 3 are shown in fig. 4.

## 8 Conclusions

The problem of documenting system-wide signal paths in the construction industry can be overwhelming. The workflow SUTRA, built upon the YAWN framework [1], and utilizing ConTeXt’s Natural Tables mechanism, offers a viable option to make this problem tractable.





## Typesetting figures for computer science

Andrew Mertz, William Slough and  
Nancy Van Cleave

### Abstract

Presentation of concepts from computer science can benefit from informative diagrams and figures. These include trees, graphs, logic circuits, stacks and stack frames, algorithms expressed with pseudocode, code listings and memory layout, for example. Producing these types of diagrams can sometimes be challenging. Fortunately, there are a number of  $\LaTeX$  packages that can be used for this purpose. In this paper, we will illustrate the use of a few packages — including `tikz`, `bytefield`, `forest`, `drawstack`, and `listings` — that are well-suited for constructing high-quality figures for computer science.

### 1 Introduction

Popular wisdom dictates that *a picture is worth a thousand words*. Illustrations can be an indispensable aid to understanding new concepts, and never more so than in the classroom. The types of figures we most often use in our computer science courses fall mainly into one of four categories:

- systems: logic circuits, instruction set details, and stack frames
- algorithms/data structures: pseudocode, graphs, binary trees, and search trees
- theory of computation: automata, grammars, and parse trees
- discrete mathematics: graphs and trees

In all courses, there is a need for clearly presenting algorithms and code.

There are two approaches one can take: either generate a graphic with some third-party software, export it in an appropriate format and subsequently include it in the  $\LaTeX$  source with the `graphicx` package [1], or generate the graphic with one of the “friends” of  $\TeX$ , such as `METAPOST` [6] or `TikZ` [16]. Where practical, we prefer the latter approach, since the results blend well with the surrounding typeset text and produce vector graphics, contributing to their quality. This quality derives from the fact that images generated in this manner use the same fonts as the typeset text and can utilize mathematical typesetting as needed.

Additionally, there are a number of packages designed for special purposes — such as typesetting trees, stacks, and graphs, to name a few. Some of the more recent packages on CTAN make use of `TikZ`,

adding a syntax layer to make them easier to use while preserving excellent typeset results.

The number of packages available from CTAN can be somewhat overwhelming to a  $\LaTeX$  user with a specific task in mind. In a recent search, we found 35 packages for the topic *tree*. How does one choose from this embarrassment of riches? While references such as [4] and [10] certainly provide useful guidance they are snapshots in time, so a number of packages do not appear there.

Our purpose here is to provide examples of a number of typesetting tasks that are of interest to computer scientists. For each such task, we have identified packages we feel are especially appropriate. Although we are not providing tutorial introductions to these packages, we hope these examples may introduce readers to a few unfamiliar packages and provide motivation for further exploration.

### 2 Typesetting code

We often have a need to typeset code in a specific programming language, such as Python or Java. For such situations, we recommend the use of the `listings` package [5].

The `listings` package, introduced in 1996, is relatively mature and still enjoys current support. While perhaps best-known for its ability to produce “pretty-printed” output, verbatim-like results can also be produced. It provides support for more than 100 programming languages and dialects, including  $(\LaTeX)$  $\TeX$ , with the ability to define styles for yet others.

It is easy to get started with this package, knowing just two commands and one environment:

```
\lstset
\lstinputlisting
{lstlisting}
```

Appearance of typeset code is controlled with `\lstset`, which provides options for languages, colors, font sizes and styles, line numbering, and a host of other possibilities. Desired options are specified using a comma-separated list of key/value pairs, as follows:

```
\lstset{key1=value1,...}
```

An example of this is shown in Figure 1, where a number of such options are specified: `language`, `showstringspaces`, `columns`, etc. To typeset code, we can use either `\lstinputlisting` or `lstlisting`, the difference being where the code to be typeset is located. The command

```
\lstinputlisting{filename}
```

typesets the contents of the file *filename*, using the options previously requested. In contrast, code appears directly within an `lstlisting` environment:

<pre> 1 def gcd(p, q): 2     """ 3     Computes the greatest common divisor of 4     two nonnegative integers p and q using 5     Euclid's method. 6     """ 7     if (q == 0): 8         return p 9     else: 10        remainder = p % q 11        return gcd(q, remainder) </pre>	<pre> \lstset{language = Python,         showstringspaces = false,         columns = fullflexible,         numbers = left,         numberstyle = \tiny,         frame = single} \lstinputlisting{gcd.py} </pre>
--	---

Figure 1: Typeset Python code using the listings package, producing pretty-printed output.

<pre> 1 def gcd(p, q): 2     """ 3     Computes the greatest common divisor of 4     two nonnegative integers p and q using 5     Euclid's method. 6     """ 7     if (q == 0): 8         return p 9     else: 10        remainder = p % q 11        return gcd(q, remainder) </pre>	<pre> \lstset{basicstyle = \ttfamily,         columns = fullflexible,         keepspaces = true,         numbers = left,         numberstyle = \tiny,         frame = single} \lstinputlisting{gcd.py} </pre>
--	---

Figure 2: Typeset “anonymous” code with the listings package, producing verbatim-like output.

```

\begin{lstlisting}
code to typeset
\end{lstlisting}

```

Revisiting Figure 1, we see how Euclid’s algorithm for computing the greatest common divisor, as implemented in Python and stored in the file named `gcd.py`, can be typeset.

Most of the options specified here are evident from the typeset result: a single frame enclosing the code, tiny line numbers on the left, etc. Two of these options, however, are perhaps less obvious—`showstringspaces` and `columns`. Setting the first of these to `false` prevents the visible space (–) from appearing within any of the strings in the code. To understand the `columns` setting, it helps to know that there are two broad categories possible: fixed and flexible. A fixed column format adjusts the space between letters in an attempt to align columns; a flexible format makes no such attempt. Of the flexible formats available, we prefer `fullflexible`.

As a second example of the listings package, refer to Figure 2. Here, we did not specify a language, so no keywords are highlighted. To obtain a verbatim-like appearance, `basicstyle` is set to a monospace

font. Setting the `keepspace` option causes spaces which appear in the code to be respected which, in turn, preserves column alignment and indenting.

The listings package has many other options and advanced capabilities. For example, it is possible to include  $\TeX$  markup within a listing by providing “escaped” code, making it possible to blend mathematical comments with code.

### 3 Typesetting algorithms

To display algorithms in pseudocode, we think the style exhibited in [3] (famously known as *CLRS*, after the authors’ initials) is quite attractive, recognizing that its use requires some markup effort. For this, the `clrscode3e` package [2] can be used. An older version, `clrscode`, is also available; the 3e version matches the style used by the 3<sup>rd</sup> edition of *CLRS*.

Presenting an algorithm in this style is done using a `codebox` environment, as follows:

```

\usepackage{clrscode3e}
...
\begin{codebox}
algorithm, with markup
\end{codebox}

```

```

\Procname{\proc{Euclid-gcd}(p, q)}
\zi \Comment Pre: $p$ and $q$ are two nonnegative integers.
\zi \Comment Post: $\proc{Euclid-gcd}(p, q) = \func{gcd}(p, q)$.
\li \If $q \isequal 0$
\li   \Then
      \Return $p$
\li   \Else
      $\id{remainder} \gets p \bmod q$
\li   \Return $\proc{Euclid-gcd}(q, \id{remainder})$
\End

```

Figure 3: Typesetting an algorithm with `clrscode3e`.

```

EUCLID-GCD( $p, q$ )
  // Pre:  $p$  and  $q$  are two nonnegative integers.
  // Post:  $\text{EUCLID-GCD}(p, q) = \text{gcd}(p, q)$ .
1  if  $q == 0$ 
2    return  $p$ 
3  else  $\text{remainder} = p \bmod q$ 
4    return  $\text{EUCLID-GCD}(q, \text{remainder})$ 

```

Figure 4: An algorithm presented with `clrscode3e`.

Figure 3 shows sample markup for presenting an algorithm with this package; the typeset result is in Figure 4. A few remarks will clarify some of the markup details being used here. Consult the documentation provided with this package for more complete information.

`Euclid-gcd` is the name of this algorithm, so it is being identified as such with the `\proc` (procedure) macro. In a similar way, `remainder` is an identifier indicated by `\id`. Each of `\li` and `\zi` commands cause new lines to begin, either numbered or not. The package also supplies miscellaneous commands such as `\gets` and `\isequal` which produce assignment and equality operators. Commands for control structures, such as the conditional shown here, mirror those found in modern programming languages.

#### 4 Logic circuits

Circuits consisting of `and`, `or`, `not`, and `nand` gates are discussed in our systems course and also play a minor role in one of our general education mathematics courses. We have recently been looking at ways to produce diagrams of these types of circuits.

`TikZ` provides support for logic circuits and produces nice results. In this context, a circuit consists of a number of nodes and a collection of interconnections. The placement of the nodes can be specified either in absolute or relative coordinates. Using relative coordinates is handy, as circuits often consist

```

\tikzstyle{dot}=[fill,
  shape = circle,
  minimum size = 4pt,
  inner sep = 0pt,
  text height = 0pt,
  text depth = 0pt]
\tikzstyle{twoAnd}=[draw,
  and gate US,
  logic gate inputs=nn]
\tikzstyle{threeOr}=[draw,
  or gate US,
  logic gate inputs=nnn,
  anchor = input 2]

```

Figure 5: Some preliminary definitions for the majority circuit.

of components arranged either horizontally or vertically. Since the interconnections can be specified symbolically (e.g., “the output of the first and gate is connected to the first input of the or gate”), it is easy to stretch or compress the final drawing with just a minor change.

To illustrate, we will dissect some of the code details required to draw a circuit which computes a 3-input majority function. In the discussion that follows, refer to Figures 5 and 6. Since we want to use the U.S.-style gates found in the `circuits` library of `TikZ`, the following is needed in the preamble:

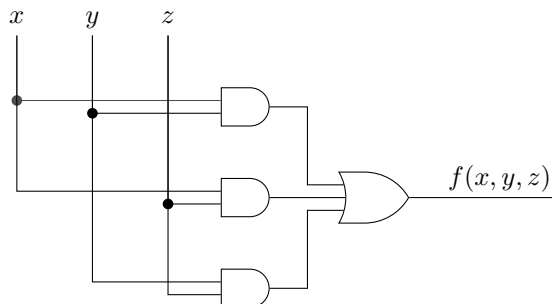
```

\usepackage{tikz}
\usetikzlibrary{circuits.logic.US}

```

In order to streamline some of the code for the circuit, Figure 5 introduces style names for each type of gate which will appear: a two-input `and`, and a three-input `or`. The specification

```
logic gate inputs = nn
```



```

\begin{tikzpicture}[x = 1cm, y = 1cm, text height = 1.5ex, text depth = 0.25ex]
  % Circuit has 3 inputs
  \node (x) at (0, 0) {$x$};
  \node (y) at (1, 0) {$y$};
  \node (z) at (2, 0) {$z$};

  % First column of AND gates
  \node[twoAnd] at ($(z) + (1, -1)$) (And1) {};
  \node[twoAnd] at ($(z) + (1, -2)$) (And2) {};
  \node[twoAnd] at ($(z) + (1, -3)$) (And3) {};

  % Connect inputs to the AND gates
  \draw (x) |- node[dot]{} (And1.input 1);
  \draw (y) |- node[dot]{} (And1.input 2);
  \draw (x) |- (And2.input 1);
  \draw (z) |- node[dot]{} (And2.input 2);
  \draw (y) |- (And3.input 1);
  \draw (z) |- (And3.input 2);

  % 3-input OR gate
  \node[threeOr] at ($(And2.output) + (1, 0)$) (Or1) {};

  % Connect AND output to OR inputs
  \draw (And1.output) -- ++(0.5, 0) |- (Or1.input 1);
  \draw (And2.output) -- (Or1.input 2);
  \draw (And3.output) -- ++(0.5, 0) |- (Or1.input 3);

  % Draw and label final output
  \draw (Or1.output) -- ++(2, 0) node[above left] {$f(x, y, z)$};
\end{tikzpicture}

```

**Figure 6:** A logic circuit which computes the majority function:  $f(x, y, z) = \text{true}$  if two or more of its inputs are true; otherwise it is false.

indicates a two-input gate in which each input is “normal”; i.e., not inverted. The node style named `dot` will be used to produce a small, filled circle to indicate an electrical connection.

Moving on to Figure 6, let’s examine how this circuit is specified. The `tikzpicture` environment specifies  $x$  and  $y$  units of 1 cm each, which could easily be adjusted to obtain a stretched or compressed variant. The text height and depth which appears is

present to ensure that the three inputs,  $x$ ,  $y$ , and  $z$  are typeset on the same baseline.

Within the body of the environment, the comments provide some guideposts for understanding how the circuit is drawn. A few additional remarks may be helpful. Inputs  $x$ ,  $y$ , and  $z$  are placed at absolute coordinates, spaced one centimeter apart horizontally. The three `and` gates are placed relative to the  $z$  input: “over 1, down 1”, “over 1, down 2”,

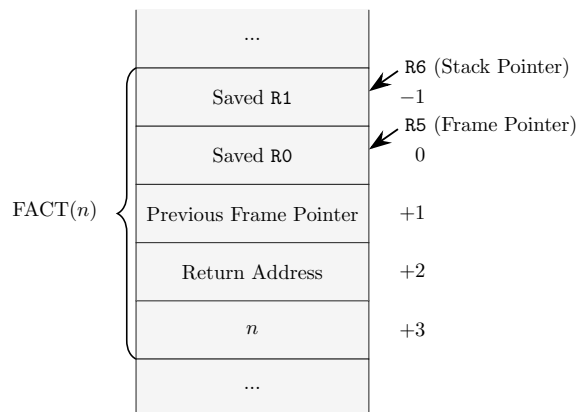


Figure 7: A simple stack frame produced by the `drawstack` package.

and “over 1, down 3”. The `or` gate is placed in a similar way.

The interconnections are simply vertical and horizontal line segments, specified with the `--` and `|-` line drawing capabilities of `TikZ`: a single line segment is produced by `--`, whereas `|-` draws an ell-shape consisting of a vertical segment followed by a horizontal one.

To illustrate specific input values, we could use a `draw` option to easily add colors (e.g., red for `false`, blue for `true`) to this circuit. For example, replacing the `\draw` commands with `\draw[red]` will change the color of the lines and dots.

## 5 Stacks and stack frames

To understand function calls at the machine-language level, visualization of stacks and stack frames within memory is key. The package `drawstack` [11] provides a means to illustrate memory with annotations and explicit links. Refer to Figure 7 for an example of a simple stack frame.

This package provides the `drawstack` environment. Each frame is enclosed within a pair of commands: `\startframe` and `\finishframe`. The argument provided to `\finishframe` specifies the brace label. The brace itself groups all of the cells within a given frame. In this first example, just one frame appears, but any number of frames may be produced.

Each component of a frame consists of a *cell*, an optional *comment*, and an optional cell *pointer*. The comment and pointer appear to the cell’s right. Cell comments are useful for memory addresses or offsets, for example. In this example, we are using

```
\begin{drawstack}
\startframe
\tikzset{>={Stealth[width = 2mm,
length = 3mm]}}
\cell{Saved \texttt{R1}}
\cellcom{-$-1$}
\cellptr{\texttt{R6} (Stack Pointer)}
\cell{Saved \texttt{R0}}
\cellcom{${\phantom{+}}0$}
\cellptr{\texttt{R5} (Frame Pointer)}
\cell{Previous Frame Pointer}
\cellcom{+1$}
\cell{Return Address}
\cellcom{+2$}
\cell{n}
\cellcom{+3$}
\finishframe{FACT($n$)}
\end{drawstack}
```

`\tikzset` to specify an arrowhead of a customized size. Other features of `TikZ` can be used to add additional arrows, change colors, and so on. Figure 8 provides a glimpse of the possibilities.

## 6 Displaying fields of bits

In computer systems courses students are introduced to machine language instructions, written in binary. These instructions are subdivided into fields of bits and given mnemonic names. Similarly, memory maps, network protocols, and file formats all consist of fields of data. The package `bytefield` [12] can be used to show these layouts, and provides a wide variety of options to do so.

The `bytefield` environment may be used to create diagrams similar in format to Figure 9. The argument, 13 in this case, specifies the width of the figure in bits. The `bitwidth` option controls the amount of horizontal space given to each bit. Within this environment, `\bitbox` is used to add a field one or more bits wide in a single row. The command `\wordbox` adds a field that fills one or more rows deep. Double slashes are used to end a row in a `bytefield` diagram, much like a `tabular` environment.

Figure 10 illustrates how `bytefield` can be used to format a machine instruction; in this example, from a hypothetical computer called LC-3 [13]. The indices are added with `\bitheader`, while the `endianness` option reverses the order of the indexing.

Groups can be used to add labels that span multiple rows, and can be placed on the left or the right of the diagram. Groups do not have to nest properly, unlike most environments. Use the `leftcurly`

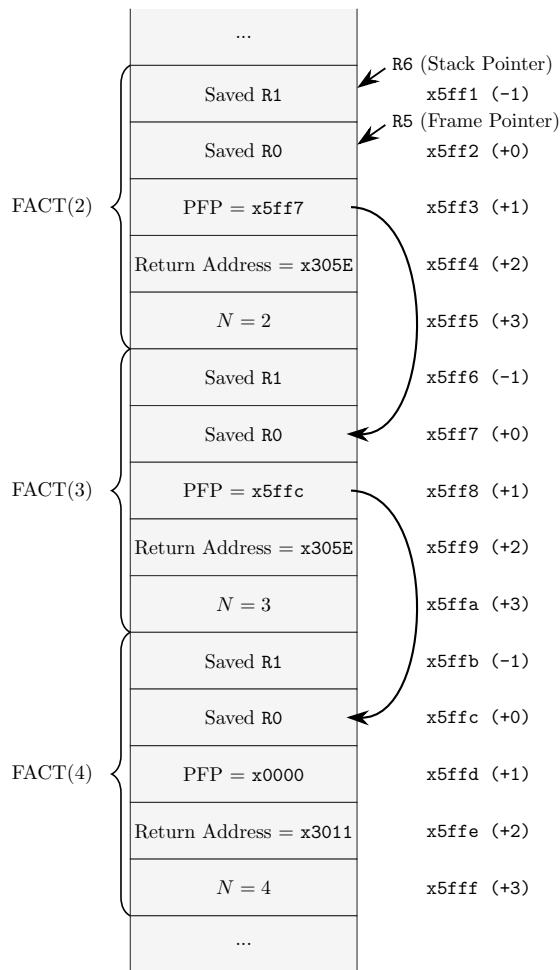


Figure 8: Linked stack frames corresponding to a recursive computation of FACT(4).

```
\begin{bytefield}[bitwidth = 1.5em]{13}
  \bitbox{1}{bit}
  \bitbox{4}{nybble}
  \bitbox{8}{byte}\\
  \wordbox{1}{one row}\\
  \wordbox{2}{two rows with
    additional text so that it will
    wrap around}
\end{bytefield}
```

bit	nybble	byte
one row		
two rows with additional text so that it will wrap around		

Figure 9: An example of basic bytefield commands, and its output.

```
\begin{bytefield}[bitwidth = 1.2em,
                 leftcurly = .]{16}
  \bitheader[endianness = big]{0-15}\\
  \begin{leftwordgroup}{ADD}
    \bitbox{4}{0001}
    \bitbox{3}{DR}
    \bitbox{3}{SR1}
    \bitbox{1}{0}
    \bitbox{2}{00}
    \bitbox{3}{SR2}
  \end{leftwordgroup}
\end{bytefield}
```

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD	0001		DR		SR1		0	00		SR2						

Figure 10: Formatting a machine instruction using bytefield.

option to set the type of brace used to highlight the group. In the case of Figure 10, no brace was used.

When indices are shown it might be preferable to have each bit centered under its index. The `\bitboxes` command accomplishes this, as shown in Figure 11. The first argument to `bitboxes` specifies the number of “symbols” to place at each bit location. To remove the internal lines within a field, use `\bitboxes*`.

By default, `bytefield` vertically centers bit and word boxes without respecting the baseline of their contents. The `bytefield` documentation gives a technique for adjusting this, as shown in Figure 11, by using a `\raisebox` command within `\bytefieldsetup` to initialize `boxformatting`.

Extra space can be added between rows, similar to a `tabular` environment. The `bytefield` documentation also gives a technique for filling a `bitbox` to gray it out. However, it is not compatible with the technique given for aligning text on the baseline, so the box formatting needs to be reset, as also shown in Figure 12.

### 7 Automata

Central to the theory of computation is the concept of various types of automata, such as finite-state machines, pushdown machines, and Turing machines. These abstract computing devices are typically presented as directed graphs with labeled edges. These kinds of diagrams are good for understanding the *static* aspect of these machines.

How might we gain further appreciation of the *dynamic* aspect of automata — that is, the nature of how these machines process input strings? One way is to utilize a program like JFLAP [14], a Java-based

```

\newlength{\maxheight}
\setlength{\maxheight}{\heightof{W}}
\newcommand{\baselinealign}[1][\maxheight]
  {\centering\raisebox{0pt}{#1}[0pt]}

\begin{bytefield}[bitwidth = 1.2em,
  leftcurly = .]{16}
\bitheader[endianness = big]{0-15}\
\bytefieldsetup{boxformatting=
  \baselinealign}%
\begin{leftwordgroup}{BR}
  \bitboxes{1}{0000}
  \bitboxes*{1}{nzp}
  \bitbox{9}{PCoffset9}
\end{leftwordgroup}
\end{bytefield}

```

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BR	0	0	0	0	n	z	p	PCoffset9								

Figure 11: Baseline alignment of bit boxes.

```

\begin{bytefield}[bitwidth = 1.5em,
  leftcurly = .]{16}
\bitheader[endianness = big]{0-15}\
\bytefieldsetup{boxformatting=
  \baselinealign}%
\begin{leftwordgroup}{BR}
  \bitboxes*{1}{0000}
  \bitboxes*{1}{nzp}
  \bitbox{9}{PCoffset9}
\end{leftwordgroup}\[1ex]

\bytefieldsetup{boxformatting=
  {\centering}}
\begin{leftwordgroup}{RSV}
  \bitboxes*{1}{1101}
  \bitbox{12}{\color{lightgray}
  \rule{\width}{\height}}
\end{leftwordgroup}
\end{bytefield}

```

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BR	0	0	0	0	n	z	p	PCoffset9								
RSV	1 1 0 1															

Figure 12: Adding space between two machine instructions and filling a bitbox with color.

formal language and automata package now often used in computer science education.

Using JFLAP, we can build automata using a convenient GUI interface, then explore their behavior on various input strings. In addition, we can experiment with constructions used in proofs, such as constructing an equivalent deterministic automaton given a non-deterministic one.

What options exist for incorporating a JFLAP-based automaton in a  $\text{\LaTeX}$  document? At present, the software exports in JPG, PNG, GIF, BMP and SVG formats. Unfortunately, these are less than ideal, as Figure 13 shows: fonts do not match those used in the document, “true” subscripts are not used, color may not be desired, and it is difficult to achieve accurate placement of the circular nodes, since JFLAP does not use a “snap to” grid for node layout.

Since TikZ has a library for drawing automata, we can imagine an export option from JFLAP which produces appropriate TikZ code which could then be included in a  $\text{\LaTeX}$  document. Such an option would be a vector-based format and allow for  $\text{\TeX}$ -based markup, thereby eliminating most of the undesirable effects of using one of the bitmap formats now available. The possibility of inaccurately placed nodes would continue to be a problem.

Although not directly incorporated into the JFLAP software, the script `jflap2tikz` [8] achieves the goal of presenting automata created with JFLAP using TikZ-based graphics. The TikZ code output by `jflap2tikz` for our example automaton is shown in Figure 14, and Figure 15 shows the resulting processed output. The TikZ code produced by `jflap2tikz` is human-readable and thus can be further edited, if desired. The script allows for a “snap to” style grid which can improve alignment of the nodes. A large grid spacing corresponds to a coarser grid. Figure 16 shows the result of using such a grid on the example finite-state machine.

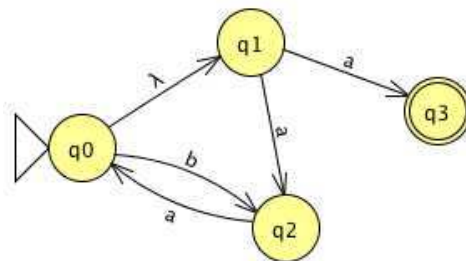


Figure 13: An example finite-state machine excerpted from [15], as exported by JFLAP using PNG format.

```

\begin{tikzpicture}[>={Stealth[width = 6pt, length = 9pt]},
    accepting/.style = {double distance = 2pt,
        outer sep = 1pt + \pgflinewidth},
    shorten >= 1pt,
    auto]
\draw (62pt, -114pt) node[state, initial, initial text =](0){$q_{0}$};
\draw (163pt, -51pt) node[state](1){$q_{1}$};
\draw (184pt, -162pt) node[state](2){$q_{2}$};
\draw (275pt, -92pt) node[state, accepting](3){$q_{3}$};
\path[->] (0) edge[bend left] node{b}(2);
\path[->] (2) edge[bend left] node{a}(0);
\path[->] (1) edge node{a}(2);
\path[->] (0) edge node{$\lambda$}(1);
\path[->] (1) edge node{a}(3);
\end{tikzpicture}

```

Figure 14: Result of processing the source file for Figure 13 with the `jflap2tikz` script (slightly edited for space).

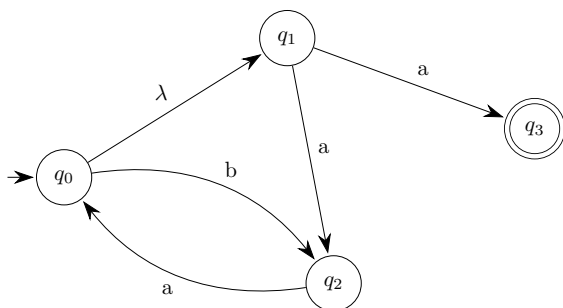


Figure 15: The finite-state machine from Figure 13, converted into TikZ and then processed as usual.

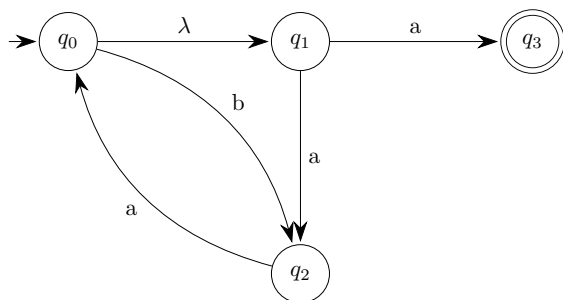


Figure 16: The finite-state machine from Figure 13, converted with a grid spacing of 100.

## 8 Trees

Forests and trees appear in a wide variety of forms in many areas of computer science. Drawing trees with GUI-style graphics software can be awkward and may yield imperfect results. Fortunately, there are a number of T<sub>E</sub>X-based approaches. One of these

involves the use of the `forest` package [17], which produces excellent results.

This package provides the `forest` environment, wherein a tree can be defined using *bracket notation*, a well-known syntax among linguists which captures the recursive nature of a tree. In this notation, a tree with a single root node  $r$  is represented by  $[r]$ . If a tree has more than a single node, then it has a root node  $r$  and  $n$  nonempty subtrees  $T_1, T_2, \dots, T_n$ . The bracket representation for this tree is  $[r \text{ br}(T_1) \text{ br}(T_2) \dots \text{ br}(T_n)]$ , where  $\text{br}(T)$  is the bracket representation of tree  $T$ .

Figure 17 illustrates how this package can be used to draw a tree. In this example, 6 is the root with two subtrees,  $T_1$  and  $T_2$ , so the bracket notation for this tree is of the form  $[6 \text{ br}(T_1) \text{ br}(T_2)]$ . Using this same pattern to expand  $\text{br}(T_1)$  and  $\text{br}(T_2)$ , we eventually obtain the bracketed form shown in the figure. Notice that spaces and newlines can be introduced to assist with readability and to help make clear the structure of the tree.

Another common structure in computer science is the *binary tree*, in which every node has at most two children, referred to as left and right children. The tree depicted in Figure 17 is not a binary tree, since it is not clear whether the leaf node 1 is a left child or a right child.

Indicating whether a node is a left or right child can be accomplished with the `phantom` option which reserves space for a node, but doesn't draw an edge to it. Phantom nodes may occur anywhere in the tree; when one is used at the root it produces a forest. Figure 18 shows two phantom nodes, one as the left child of 8, the other as the right child of 7.



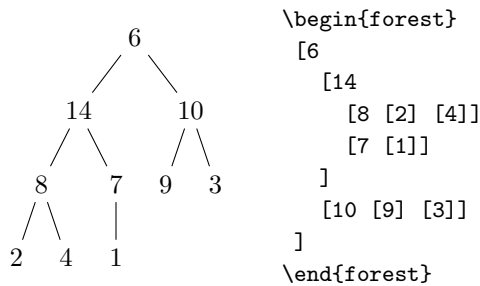


Figure 17: A tree produced by the forest package.

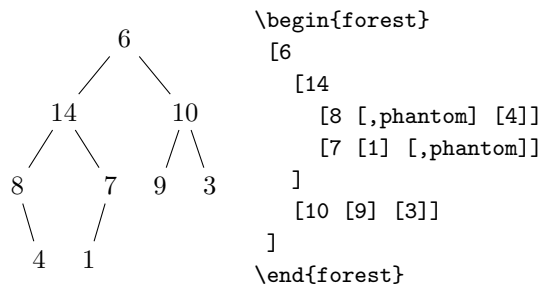


Figure 18: Use of phantom to force node alignment.

Trees are commonly drawn with circular nodes. This, too, is possible with the forest package since it is built upon *TikZ*. Thus, options from *TikZ* can be used to alter the result, and can be applied to individual nodes, a subtree, or the entire tree. To add circles to the tree of Figure 18, a first attempt would be to apply this option to the entire tree:

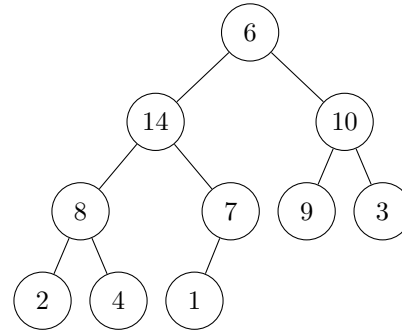
```
for tree = {draw, circle}
```

This would draw a circle at each node of the tree, but the size of each circle would depend on the dimensions of the text at each node—giving circles of varying sizes.

Specifying an appropriate minimum circle size, as shown in Figure 19, solves that problem, producing a tree with a uniform appearance.

It is also useful to be aware of the `fit` option. Figure 20 shows the same binary search tree formatted in two ways. On the left is the default result, also known as the `tight` fit, whereas the tree on the right shows the result with the `band` fit. Compact trees might be nice in many situations, but the wider tree on the right is the one typically encountered for search trees.

A red-black tree [3] is another type of binary tree, where the color of each node is important. Figure 21 displays a red-black tree. The text has been set to white, and the default color for nodes is black. The red nodes have the default color overridden with the option `fill = red` (printed in gray for this article).



```

\begin{forest}
for tree = {draw, circle,
node options = {minimum width = 5ex}}
[6
[14
[8 [2] [4]]
[7 [1] [,phantom]]
]
[10 [9] [3]]
]
\end{forest}

```

Figure 19: Uniform node size obtained by setting a minimum node width.

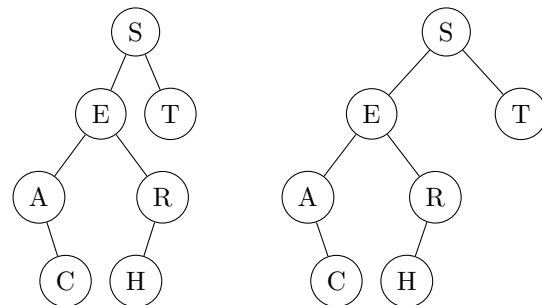
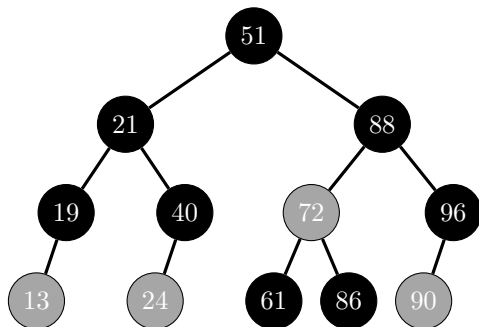


Figure 20: Compact tree on left (`fit = tight`); wider on right (`fit = band`).

## 9 Grammars and parse trees

Before looking at an example of a parse tree, we take a short detour into grammars. A context-free grammar is a formal system which describes a language as a set of rules.

The `syntax-mdw` package [18] may be used to typeset the syntax rules for a given language. A `grammar` environment is used, and produces nicely formatted output, as demonstrated in Figure 22. The `\alt` command is used to specify alternative rules, and when typeset produces the `|` symbol, pronounced “or”. Setting the value of `\grammarindent` determines the amount to indent each of the alternatives in the grammar definition.



```

for tree = {fit = band, circle, draw,
            fill = black, text = white,
            edge = {black, very thick}}
[51
 [21
 [19 [13, fill = red] [,phantom]]
 [40 [24, fill = red] [,phantom]]
 ]
 [88
 [72, fill = red [61][86]]
 [96 [90, fill = red] [,phantom]]
 ]
 ]

```

Figure 21: An example of a red-black tree.

```

⟨expr⟩ ::= ⟨expr⟩ '+' ⟨expr⟩
         | ⟨expr⟩ '-' ⟨expr⟩
         | ⟨expr⟩ '*' ⟨expr⟩
         | ⟨expr⟩ '/' ⟨expr⟩
         | '(' ⟨expr⟩ ')'
         | '-'⟨expr⟩
         | 'id'

```

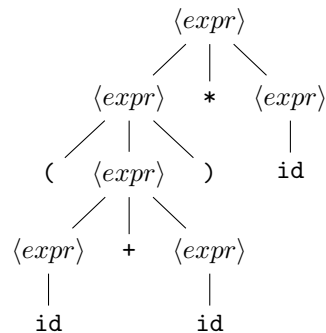
```

\setlength{\grammarindent}{5em}
\begin{grammar}
  <expr> ::= <expr> '+' <expr>
    \alt <expr> '-' <expr>
    \alt <expr> '*' <expr>
    \alt <expr> '/' <expr>
    \alt '(' <expr> ')'
    \alt '-'<expr>
    \alt 'id'
\end{grammar}

```

Figure 22: The syntax rules of a simple grammar.

A parse tree represents the syntactic structure of a string according to some context-free grammar. Given the grammar in Figure 22, a parse tree can be used to represent an expression such as  $(a + b) * c$  as shown in Figure 23.



```

% Expression: (a+b)*c
\newcommand{\E}{$\langle expr \rangle$}
\newcommand{\id}{\texttt{id}}
\newcommand{\plus}{\texttt{+}}
\newcommand{\mult}{\texttt{*}}
\newcommand{\lpar}{\texttt{(}}
\newcommand{\rpar}{\texttt{)}}
\begin{forest}
[\E
 [\E
 [\lpar
 [\E
 [\E
 [\E [\id]] [\plus] [\E [\id]]
 ]
 ]
 ]
 ]
 ]
 ]
 ]
 [\mult]
 [\E [\id]]
 ]
\end{forest}

```

Figure 23: A parse tree for the expression  $(a + b) * c$ , using the grammar from Figure 22.

## 10 Combinatorial graphs

In graph theory and similar courses, combinatorial graphs must be produced in great numbers. The package `tkz-graph` [7] provides a variety of styles and macros for creating high-quality representations of graphs. The documentation is available only in French, but is so abundantly illustrated with well-done examples that it is accessible to those with little or even no knowledge of French.

Figure 24 displays several of the options available when drawing graphs. To select the vertex style (from a list of ten possibilities, for example `Simple`, `Classic`, or `Shade`), set `vstyle` to your choice using the `\GraphInit` command.

`\SetUpVertex` can be used to set such options as the position of the label relative to the node (`Lpos`), the distance of the label from the node (`Ldist`), and whether the label is outside of the node (`LabelOut`).

```

\begin{tikzpicture}
  % Select vertex style
  \GraphInit[vstyle = Classic]
  % Indicate vertex color
  \SetUpVertex[FillColor = gray!30]
  % Set vertex size relative to label
  \renewcommand*\VertexInnerSep}{3pt}
  % Position of vertex labels
  \SetVertexLabelIn
  % Establish Line width
  \tikzset{EdgeStyle/.append style =
           {line width = 2pt}}

  % --- tkz-graph commands here ---
\end{tikzpicture}

```

Figure 24: Details of initializing values for a graph.

Alternately, macros such as `\VertexInnerSep` can be redefined. Also, several macros are provided to modify default options, for example, `\SetVertexLabelIn`. And finally, `\tikzset` can be used to modify both the vertex and edge styles.

Figure 25 illustrates the format of a simple graph using relative positioning to place the vertices. Vertex A is established, then all the other vertices are relative to A or the position of another vertex. It is of interest to note that the distance given to combined directions such as `\SOEA`, is applied in *both* directions. Thus `\SOEA[unit = 1](A){C}` results in vertex C being one unit south and one unit east of vertex A. Recall that this code resides in the `tikzpicture` environment following the code given in Figure 24. Hence, native TikZ commands such as

```
\draw[help lines](0, 0) grid (4, -4);
```

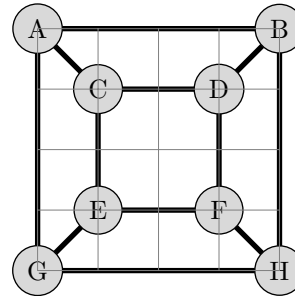
can be issued here.

An alternative way to place the vertices can be seen in Figure 26. Here the vertices are placed using absolute coordinates, and all edges can be included in the single `Edges` macro.

By utilizing `\tikzset`, it is possible to design a new vertex style, as shown in Figure 27. Here the shape, size, and color have been redefined.

It is also possible to modify the vertex style within the `\SetVertexSimple` macro, as shown in Figure 28, where the shape, fill color, size and line have all been specified.

As Figure 29 shows, creating directed edges is as simple as setting the edge style. To improve the default arrowheads, `\tikzset` is used to select the `Stealth` shape, with the width and height enlarged



```

% Two nested squares of vertices
\Vertex{A}          \EA[unit = 4](A){B}
\SO [unit = 4](B){H} \WE[unit = 4](H){G}
\SOEA[unit = 1](A){C} \EA[unit = 2](C){D}
\SO [unit = 2](D){F} \WE[unit = 2](F){E}
% Outer square
\Edge(A)(B)         \Edge(B)(H)
\Edge(H)(G)         \Edge(G)(A)
% Inner square
\Edge(C)(D)         \Edge(D)(F)
\Edge(F)(E)         \Edge(E)(C)
% Connect the squares
\Edge(A)(C)         \Edge(B)(D)
\Edge(H)(F)         \Edge(G)(E)
% Cause a grid to appear
\draw[help lines] (0, 0) grid (4, -4);

```

Figure 25: A simple graph using relative positioning.

```

% Two nested squares of vertices
\Vertex[x = 0, y = 4]{A}
\Vertex[x = 0, y = 0]{G}
\Vertex[x = 1, y = 3]{C}
\Vertex[x = 1, y = 1]{E}
\Vertex[x = 3, y = 3]{D}
\Vertex[x = 3, y = 1]{F}
\Vertex[x = 4, y = 4]{B}
\Vertex[x = 4, y = 0]{H}

% All edges
\Edges(A, B, H, G, A, C, D, F,
       E, C, E, G, H, F, D, B)

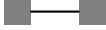
```

Figure 26: The same graph as Figure 25 using absolute placement of vertices.

to improve visibility. Since the edges are now directed, the order of vertices when creating edges becomes important.

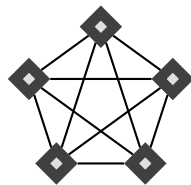
Simple changes can affect the appearance of a graph. For example, changing the vertex style to `Shade` produces a sophisticated-looking graph as seen in Figure 30.

```


\begin{tikzpicture}
  \SetVertexSimple
  \tikzset{VertexStyle/.style = {
    shape      = rectangle,
    fill       = gray,
    inner sep  = 0pt,
    outer sep  = 0pt,
    minimum size = 10pt}}
  \Vertex{A} \EA(A){B}
  \Edge(A)(B)
\end{tikzpicture}

```

Figure 27: Designing your own vertex style, version 1.



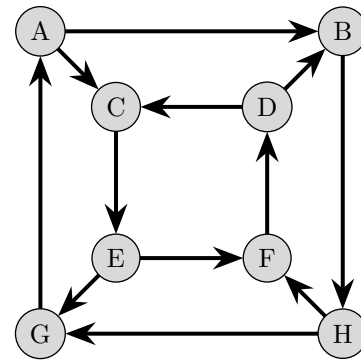
```

\begin{tikzpicture}[rotate = 18]
  \SetVertexSimple[Shape      = diamond,
    FillColor = gray!25,
    MinSize   = 12pt,
    LineWidth = 4pt,
    LineColor = black!75]
  \tikzset{VertexStyle/.append style = {
    inner sep = 0pt,
    outer sep = 2pt}}
  \Vertices{circle}{A, B, C, D, E}
  \Edges(A, B, C, D, E, A, C, E, B, D)
\end{tikzpicture}

```

Figure 28: Designing your own vertex style, version 2.

Figure 31 represents a flow network, with an augmenting path (of flow 4) highlighted. The source and sink nodes have been emphasized with different colors, and the augmenting path edges are wider than other edges and also of a different color. Observe that an edge may be curved using the `bend right` or `bend left` option when setting the edge style. Additional examples from the `tkz-graph` package can be seen in [9].

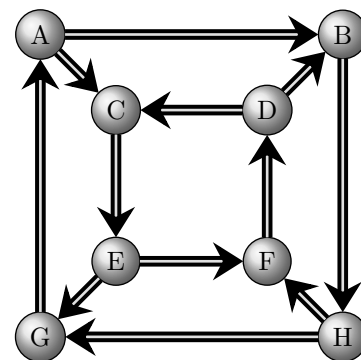


```

% Set arrow type and size
\tikzset{>={Stealth[width = 3mm,
  length = 4mm]}}
% Set edge to arrow
\tikzset{EdgeStyle/.append style =
  {>,line width = 1.5pt}}
% Nested squares of vertices as before
% Outer square - clockwise
\Edges(A, B, H, G, A)
% Inner square - counterclockwise
\Edges(C, E, F, D, C)
% Connect the corners
\Edge(A)(C)
\Edge(D)(B)
\Edge(H)(F)
\Edge(E)(G)

```

Figure 29: Example of a directed graph.

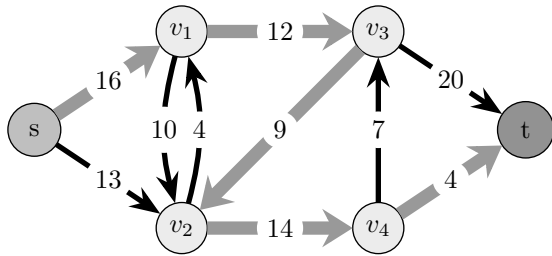


```

% Simple changes:
\GraphInit[vstyle = Shade]
\SetGraphShadeColor
  {gray!25} % ball
  {black}  % edge outline
  {gray!25} % inner edge
% vertices and edges as before

```

Figure 30: A change in vertex style. (Partial L<sup>A</sup>T<sub>E</sub>X code)



```
% Change vertex attributes
\SetUpVertex[FillColor = gray!50,
  InnerSep = 5pt]

% Change edge attributes
\tikzset{LabelStyle/.style =
  {shape = circle, inner sep = 2pt}}
\Edge[color = gray!80,
  lw = 5pt, label = 16](s)(v1)

% Add curve to edges
\tikzset{EdgeStyle/.append style =
  {bend right = 15}}
\Edge[lw = 2pt, label = 10](v1)(v2)
```

**Figure 31:** A sample flow network with changing attributes.

## 11 Summary

As educators in the field of computer science, we find ourselves challenged to produce a wide variety of figures and diagrams. Being able to replicate (or in some cases, exceed) the quality found in textbook presentations is a practical and intrinsically rewarding skill. Fortunately, the  $\text{\TeX}$  community has provided a wealth of resources which can be brought to bear on this problem. We hope that the examples and explanations provided in this paper will encourage others to explore these and other packages further.

## References

- [1] David Carlisle. Guide to graphics in  $\text{\LaTeX}$ . <http://ctan.org/pkg/graphicx>.
- [2] Thomas Cormen. The `clrscode3e` package: Typesets pseudocode as in Introduction to Algorithms. <http://ctan.org/pkg/clrscode3e>.
- [3] Thomas Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein. *Introduction to Algorithms, 3<sup>rd</sup> Edition*. The MIT Press, 2009.
- [4] Michel Goossens, Frank Mittelbach, Sebastian Rahtz, and Denis Roegel. *The  $\text{\LaTeX}$  Graphics Companion, 2<sup>nd</sup> Edition*. Addison-Wesley Professional, 2007.

- [5] Carsten Heinz, Brooks Moses, and Jobst Hoffmann. The listings package: Typeset source code listings using  $\text{\LaTeX}$ . <http://ctan.org/pkg/listings>.
- [6] John Hobby. MetaPost. <http://tug.org/metapost>.
- [7] Alain Matthes. The `tkz-graph` package: Draw graph-theory graphs. <http://ctan.org/pkg/tkz-graph>.
- [8] Andrew Mertz and William Slough. The `jflap2tikz` script: Convert JFLAP files to TikZ. <http://ctan.org/pkg/jflap2tikz>.
- [9] Andrew Mertz and William Slough. Graphics with PGF and TikZ. *TUGboat*, 28(1):91–99, 2007. <http://tug.org/TUGboat/tb28-1/tb88mertz.pdf>.
- [10] Frank Mittelbach, Michel Goossens, Johannes Braams, and David Carlisle. *The  $\text{\LaTeX}$  Companion, 2<sup>nd</sup> Edition*. Addison-Wesley Professional, 2004.
- [11] Matthieu Moy. The `drawstack` package: Draw execution stacks. <http://ctan.org/pkg/drawstack>.
- [12] Scott Pakin. The `bytefield` package: Create illustrations for network protocol specifications. <http://ctan.org/pkg/bytefield>.
- [13] Yale Patt and Sanjay Patel. *Introduction to Computing Systems: From bits & gates to C & beyond*. McGraw-Hill, 2003.
- [14] Susan H. Rodger. JFLAP: Java Formal Languages and Automata Package. <http://www.jflap.org>.
- [15] Susan H. Rodger and Thomas W. Finley. *JFLAP: An Interactive Formal Languages and Automata Package*. Jones and Bartlett Learning, 2006.
- [16] Till Tantau. The PGF package: Create PostScript and PDF graphics in  $\text{\TeX}$ . <http://ctan.org/pkg/pgf>.
- [17] Sašo Živanović. The `forest` package: Drawing (linguistic) trees. <http://ctan.org/pkg/forest>.
- [18] Mark Wooding. The `syntax-mdw` package: Typeset syntax descriptions. <http://ctan.org/pkg/syntax-mdw>.

◇ Andrew Mertz, William Slough  
and Nancy Van Cleave  
Department of Mathematics and  
Computer Science  
Eastern Illinois University  
Charleston, IL 61920  
`aemertz (at) eiu dot edu`,  
`waslough (at) eiu dot edu`,  
`nkvanleave (at) eiu dot edu`

## Dynamic documents

David Allen

### Abstract

The term *dynamic document* covers a wide assortment of documents; see Wikipedia for a general definition. The discussion here involves a situation where the document is a report including statistical analyses and graphical displays based on data. The data are continually being augmented or replaced. What is needed is a way to automate the revision of the document when the data changes. Our approach here is to use R to do the statistical calculations and graphics, `tikzDevice` (an R package) to output the graphics to  $\LaTeX$ , and `knitr` (also an R package) to process an input file to a  $\LaTeX$  file.

### 1 The Kentucky Senate race

On November 4, 2014, the Commonwealth of Kentucky will elect a United States Senator. The candidates are Alison Lundergan Grimes and Mitch McConnell. This race has high national impact and is closely watched. A poll yields the number of people in a sample, from a population of potential voters, favoring each candidate. The proportion of the sample favoring Alison (or Mitch) is reported. However, this provides no indication of the sampling variability.

The parameter of interest is the population proportion favoring Alison. A *credible interval* is such that the parameter lies within the interval with high probability. A credible interval is a more informative mode of presentation, as it conveys the uncertainty of knowledge about the parameter. Calculation of the credible interval is the statistical analysis portion of the report.

Denote the proportion of the population favoring Alison by  $p$ . The first step in calculating a credible interval is finding the posterior density function of  $p$  given the sample results. One needs to select a level of credibility. The value 0.95 has a strong tradition and is used here. The 0.95 credibility interval is an interval  $(p_1, p_2)$  where  $P(p_1 < p < p_2) = 0.95$ . There are multiple intervals satisfying the probability statement. The interval having minimal length is usually used.

An example assuming a sample with 55 favoring Alison and 45 favoring Mitch is shown in fig. 1. The 0.95 credible interval (0.4528, 0.6428) is the base of the shaded region. A report might look like:

A current poll produced 55 potential voters favoring Alison and 45 favoring Mitch. These results give a 0.95 credible interval for the proportion favoring Alison of (0.4528, 0.6428).

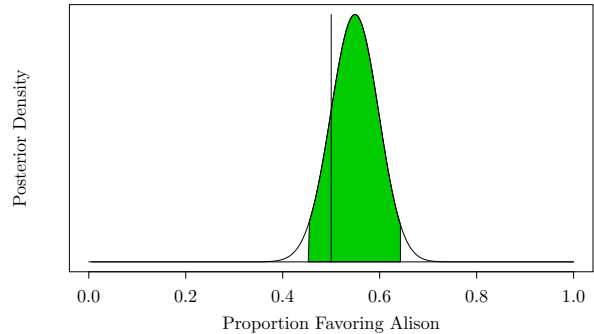


Figure 1: Graph showing credible interval.

The report might also include the above graph (fig. 1).

The preceding graphic and the credible interval were produced with R. The output from R was then transcribed to a  $\LaTeX$  file to produce the “report”. Polling will be a continuing activity from now until election day. Rerunning R and cutting and pasting output into a  $\LaTeX$  document is tedious and error prone. Subsequent sections show how to automate the process.

### 2 TikZ graphics

`TikZ` is a graphics package used in conjunction with  $\TeX$ . It is included with most distributions of  $\TeX$ , or may be downloaded at <http://sourceforge.net/projects/pgf/>. A large selection of examples of `TikZ` graphics are posted at <http://www.texample.net/tikz/examples/>.

I think it likely that most *TUGboat* readers are familiar with *TikZ*. I give just two examples I have composed. The graphic in fig. 2 was hand-coded in Sketch (<http://www.frontiernet.net/~eugene.ressler/>), and then processed into `TikZ`, and fig. 3 was written directly in `TikZ`.

### 3 An overview of R

R is a language and environment for statistical computing and graphics. Its home page is <http://www.r-project.org>. R is a free software project. It compiles and runs on a wide variety of Unix platforms and similar systems, including FreeBSD, GNU/Linux,

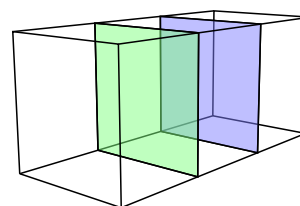


Figure 2: Graphic made in Sketch and exported to `TikZ`.

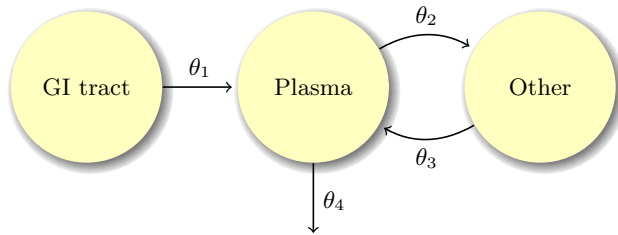


Figure 3: Hand-coded TikZ example.

and Mac OS X, as well as Windows. R is often the vehicle of choice for research in statistical methodology, and it provides an open source route to participation in that activity.

R provides a wide variety of statistical techniques including linear and nonlinear modeling, classical statistical tests, time-series analysis, classification, and clustering. R is highly extensible and contains a rich collection of graphical techniques. One of R's strengths is the ease with which well-designed publication-quality plots can be produced, including mathematical symbols and formulas where needed. Defaults for the minor design choices in graphics have been carefully considered, but the user retains full control.

#### 4 tikzDevice

The `tikzDevice` package enables L<sup>A</sup>T<sub>E</sub>X-ready output from R graphics functions. This is done by producing code that can be understood by the TikZ graphics language. All text in a graphic output with the `tikz()` function will be typeset by L<sup>A</sup>T<sub>E</sub>X and therefore will match whatever fonts are currently used in the document. This also means that L<sup>A</sup>T<sub>E</sub>X mathematics can be typeset directly into labels and annotations. Graphics produced this way can also be annotated with custom TikZ commands. An example R graphic output using `tikzDevice` is shown in fig. 4. The program that produced the graph is

```
setwd("~/tug2014")
source("quadratic-data.R")
source("quadratic-graph.R")
```

The `source` command executes the statements in the named file. Here I group code into small parts to focus discussion.

The file `quadratic-data.R` contains the data generation code:

```
x <- (0:100)/10
y <- 10 + (x-5)^2
```

R formulas are different from standard mathematical formulas. A function call operates on every element of a vector. When vectors of different lengths are

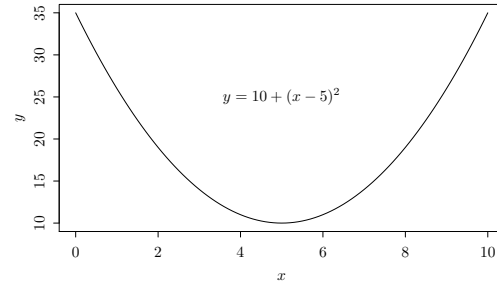


Figure 4: Example output using `tikzDevice`.

added or subtracted, the shorter one is recycled in an effort to make them the same length.

The file `quadratic-graph.R` contains the graphics code:

```
require(tikzDevice)
tikz("quadratic-graph.tex",standAlone=FALSE,
     width=4.5, height=2.5)
par(mex=0.6, mar=c(4.5,5,0,0)+0.1)
plot(x, y, type='l', xlab="$x$", ylab="$y$")
text(5, 25, "$y = 10+(x-5)^2$")
dev.off()
```

Most of this code is understandable by comparison to the resulting graph. An exception might be the `par` function used to change graphic parameters from their default values. The ones used here are:

Arg	Description
<code>mex</code>	A character size expansion factor used to describe coordinates in the margins of plots.
<code>mar</code>	Vector of the form <code>c(bottom, left, top, right)</code> which gives the number of lines of margin to be specified on the four sides.

#### 5 Implementation

This section implements a dynamic document that facilitates reporting the current status of the race between Alison and Mitch. The document has a title, a graph of the posterior density, and a short statistical report. The data file for this senate race is `senate.dat` and contains just two numbers, the number in the sample that favor Alison, and the number that favor Mitch. This two-number file can be updated with an editor. For more complicated situations a program might update the data file.

Knitr is an R package containing a function `knit`, which takes a file name with an extension `.Rnw` as an argument. An `.Rnw` file is like a L<sup>A</sup>T<sub>E</sub>X file with interspersed R chunks. The output is a pure L<sup>A</sup>T<sub>E</sub>X file containing the output from running the R chunks. Documentation for `knitr` is available online and in Yihui Xie's book [1].

R is an implementation of a language S. There is a macro `\Sexpr()`, for *S expression*, that may be placed in the  $\TeX$  portion of the file. `\Sexpr()` takes an R expression as an argument. The expression is evaluated, converted to text, and passed into the  $\LaTeX$  output. The content of `senate.Rnw` is

```
\documentclass[12pt]{article}
\begin{document}
<<setup,echo=FALSE>>=
source("chunk1.R")
@
\title{Alison Versus Mitch}
\author{David Allen\University of Kentucky}
\maketitle
\thispagestyle{empty}
%
<<params,echo=FALSE>>=
source("chunk2.R")
@
A poll released July 28, 2014 produced
\Sexpr{a-2} potential voters favoring Alison and
\Sexpr{b-2} favoring Mitch.
These results give a \Sexpr{level} credible
interval for the proportion favoring Alison of
(\Sexpr{p1}, \Sexpr{p2}).
%
The posterior density function, with the area
over the credible interval shaded, is
<<label="density",dev='tikz',echo=FALSE,
  fig.width=4,fig.height=2.75,
  fig.align='center'>>=
source("chunk3.R")
@
\end{document}
```

The R chunks have a header of the form

```
<< ... >>=
```

where commands are inside the double brackets, on a single line (the line breaks above in the `<<label...` above are editorial). A chunk ends with an `@`.

The content of `chunk1.R` is

```
setwd("~/tug2014")
interval.length <- function(p1,a,b,level=0.95)
{
  q <- qbeta(1-level, a,b)
  if( p1 > q ) return(1 - q)
  if( p1 < 0 ) return(qbeta(level, a, b))
  p2 <- qbeta(pbeta(p1, a, b) + level, a, b)
  return(p2-p1)
}
```

The function `interval.length` is a function I wrote that gives the length of an interval starting at  $p_1$ . The posterior distribution of  $p$  is the beta distribution. `qbeta` is a built-in function giving quantiles of the beta distribution.

The content of `chunk2.R` is

```
vote <- vector(mode="numeric")
vote <- scan(file="senate.dat")
a <- vote[1] + 2
b <- vote[2] + 2
level <- 0.95
p1 <- optimize(f = interval.length,
  interval = c(0, qbeta(1-level, a,b)),
  a=a, b=b, level=level)$minimum
p2 <- qbeta(pbeta(p1, a, b) + level, a, b)
```

Here the built-in `optimize` function is used to find the value of  $p_1$  associated with the shortest interval. Then the corresponding  $p_2$  is computed.

The content of `chunk3.R` is

```
left <- (1:80)/80*p1
interval <- p1 + (1:80)/80*(p2-p1)
right <- p2 + (1:80)/80*(1-p2)
domain <- c(left, interval, right)
range <- dbeta(domain, a, b)
par(mex=0.6, mar=c(4.5,5,0,0)+0.1)
plot(c(0,1), c(0, max(range)), type="n",
  xlab="Proportion Favoring Alison",
  ylab="Density", yaxt='n')
polygon(c(interval, p2, p1),
  c(dbeta(interval, a, b), 0, 0), col=27)
lines(domain, range); lines(c(0,1),c(0,0))
lines(c(0.5,0.5),c(0,max(range)))
```

This code produces the graph.

After each data update, run the following command in a terminal:

```
Rscript -e "library(knitr);knit('senate.Rnw')"
```

A  $\LaTeX$  file is produced that can be processed in the usual ways. `Rscript` is just the command line version of R. The `-e` option means the following are statements to be run, as opposed to a file containing statements.

I conclude with an exercise. A poll was released on July 28, 2014 (the first full day of the conference in Portland) showing 321 for Alison and 336 for Mitch. I invite you to prepare a new data file, `senate.dat`, containing

```
321 336
```

Then *knit* the `.Rnw` file and  $\LaTeX$  the resulting `senate.tex`.

## References

- [1] Yihui Xie. *Dynamic Documents with R and knitr*. Chapman & Hall/CRC Press, 2014. ISBN 978-1482203530.

◇ David Allen  
University of Kentucky  
david dot allen (at) uky dot edu



## Typography and readability: An experiment with post-stroke patients

Leyla Akhmadeeva and Boris Veytsman

### Abstract

Typography for challenged readers has unique problems. There is a large amount of research about reading by people with impaired vision. Since reading is a complex process, other impairments, for example, cognitive problems, may also influence it. Should a publisher of texts for this audience be aware of this? Which typographical devices must be used for these texts?

In our previous reports we showed that serifs do not influence the readability and understandability of texts by healthy students. In this report we study the readability and understandability of serif and sans-serif texts by post-stroke patients. We discuss the experimental setup and preliminary results.

### 1 Introduction

There is a certain typographic lore about the influence of different typographic elements on readability. For example, every beginner “knows” that text with serifs is read faster than sans serif, and therefore the former should be used for body copy, and the latter for advertisements and slides. Like the remedies of traditional medicine, some of these pieces of knowledge turn out to be corroborated by evidence, while some do not.

Some time ago we discussed an evidence-based approach to the verification of these old wisdoms [5].  $\text{\TeX}$  and friends turned out to be useful in this approach since they provide repeatable controlled typesetting, where we can change a limited number of parameters and study their influence on the results.

One of the first applications of this approach was to test the hypothesis of difference in readability of serif and sans serif fonts. Our experiments with the volunteer students of Bashkir State Medical University [1, 6] showed no discernible difference between these fonts, when either the speed of reading or text comprehension were measured. This result resembles the finding by Legge and Bigelow [3]—font sizes used in the modern and historical typography are within the range of fluent reading. Our experiments indicate that not only font sizes but other typographic features as well are within this range.

These conclusions are based on the experiments with healthy readers. It is still an open question whether subtle typographic differences influence readability for people with different kinds of impairment. A prime example of such a group is post-stroke pa-

## Paratype Serif Paratype Sans

Figure 1: Fonts used (enlarged to show the difference)

tients. It is well-known [4] that post-stroke patients often suffer from reading problems which adversely influence their adaptation. The study of whether these patients can better read serif or sans serif texts has a considerable practical importance: many materials are printed for this audience (instructional texts, drug leaflets, etc.), and the optimal use of typographic devices is clearly desirable.

We have started a study of reading by post-stroke patients. In this paper we report the setup and the first preliminary results.

### 2 Methods

We selected the patients for this study according to the following criteria:

1. Post-stroke patients;
2. Ability to read text;
3. Fluency in Russian language;
4. Absence of dementia;
5. Absence of aphasia.

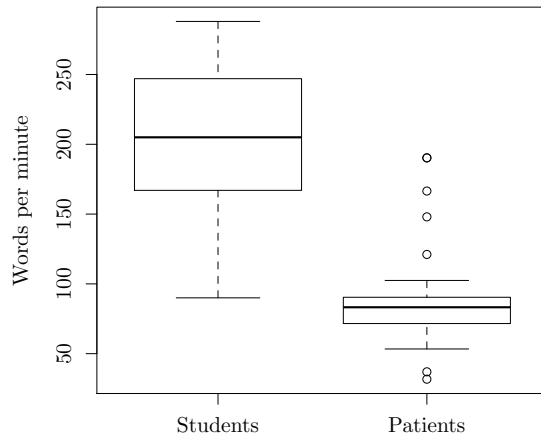
The study’s methods resemble those of our previous works [1, 6]. We typeset four one-page texts in Russian using the Paratype fonts (shown in Figure 1) at 12 pt scaled 0.95 [2]. We measured the reading time of the page by the patients, and then asked them to answer 10 multiple-choice questions about the text (each question had four suggested answers).

Due to ethical considerations, all texts contained information useful for the patients: advice for post-stroke rehabilitation. To facilitate paired comparisons, we asked each patient to read all four texts with an interval of approximately one week between the texts. According to a randomized choice (using <http://www.randomization.com>) half of the patients read odd-numbered texts (1 and 3) in the serif font, and even-numbered texts (2 and 4) in sans serif, while the other half read odd-numbered texts in sans serif, and even-numbered texts in the serif font.

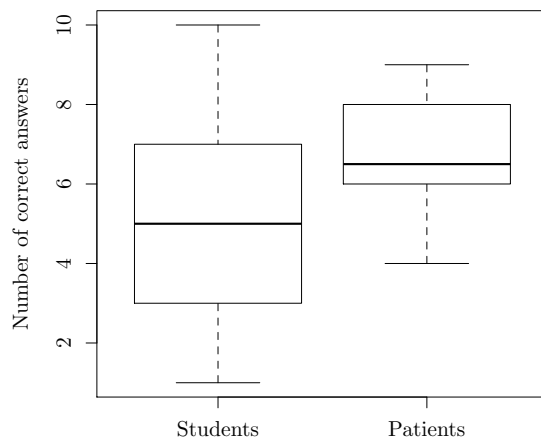
### 3 Preliminary results

At this point we can report the results for  $N = 19$  participants, including 12 males and 7 females, average age  $54 \pm 11$  years.

First, it is interesting to compare their results with those of healthy students [1, 6], who read a different text on the history of science. These comparisons are shown in Figures 2 and 3. We see that



**Figure 2:** Comparison of speed of reading for medical students and post-stroke patients

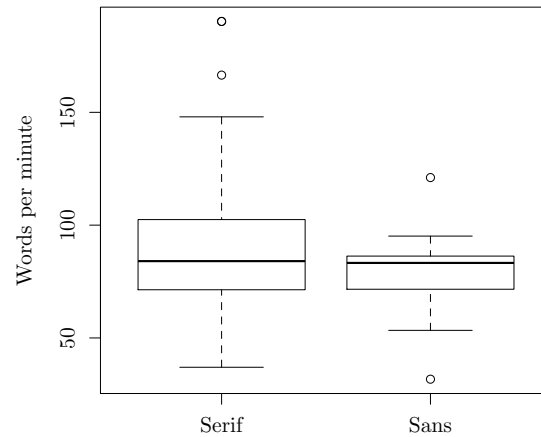


**Figure 3:** Comparison of reading comprehension for medical students and post-stroke patients

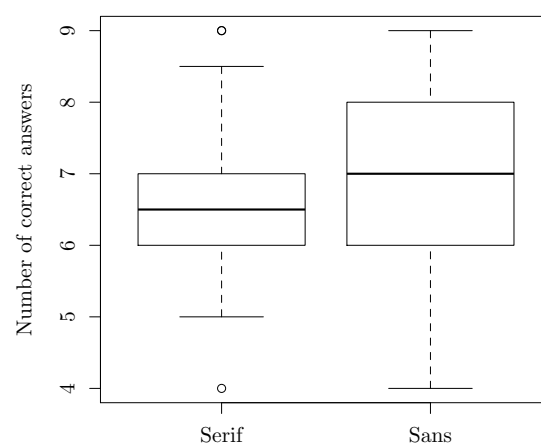
students definitely read faster than patients (on the average 207.13 words per minute vs. 87.84 words per minute with  $p = 6.18 \times 10^{-26}$ ). A rather unexpected fact is that the number of correct answers is *higher* for patients (5.23 vs. 6.78 with  $p = 5.19 \times 10^{-7}$ ). One might argue that the patients were more motivated to learn the information related to their health than students the historical information.

Let us now compare the reading of serif and sans serif fonts by the patients. This is shown in Figures 4 and 5. The figures show that there is not much difference between serif and sans serif fonts. A mathematical expression of this impression is the so-called Student's *t*-test. It gives  $p = 0.13$  for the reading speed and  $p = 0.68$  for the reading comprehension. Such values usually show low statistical significance of the measured difference.

Another way to analyze the data is to perform *paired comparisons*. Instead of looking at the av-



**Figure 4:** Comparison of speed of reading by patients for different fonts



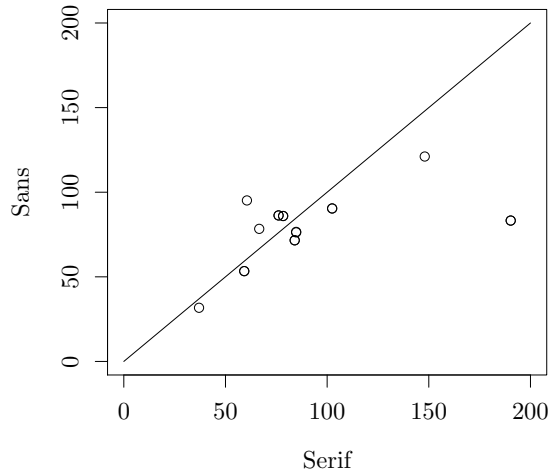
**Figure 5:** Comparison of reading comprehension by patients for different fonts

erages, let us check how much the reading of serif and sans serif texts differs for the same patient. To visualize this, let us put the values for serif fonts on the *x* axis, and the values for sans serif fonts on the *y* axis. Each patient is a data point on this plot. If the data tend to cluster in the upper left part of the plot, then values for sans serif are greater than those for serif. Otherwise the opposite is true.

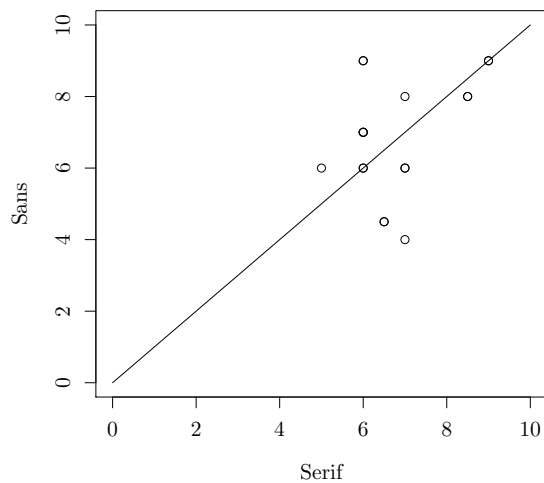
The results are shown in Figures 6 and 7. Again, there seems to be no difference between the fonts on average. So-called paired *t*-tests confirm this observation, giving the values  $p = 0.14$  and  $p = 0.88$ , respectively.

While there was no difference in general, some data points on the figures are far from the diagonal. This means that there were patients for whom the choice of fonts made the difference.<sup>1</sup> The small

<sup>1</sup> We are grateful to Karl Berry for this observation



**Figure 6:** Paired comparison of speed of reading by patients for different fonts



**Figure 7:** Paired comparison of reading comprehension by patients for different fonts

number of patients does not allow us to say at this point whether these points are outliers, or tell an interesting story about individual perception of fonts by some patients. We plan to provide an in-depth study of these patients.

#### 4 Conclusions

The reading by post-stroke patients is an important and interesting topic of research. So far our results, while obtained with a very low number of patients, show that they, like healthy readers, read with the same speed and comprehension both serif and sans serif texts.

#### Acknowledgements

We are grateful to Lilia Nurtdinova (medical student, Bashkir State Medical University, Ufa, Russia), our

volunteers among the patients, the staff of Republic Clinical Hospital, Bashkortostan for the help with this research. The participants at TUG 2014 gave us many useful comments during our talk and afterwards; we want to thank Karl Berry, Alan Wetmore, Pavneet Arora, Jennifer Claudio, and many others. Last but not least we are grateful to the T<sub>E</sub>X Users Group for their encouragement and financial help with attending TUG meetings.

#### References

- [1] Leyla Akhmadeeva, Ilnar Tukhvatullin, and Boris Veytsman. Do serifs help in comprehension of printed text? An experiment with Cyrillic readers. *Vision Research*, 65:21–24, 2012.
- [2] Pavel Farář. *Support Package for Free Fonts by ParaType*, May 2011. <http://mirrors.ctan.org/fonts/paratype>.
- [3] Gordon E. Legge and Charles A. Bigelow. Does print size matter for reading? A review of findings from vision science and typography. *J. Vision*, 11(5)(8):1–22, 2011. <http://www.journalofvision.org/content/11/5/8.long>.
- [4] Debjani Mukherjee, Rebecca L. Levin, and Wendy Heller. The cognitive, emotional, and social sequelae of stroke: Psychological and ethical concerns in post-stroke adaptation. *Top Stroke Rehabil*, 13(4):26–35, 2006.
- [5] Boris Veytsman and Leyla Akhmadeeva. Towards evidence-based typography: Literature review and experiment design. *TUGboat*, 32(3):285–288, 2011. <http://www.tug.org/TUGboat/tb32-3/tb102veytsman-typo.pdf>.
- [6] Boris Veytsman and Leyla Akhmadeeva. Towards evidence-based typography: First results. *TUGboat*, 33(2):156–157, 2012. <http://www.tug.org//TUGboat/tb33-2/tb104veytsman-typo.pdf>.

◇ Leyla Akhmadeeva  
Bashkir State Medical University  
3 Lenina Str., Ufa, 450000, Russia  
la (at) ufaneuro (dot) org  
<http://www.ufaneuro.org>

◇ Boris Veytsman  
Systems Biology School &  
Computational Materials  
Science Center, MS 6A2  
George Mason University  
Fairfax, VA, 22030, USA  
borisv (at) lk (dot) net  
<http://borisv.lk.net>

---

## TeX and copyediting

SK Venkatesan and CV Rajagopal

### Abstract

Copyediting of a manuscript involves bringing consistency at many levels, with many kinds of local and non-local changes. The L<sup>A</sup>T<sub>E</sub>X macros of the proposed `copyediting` package offer an excellent way to create markup that can handle the types of changes made by a copyeditor in a consistent way.

The English language also has certain localization requirements that could be handled through language switches in the spirit of the Babel package. Localization can be achieved through macros such as `\vara{color}` that take care of variant spellings. We also propose a family of macros for other copyediting requirements such as parenthetical commas, serial commas, Latin abbreviations, firstly, secondly, thirdly usages, juxtaposing and the use of appropriate synonyms. This `copyediting` package will streamline the task of copyediting and bring a higher level of quality, visibility and control to the final output.

### 1 Introduction

There can be many a slip between the cup and the lip in the publishing process. The manuscript that arrives in a modern publisher's office, usually as a L<sup>A</sup>T<sub>E</sub>X or an MS Word file, gets transformed bit by bit into a common XML form and then it is typeset into its final PDF form. It is a bit like smelting and purifying iron from its raw form and pouring it into an XML mold, using a DTD as the sieve, to produce the final finished products. Copyediting is a crucial step in the process and is receiving increasing attention now, as copyediting changes are being clearly indicated in the proofing process to the author.

Copyediting involves a broad range of activity: the accurate conversion of the initial input to XML, ensuring consistency of usage within the manuscript, correcting basic language and grammar, applying the finer aspects of the publisher's style, and placing XML hooks to ensure finer typographic aspects are taken care. The XML keeps the link alive between the present print-led world and future worlds such as HTML5. Copyeditors and XML form the bridge between these two worlds. Although TeX can be misused in many ways to make life difficult for a copyeditor [2], we have come a long way from the earlier days when the technology was still underdeveloped [1]. L<sup>A</sup>T<sub>E</sub>X's own secret little macros and TeX4ht have also made it easier to form this bridge between the two worlds.

Just as in all professions, copyeditors come from

a long lineage of tradition. Copyediting tries to filter out what are deemed imperfections and inconsistencies in the manuscript, and ensure that author–reader communication is improved. Each publisher has an in-house style guide that has been refined over many years and forms the basis for copyediting. Our experience with different publishers has shown that it is possible to design a generic set of TeX macros that can be used in the spirit of BIBL<sup>A</sup>T<sub>E</sub>X macros.

It should be mentioned here that these macros are not designed to replace copyeditors but to make it easier for them to take care of mundane aspects of copyediting in a systematic way, so that they will be able to concentrate on improving what's crucial, the author–reader communication. Despite market trends, the role of copyediting has never been more important in the present world with varied rendering devices, with different aspect ratios and modern semantic capabilities.

### 2 Copyediting macros

Copyediting involves quite a broad spectrum of activity. At one end of the spectrum it improves semantic communication between the author and the reader. At the other end of the spectrum it reinforces certain stylistic and typographic conventions of the publisher. Semantic aspects are much beyond the capability of ordinary TeX macros, so it is at the latter end of the spectrum that most of this effort will be focussed.

We break down the copyediting process into various modular components:

1. Localization (`loc`)
2. Close-up, Hyphenation, and Spaced words (`chs`)
3. Latin abbreviations (`lat`)
4. Acronyms and Abbreviations (`abr`)
5. Itemization, nonlocal lists and labels (`itm`)
6. Parenthetical and serial commas (`pc`)
7. Non-local tokenization (`nlt`)
8. Genus-species identification (`gsp`)
9. Juxtaposing (`jxt`)
10. Synonyms (`syn`)

A macro `\delins` is used to indicate the copyediting text changes as in:

```
\delins[opt]{deleted text}{inserted text}
[comments about the change]
```

The option (`opt`) within the square brackets indicate the category to which this change belongs:

```
opt=loc,chs,lat,abr,itm,pc,nlt,gsp,jxt,syn
```

However, these macros do not expose the detailed information about the copyediting categories, so we have created specific macros for each of the copyediting categories.

### 3 Localization —

#### British-American-Australian-Canadian

There are many sub-categories in British-American-Australian-Canadian variations:

- DG (Am) vs. DGE (Au, Br, Ca). In American spelling, **Acknowledgement**, **Judgement** (et al.) lose the e to become **Acknowledgment**, **Judgment**.

- Z (Am, Ca) vs. S (Br, Au). American and Canadian spelling prefers ize, while Australian and British use ise in words like **apologize/apologise** and **authorize/authorise**. However, the rule is different for yze/yse patterns, in words like **analyze/analyse**, where American prefers z and the rest use s.

- S (Am, Ca) vs. C (Br, Au). In words like **defence/defence**, **offense/offence**, American and Canadian prefer s instead of c.

- G (Am) vs. GUE (Br, Au, Ca). In words like **dialog/dialogue**, **catalog/catalogue** American prefers to drop the ue.

- OR (Am) vs. OUR (Br, Au, Ca). In words like **color/colour**, **favor/favour** American omits the u.

- ER (Am) vs. RE (Br, Au, Ca). In words like **center/centre**, **caliber/calibre** American prefers the er spelling.

- L (Am) vs. LL (Br, Au, Ca). In words like **canceled/cancelled**, **modeled/modelled** American prefers the single l spelling while the rest prefer double l.

- Others. Many other differences don't fall into any regular pattern like the above, and so can be handled only by a word list with their corresponding language mapping table.

We use one macro to care of all of this complexity: `\vara{color}`. The switch to a particular language spelling is done in the preamble, e.g.:

```
\usepackage[lang=uk]{copyediting}
```

Both `\vara{color}` and `\vara{colour}` produce the same output, in this case **colour**, so the author's original text need not be changed. The other options for language switch in this context are **am**, **ca**, **au**. The default language is the British spelling. In exceptional cases when one wants to force a particular use in a particular instance one can use `\vara*{analog}`, which will leave the input unchanged.

#### 4 Close-up, hyphenation, and spaced words

Although American spellings use fewer hyphens, that modern preference for closed prefixes has exceptions:

1. if the root word is a proper noun or a number (post-Depression, pre-2001)
2. for double prefix (non-self-governing)
3. if the prefix precedes a proper open compound then en-dash is used (pre-Civil War)

4. if two instances of the letter i or the letter a are adjacent (anti-intellectual, extra-action), or other combinations of letters that could hamper reading (**pro-labor**)

5. for a double prefix (**anti-antibody**)

6. for a repeated prefix with implicit use (**over- and understimulation**)

We use the macro `\hyp{{anti}{body}}` to hyphenate a compound word. For a closed-up word we use `\closeup{{anti}{body}}`. For compound words that occur as two separate words we use `\sword{{Civil}{War}}`. You might wonder what use are such complex verbose macros in a  $\LaTeX$  file? They give visibility to the corrections the copyeditor makes and offer hooks to produce a global inventory of various changes while at the same time making it feasible to make switches on a global scale.

#### 5 Latin abbreviations

Latin abbreviations such as:

cf.	compare	et al.	and others
etc.	and so forth	e.g.	for example
i.e.	that is	NB	note

are straightforward to handle with a macro, as in `\lat{et al.}`, where the stylistic aspect is taken care of by global switches:

```
\usepackage[lat=0,abbr=italic]{copyediting}
```

The default `lat=0` leaves the text as is, and `italic` defines the font used. Using option `lat=1` removes all the dots, and `lat=2` changes the text to its English equivalent shown above.

#### 6 Acronyms and abbreviations

If the initial letter abbreviations are spoken together as a word, as in **Acquired Immune Deficiency Syndrome (AIDS)**, the term “acronym” is used — but we will not make this distinction here and treat them as one and the same. A simple macro, `\ac{AIDS}`, is good enough and the default global switch ensures that it is expanded correctly the first time within parenthesis. The mapping between acronym and expansion is declared using:

```
\newacro{AIDS}
{Acquired Immune Deficiency Syndrome}
```

Many standard acronyms are available by default from the package, and only new acronyms need to be added this way. This can be checked during compilation. To avoid expansion of trite acronyms the first time, one can use the starred form, `\ac*{UK}`.

#### 7 Itemizations, nonlocal lists, labels

A list with only a few items may be written like this:

- Firstly, this is an endangered species;

- Secondly, humans find them delicious;
- Thirdly, they are only found on this island.

In this example we could have as well have used ‘first’, ‘second’, ‘third’, instead of ‘...ly’, making that a global option. It is also possible that the items may be changed to use standard arabic numeral: 1, 2, 3, ... In order to make such changes possible with a simple switch, one can use macros:

```
\begin{eitem}
  \item this is a endangered species;
  \item humans find them delicious;
  \item they are only found on this island.
\end{eitem}
```

Given that we run L<sup>A</sup>T<sub>E</sub>X at least three times, we can have an option to change the last item in the list to lastly, as a global switch:

```
\usepackage[eitem=0,last]{copyediting}
```

where `eitem=0` is the (default) option that causes firstly, secondly ... and last indicates that the last item should be lastly. With `eitem=1` set, ly drops out, and with `eitem=2,3,...`, a standard list (enumerated, bulleted, etc.) list is output.

## 8 Parenthetical and serial commas

Many long sentences are difficult to read and can be communicated better with parenthetical constructs or footnotes rather than commas. It would be nice to have switches that can make this change, as in:

The enthusiastic young ducks flying in front of the group\pc{led by the sagacious older ones at the back, make a lot of noise and turbulence} which are used by older ones at the back to warm their heart and the wings.

which outputs to some variation of:

The enthusiastic young ducks flying in front of the group, led by the sagacious older ones at the back, make a lot of noise and turbulence, which are used by older ones at the back to warm their heart and the wings.

Depending on the global switch `pc=0,1,2,3,4` we can choose a parenthetical comma, parenthesis, em-dash, footnote or sidenote.

## 9 Elist (Oxford comma)

For a list of items as in this sentence:

Suddenly warblers, tits and wrens started singing in chorus ...

we change the source to:

```
Suddenly \elist{warblers,tits,wrens}
started singing in chorus...
```

which can then be transformed into:

Suddenly warblers, tits, and wrens, started singing in chorus ...

This macro helps bring consistency across the document regarding the placement of comma before and in the last item and in ensuring proper white-space after the comma for each item. The comma before and is known as the Oxford comma and can be triggered by a global switch, `oxfordcomma`.

## 10 Non-local tokenization

In a sequence of minimization operation, in a typical newspaper article the copyeditor encounters:

His Holiness, the Prince of Mangoistan, addressed a gathering of ordinary mangoes in the capital New Mango. The Prince of Mangoistan pointed out the serious threat of foreign insects in the country. He further pointed out ...

His Holiness the Prince of Mangoistan shrinks to The Prince of Mangoistan and then finally to He. This copyediting operation can be denoted using:

```
\defin token{mango}
  {His Holiness, the Prince of Mangoistan}
  {The Prince of Mangoistan}
  {He}
```

at the first instance and then `\Token{mango}` at the later instances. The `\Token{mango}` macro can thus be useful simply to indicate to which nouns the important pronouns link in a paragraph. (However, not all pronouns in English language have corresponding original objects as in the case of ‘It’ in ‘It is raining!’.)

## 11 Genus-species identification

The Genus species formatting is similar to Latin abbreviations in many ways but has its own conventions as well. The macro:

```
\gensp{E. coli}
```

italicizes all instances and expands the abbreviations at the first instance. Like the Latin abbreviation macro `\lat`, this macro also allows the embedding of new undefined genus species entities, as in:

```
\definegensp{E. coli}{Escherichia coli}
```

## 12 Juxtaposing

Consider the sentence:

It is another politician that we can't trust in the White House.

This can be changed to:

It is another politician \pull that we can't trust\push{ in the White House}.

The output of this being:

It is another politician in the White House  
that we can't trust.

If this is the original and we want it the other way  
around, we can similarly write:

It is another politician `\push{in the White  
House}` that we can't trust `\pop`.

Juxtaposing has wide application, especially for copy-  
editing misplaced modifiers. We use three macros  
`\pull`, `\push` and `\pop` to achieve juxtaposing.

### 13 Synonyms

Sometimes we also need to use the appropriate syn-  
onyms (e.g., due to redundancy in the original), as  
in:

The temperature of the water was raised to  
80°C to see if some bacteria ...

This could be changed to:

The temperature of the water was  
`\syn{raised}{increased}` to 80°C to see if  
some bacteria ...

producing the output:

The temperature of the water was increased  
to 80°C to see if some bacteria ...

The `\syn` macro can thus be used for fixing inappro-  
priate usage with an appropriate equivalent.

### 14 Interactive proofing

The above set of macros bring a certain level of trans-  
parency and consistency to the copyediting process.  
Using additional macros, this also has the potential  
to convey further the key aspects of copyediting to  
the author using menus and dashboards, bringing an  
interactive aspect to the proofing process.

### 15 Conclusion

We have made an attempt at bringing together many  
copyediting aspects as  $\LaTeX$  macros. This involves  
some amount of drastic simplification and abstraction  
that cannot suffice in all cases. The starred macros  
can be used in those exceptional cases where one  
needs to escape the global switch. The non-local  
linkages work just as in the case of bibliography  
links by multiple compilation of  $\LaTeX$  that passes  
information through auxiliary files.

All this is only a small step towards the Hi-  
malayan task of climbing the semantic hill through  
 $\LaTeX$  macros as envisaged by Sense $\TeX$  [5].

### 16 Related work

There are some CTAN packages which we would like  
mention in relationship to our copyediting macros.  
The `abbrevs` package [4] has an interesting set of  
macros but the spirit of the macros here is quite  
different, in that we would like to keep the original  
text of the author in some form or another, so that  
changes made can be shown in a transparent manner.  
The `acronym` package [3] is very close in concept  
to our acronym macros and has some additional  
interesting features that needs to be considered in  
our copyediting macros.

### Acknowledgements

We would like to thank Lorna O'Brien for important  
inputs on English language and its varied usages  
across countries and publishers. Of course, this work  
would not have been possible without the constant  
encouragement of Mariam Ram, TNQ and C.V. Rad-  
hakrishnan, River Valley Technologies.

### References

- [1] P. Flynn (1993).  $\TeX$  and SGML: A Recipe  
for Disaster? *TUGboat* **14**(3), 227–230. <http://tug.org/TUGboat/tb14-3/tb40flynn.pdf>
- [2] E. Gregorio (2005). Horrors in  $\LaTeX$ : How to  
misuse  $\LaTeX$  and make a copy editor unhappy,  
*TUGboat* **26**(3), 273–279. <http://tug.org/TUGboat/tb26-3/tb84gregorio.pdf>
- [3] Tobias Oetiker (2012). An Acronym  
Environment for  $\LaTeX$  2 $\epsilon$ .  
<http://ctan.org/pkg/acronym>
- [4] Matt Swift (2001). The `abbrevs`  $\LaTeX$  package.  
<http://ctan.org/pkg/abbrevs>
- [5] S.K. Venkatesan (2005). Moving from bytes to  
words to semantics. *TUGboat* **26**(2), 165–168.  
<http://tug.org/TUGboat/tb26-2/venkat.pdf>

- ◇ SK Venkatesan  
TNQ Books and Journals Pvt Ltd,  
Dr Vikram Sarabhai Instronic  
Estate, Kottivakkam,  
Chennai 600041, India  
[skvenkat \(at\) tnq dot co dot in](mailto:skvenkat@tnqdotco.in)
- ◇ CV Rajagopal  
River Valley Technologies,  
JWRA 34, Jagathy,  
Trivandrum 695014, India  
[cvr3 \(at\) river-valley dot org](mailto:cvr3@river-valley.org)

## An output routine for an illustrated book: Making the *FAO Statistical Yearbook*

Boris Veytsman

### Abstract

Output routines involving illustrations (“floating bodies” in the L<sup>A</sup>T<sub>E</sub>X lingo) are the most complex part of T<sub>E</sub>X. For most algorithms used in T<sub>E</sub>X, L<sup>A</sup>T<sub>E</sub>X and ConT<sub>E</sub>Xt the basic concept is a flow of text, occasionally interrupted by illustrations which can be placed anywhere close to the point they are mentioned. The story is told mainly by the text, and illustrations have a secondary role.

Here we discuss the different case of an *illustrated book*, where the main story is told by the illustrations and their interaction. The simplest examples of such books are art albums. Another (surprising) example is the *FAO Statistical Yearbook*, where the story is told primarily by maps, charts and tables, while the text has a secondary role.

We describe a concept of a relatively simple output routine for such books and its implementation in L<sup>A</sup>T<sub>E</sub>X.

### 1 Introduction

A recent report by the L<sup>A</sup>T<sub>E</sub>X3 team [4] contained the exhortation to engage in “collecting and classifying design tasks”. In this paper we describe a design task and propose a way to solve it. While the code here is L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>-specific, we hope the algorithm and concepts may be useful for other formats as well.

Probably one of the most difficult concepts in T<sub>E</sub>X is illustrations (“floats” in L<sup>A</sup>T<sub>E</sub>X nomenclature). They interrupt the galley, and T<sub>E</sub>X should put them on the page outside of the normal flow, using an asynchronous output routine (OTR). Various OTRs for plain T<sub>E</sub>X are described in the series of papers by Salomon [5, 6, 7]; the last part deals specifically with insertions, the usual way to typeset illustrations in plain. L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> algorithms are described in [1], probably the most complex part of L<sup>A</sup>T<sub>E</sub>X code. These algorithms deal with one- or two-column typesetting with illustrations of arbitrary height occupying one or two columns. ConT<sub>E</sub>Xt can deal with a more general situation of  $n$ -columns with illustrations occupying  $m$  columns of text [3].

It should be noted that all these cases assume that the main story is told by the text. Illustrations are put on the pages almost as an afterthought. They do not interact with each other. The only task of the algorithm is to put them somewhere not too far from the point they are mentioned, and without creating too much empty space on the pages.

Boris Veytsman



Figure 1: An example of an art album spread (from [2])

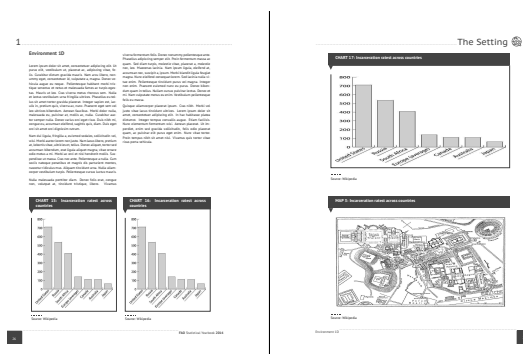


Figure 2: Mock-up spread of *FAO Statistical Yearbook* 2014

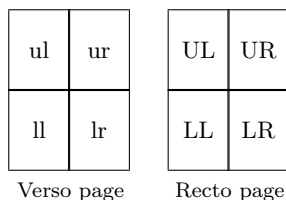
One can imagine the opposite situation: the story is told primarily by the illustrations and their interaction, while the text plays an auxiliary rôle. In this case the author spends much effort in putting the illustrations exactly where she wants them to be, and the task of the compositor is to put the text in the remaining empty space in a pleasing manner. We will call books created in this manner *illustrated books*, for lack of a better term.

An immediate example of an illustrated book is an art album (Figure 1). The importance of interaction between the pictures is well known to artists; this is why good exhibitions usually have “Hanging Committees” that carefully discuss the order and positions of the pieces. It is evident from Figure 1 that the author first put the illustrations on the pages, and then filled the rest with the text.

A more surprising example is the *FAO Statistical Yearbook* (Figure 2). The Yearbook uses tables, charts and maps to illustrate the statistical trends. Their positions on the pages is determined by the graphic designer; the text must fill the gaps.

In the rest of the paper we discuss how the design shown in Figure 2 was implemented in L<sup>A</sup>T<sub>E</sub>X. The





**Figure 3:** FAO Yearbook spread

code is available in the repository at <http://github.com/filippogheri/FAOSYBLaTeXpackage>, and a formatted version is available online with this article.

## 2 User interface

The main unit of the *FAO Yearbook* is the *spread*. As shown in Figure 3, it is split into eight *quadrants*, four per page. The quadrants are denoted by two-letter combinations like `ul` for upper left and `lr` for lower right. Lowercase is used for verso (even) pages, and uppercase for recto (odd) pages.

The illustrations have fixed sizes: they can occupy one, two, or four quadrants. Accordingly there are four kinds of illustrations: ‘Single’ ones take one quadrant, ‘Tall’ ones take two quadrants stacked vertically, ‘Wide’ ones take two quadrants stacked horizontally, and ‘Big’ ones take all four quadrants on a page.

The user specifies the illustration type (e.g. `chart` or `map`), its size (`S`, `T`, `W` or `B`), and the upper left quadrant occupied by the illustration. We used  $\LaTeX$  environments for this. The name of the environment corresponds to the illustration type, while its mandatory arguments specify its size and position. For example, the code

```
\begin{chart}{S}{LR} ... \end{chart}
```

specifies a chart occupying a single lower right quadrant on a recto page, while the code

```
\begin{map}{W}{ul} ... \end{map}
```

specifies a map occupying two top quadrants on a verso page.

The user writes down the code for the illustrations and the text, and  $\TeX$  typesets them according to the chosen pattern. The command `\clearpage` typesets all illustrations and text obtained so far.

## 3 Algorithms

In this section we describe the algorithms used to typeset the book.

The main problem is *when* to start output. If we had just illustrations, then the answer would be simple: as soon as we have enough illustrations for the full page. This is the approach used by Dave Walden in his photo album macros [8]. However, since we have illustrations *and* text, we are in a more

complex situation. We need to check whether we have enough text to fill the gaps. This is done by page builder. There are ways to inform the page builder about the space needed by illustrations [5]. However they assume that all illustrations should be put on the page being built. In our case we may have both illustrations for the current page *and* illustrations for following pages. Only the OTR knows which illustrations belong to the current page, but the OTR is started asynchronously by the page builder. Thus our algorithms must include communication between the page builder and the OTR.

Each of the environments described in Section 2 adds its contents to the bottom of an *illustration box*. There are 18 such boxes corresponding to all valid combinations of illustration size and position (an attempt to insert, e.g. a Tall illustration starting at the lower left quadrant produces an error since this combination is not valid). We use `\vsplit` to extract the top (oldest) illustration from the box.

We follow the basic idea of [1] for two-column typesetting. The page builder starts the OTR whenever a column of text is formed. It is the job of the OTR to determine whether we are at the first or second column, and proceed accordingly. One can imagine the OTR having two stages: the first deals with a first column from the page builder, and the second has two columns to work with.

So at the first stage we have a column of text. We also know whether this column is the first or the second, and whether we are on a recto or a verso page. Thus we can check whether we already have illustrations in the quadrants for this page.

First, it can happen that the current page is completely covered by Wide or Big illustrations. In this case we do not need to put any text on the page, and simply output the illustrations. Note that this should happen only when we typeset the first column — otherwise we have a full column of text which belongs to a wrong page: recto or verso.

If after this test we are still inside the OTR, then we are free to form a column. Again, it may happen that this column is completely taken by illustrations; in this case we return the text to the page builder and send illustrations to the second stage.

Now we are at the most interesting part of the algorithm. We have text and possibly illustrations to mix in the column. However, is the height of the text box right? Possibly not: the page builder might think that there were no illustrations and not correct for them. Fortunately,  $\TeX$  provides a global parameter `\vsize`, which reflects the page builder’s idea about the required text height. So we can calculate the required height in the OTR and

**Algorithm 1:** OTR, first stage

---

```

if have Big or both top & bottom Wide
illustrations then
  if second column then
    ⊥ Error
    Send the illustrations to the special OTR;
    Send text back to page builder
if have Tall or both top & bottom Single
illustrations then
  Form a column from the illustrations;
  Send the column to the second stage;
  Send the text back to page builder
Calculate column height;
if column height equals \vsize then
  Add illustrations to the column;
  Send the column to the second stage
else
  Change \vsize;
  Send text back to page builder;
  Leave OTR

```

---

**Algorithm 2:** OTR, second stage

```

if first column then
  Save column
else
  Add first column and wide illustrations,
  add decorations and ship the page out
Reset \vsize; Leave OTR

```

---

compare it with `\vsize`. If they coincide, we are good. If not, we change `\vsize` and return the text to the page builder. It is easy to see that this code produces at most two passes of OTR.

This finishes the first stage of the OTR (Algorithm 1). The second stage of OTR is relatively simple (Algorithm 2): we either save the column for the next pass or form the page for shipout.

The special OTR deals with pages completely covered by Wide or Big illustrations (Algorithm 3): we put them on the page and add decorations.

Our implementation of `\clearpage` is simpler than the one in  $\text{\LaTeX} 2_{\epsilon}$ . The latter needs to tell the OTR that this is a special case, and illustrations, if any, must be put on the page. In our case we are guaranteed that if there are illustrations for the given page number “parity” (i. e. for even or odd pages), they will in fact be put on the page. Thus we just repeatedly call OTR (Algorithm 4).

As usual, we need to add `\clearpage` to the `\AtEndDocument` hook to avoid loss of illustrations.

## 4 Conclusions

We see that  $\text{\TeX}$  can be coaxed to provide a relatively unusual layout. This document model might be of interest for the designers of new  $\text{\TeX}$ -based formats.

Boris Veytsman

**Algorithm 3:** OTR, special case

---

```

Put illustrations on the page;
Add decorations and ship the page out;
Reset \vsize;
Leave OTR

```

---

**Algorithm 4:** `\clearpage`

```

while some illustration boxes are not empty do
  ⊥ Call OTR

```

---

## Acknowledgments

This work would have been impossible without great and patient people at FAO UN: Filippo Gheri, Amy Heyman, Shira Fano, and many others.

I am grateful to Hans Hagen and Frank Mittelbach for the discussion of  $\text{\ConTeXt}$  and  $\text{\LaTeX}$  float routines and to Dave Walden for letting me know about his paper.

As always, the participants of the TUG meeting gave me many interesting comments and suggestions.

## References

- [1] Johannes Braams, David Carlisle, Alan Jeffrey, Leslie Lamport, Frank Mittelbach, Chris Rowley, and Rainer Schöpf. *lfloat.dtx*, 2014.
- [2] Rick Cusick. *What Our Lettering Needs: The Contribution of Hermann Zapf to Calligraphy & Type Design at Hallmark Cards*. RIT Cary Graphics Art Press, 2011.
- [3] Hans Hagen. *Columns*, 2003. <http://www.pragma-ade.nl/general/manuals/columns.pdf>.
- [4]  $\text{\LaTeX}$  Project Team.  $\text{\LaTeX} 3$  news, issue 9. *TUGboat*, 34(1):22–26, 2014.
- [5] David Salomon. Output routines: Examples and techniques. Part I: Introduction and examples. *TUGboat*, 11(1):69–85, 1990.
- [6] David Salomon. Output routines: Examples and techniques. Part II: OTR techniques. *TUGboat*, 11(2):212–236, 1990.
- [7] David Salomon. Output routines: Examples and techniques. Part III: Insertions. *TUGboat*, 11(4):588–605, 1990.
- [8] David Walden. Every  $\text{\LaTeX}$  document brings new (to me) programming issues. <http://walden-family.com/texland/tex-programming.pdf>, 2014.

◇ Boris Veytsman  
 George Mason University  
 borisv (at) lk (dot) net  
<http://borisv.lk.net>

## xml2tex: An easy way to define XML-to- $\LaTeX$ converters

Keiichiro Shikano

### Abstract

xml2tex is a framework to give XML a presentation layer using  $\LaTeX$ . In other words, xml2tex lets you use an XML-based format as a source of  $\LaTeX$ . It may sound awful at first, but an XML-based format has some advantages, especially for creating books. This paper describes why XML does matter, and introduces xml2tex's intuitive way of relating XML to  $\LaTeX$ , based on a Scheme dialect and SXML.

### 1 $\LaTeX$ as presentation for XML

Creating documents can be seen from two opposite aspects: structure versus presentation. In some document systems, they are divided into completely separate layers. For example, XSL [10] is the way to define a presentation of XML, which corresponds to the tree structure of a document. Håkon Wium Lie, the father of CSS [11], explains this separation in terms of a ladder of abstraction [6]. The structural tree of the document is at the highest abstraction level. Moving downwards on the ladder, the document becomes less and less abstract towards the rendered data. It's hard to climb the ladder without any manual aid. That means that reusing the document in other media requires much manual work.

Oddly enough, this separation is rather loose in  $\LaTeX$ , despite the fact that  $\LaTeX$  originally is the structured layer over the lower-level typesetting mechanism provided by the  $\TeX$  engine. This weak separation can sometimes make reusability of  $\LaTeX$  documents problematic. E.g., if you want to distribute a  $\LaTeX$  document through the Web, chances are that it will be as a PDF.

Figure 1 shows the abstraction ladder for purposes of creating books, our main concern. Arrows in Figure 1 indicate that the translation or mapping between the formats is achieved by following some restrictions. In other words, the expressiveness of your document would be limited in accordance with the formats in higher abstraction levels. The left-down arrow from XML to  $\LaTeX$ , for example, refers to a system that can transform XML files written in some given DTDs or XML Schemas, such as DB2 $\LaTeX$  [2] (a converter from DocBook to  $\LaTeX$ ) or  $\TeX$ ML [7] (a feasible XML syntax for  $\TeX$ ). In those systems, you can hardly create a book requiring more structural elements than the specifications offer. Similarly, the easy-to-read-and-write input formats like markdown and Wiki syntaxes narrow

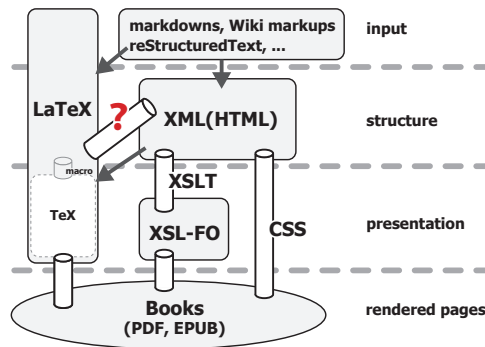


Figure 1: Ladder of abstraction in creating books

the possible expression of documents down to their intended use. This can be very good for writing and editing the texts, but not for supporting a variety of page layouts.

On the other hand, the plumbing pipes connecting different formats indicate that there's practically no restriction to a downward direction. Needless to say,  $\LaTeX$  is able to produce almost any possible page designs; as we'll see, this is one of the most important reasons we'd like to use  $\LaTeX$  in creating books. The same is true for XSL (including XSLT and XSL-FO), regarded as the best path to render a variety of page layouts from a tree structure of XML. XML also has CSS as the mechanism to apply an arbitrary style to the tree.

What is missing here is a feasible mechanism for producing  $\LaTeX$  from non-restrictive XML. The most common approach for now is to use XSLT. However, XSLT is meant to convert an XML into another XML, so it lacks support for writing XML-to- $\LaTeX$  converters. Another approach to providing a mechanism suitable for  $\LaTeX$  is XML $\TeX$  [1]: an XML parser written in  $\TeX$ . This is a great accomplishment in terms of  $\TeX$  macro programming, but we did not find it easy to write our required converter using XML $\TeX$ . A more practical approach is to use Con $\TeX$ t's XML support [3]; this supports a declarative interface to select an XML element and define the corresponding Con $\TeX$ t syntax. When we are able to use Con $\TeX$ t in typesetting Japanese books, it will be a good alternative to our own attempt, called xml2tex [9], described in the following.

### 2 Defining maps from XML element tags to $\LaTeX$ syntax, the xml2tex way

Let's start with a silly HTML example.

```
<html>
Lorem % ipsum \ ... $10,000
</html>
```

Leaving aside the escaping of special characters ('%', '\ and '\$'), we have to decide how to express this HTML in L<sup>A</sup>T<sub>E</sub>X. One feasible representation is achieved by mapping its only element (<html>) to `\begin{document} ... \end{document}`. Here is the xml2tex way to do this:

```
(define-tag html (make-latex-env 'document))
```

That's it!<sup>1</sup> Put this line down and save it as the file `silly.rule`, then run `xml2tex` like this:

```
$ xml2tex --rule="silly.rule" sample.html
\documentclass{book}
\usepackage[T1]{fontenc}
\begin{document}
Lorem {\symbol{37}} ipsum {\symbol{92}} ...
{\symbol{36}}10,000
\end{document}
```

Special characters are automatically escaped using the `\symbol` command under the T1 encoding. The argument to `\documentclass` defaults to `book`; of course this can be easily modified. Before that, however, let's give a slightly more practical example:

```
<html>
  <head>
    <title>a quite nice document</title>
  </head>
  <body>
    <p>Lorem % ipsum \ ... <em>$10,000</em></p>
    <p>dolor % sit \ amet ... <em>$42</em></p>
  </body>
</html>
```

In this HTML data, the main part of the document is wrapped with a `<body>` tag. That is, this time the `<body>` is the appropriate source for the L<sup>A</sup>T<sub>E</sub>X's document environment, instead of the `<html>` as in the previous example. So we change the previous rule like this:

```
(define-tag body (make-latex-env 'document))
```

We also need to handle the other tags in the `<body>`, namely `<p>` and `<em>`. Each `<p>` should be a paragraph in L<sup>A</sup>T<sub>E</sub>X. On the other hand, the L<sup>A</sup>T<sub>E</sub>X counterpart of `<em>` is `\emph{}`. These two types of mappings seem to be quite different. Nevertheless, when viewed as a recursive tree conversion, both mappings, and what is more almost all such mappings, can be regarded as a common routine:

1. Start a L<sup>A</sup>T<sub>E</sub>X piece with `\begin{foo}`, `\foo{}`, or other strings.
2. Recursively process the node's children. If the only child is a simple string, then output the string with any necessary conversions.

<sup>1</sup> The single quotation mark in `'document` is not a typo. It tells `xml2tex` that this is not a variable name, but a data item; specifically, a symbol in the Scheme programming language.

3. End the L<sup>A</sup>T<sub>E</sub>X with the required `\end{foo}`, `}`, etc.

In fact, the second argument to `define-tag` is a rule which encodes this routine, and `make-latex-env` is the function that yields a common rule for generating a L<sup>A</sup>T<sub>E</sub>X environment. The rule is: "Put `\begin{...}` at the head; convert the children recursively with necessary escaping; put `\end{...}` at the tail."

To explicitly define such a rule, we can use the `define-rule` declarative. `define-rule` takes three arguments, each corresponding to the above actions, in order: what to do at the beginning, what to do with the text nodes of the content, and what to do at the end.

For example, here is a possible rule for `<p>`:

```
(define-tag p ; if the node is this name ...
  (define-rule
    "\n" ; put this at the beginning ...
    trim ; its text nodes should be ...
    "\\par\n")) ; put this at the ending.
```

where `trim` is a utility for trimming a string for use in L<sup>A</sup>T<sub>E</sub>X. Although we put the rule line by line in the above example, line breaks and other white space are generally immaterial. Text after a semicolon (;) is a comment.

A rule for `<em>` could be defined like

```
(define-tag em (define-rule "\\emph{" trim "}"))
or, equivalently,
(define-tag em (make-latex-cmd 'emph))
```

where `make-latex-cmd` is a utility to define a rule outputting the given L<sup>A</sup>T<sub>E</sub>X command.

The last rules we have to define are for `<head>` and `<title>`. Although we could use this meta-information to generate L<sup>A</sup>T<sub>E</sub>X content, here we will just ignore them. To make the converter discard an XML element, we can use a predefined rule `ignore`.

```
(define-tag head (ignore))
```

Consequently, we don't need to define a rule for the `<title>` tag, because the converter already knows that its parent tag is going to be discarded.

In short, to get a decent result from the above HTML data all we need are these four lines:

```
(define-tag head (ignore))
(define-tag body (make-latex-env 'document))
(define-tag p (define-rule "\n" trim "\\par\n"))
(define-tag em (make-latex-cmd 'emph))
```

If we run `xml2tex` with those rules, we get valid L<sup>A</sup>T<sub>E</sub>X source for a book, because the default `\documentclass` is `book`. This is defined in a file `default.rule` in a way similar to the other rules below, and can be overridden with your own definition.

### 3 Details and tricks

As we have seen through the examples, `xml2tex` works as a domain-specific language (DSL) for defining maps between each XML tag and corresponding  $\LaTeX$  syntax. When it comes to DSL, a programming language in the Lisp family fits well. `xml2tex` is written in `Gauche` [4], a dialect of the Scheme programming language. In addition to being a member of the Lisp family, `Gauche` has many text processing features and libraries, useful in defining more complex conversion rules.

To take advantage of a profound feature of the general programming language: the first and third arguments of `define-rule` need not be literal strings but can be Lisp functions without arguments. For example, a rule for the `<title>` tag could be defined like this:

```
(define-tag title (define-rule
  (lambda ()
    (cond
      ((eq? ($parent) 'chapter) "\\chapter{")
      ((eq? ($parent) 'sect1)  "\\section{")
      ((eq? ($parent) 'sect2)  "\\subsection{")
      (else (error "title" $parent))))))
  trim
  "}")
```

In this definition, the first argument to `define-rule` is a function to select the appropriate  $\LaTeX$  representation for the `<title>` tag based on its parent node. If the `<title>` tag belongs to `<chapter>`, the function returns the Scheme string `"\\chapter{"`. If `<sect2>`, then `"\\subsection{"`.

The other new feature used here is the keyword `$parent`. It expands to the name of the direct parent of that node. This is one of many “shortcuts” provided by `xml2tex` that can be used to retrieve the information from the XML tree. Table 1 lists these predefined convenience keywords.

Below is a naive example of using `$@`, which works as a function to retrieve the value of the specified attribute. We will also use the `Gauche` syntax `#"..."` for string interpolation. For example, we want `#"[width=,($@ 'width)]"` to expand to `[width=100mm]` if the `<img>` tag has the attribute `width="100mm"`.

```
(define-tag img (define-rule
  (list "\\begin{figure}\n"
        "\\includegraphics"
        #"[width=,($@ 'width)]"
        #"{,($@ 'src)}"))
  trim
  "\\end{figure}"))
```

Using these `$` keywords, we are able to define most rules declaratively and intuitively. In this re-

**Table 1:** List of keywords defined by `xml2tex`

keyword	description
<code>\$body</code>	Body of the element.
<code>\$root</code>	Whole document tree.
<code>\$parent</code>	Direct parent of the element.
<code>\$parent? [name]</code>	If the element has parent with <code>[name]</code> .
<code>\$childs</code>	List of all children of the element.
<code>\$child [name]</code>	List of children with <code>[name]</code> .
<code>\$following-siblings</code>	List of following-siblings.
<code>\$siblings</code>	List of both following- and preceding-siblings.
<code>\$@ [name]</code>	String value of the attribute <code>[name]</code> .
<code>\$under? [list]</code>	If the element is a descendant of one of <code>[list]</code> .
<code>\$ancestor-of? [list]</code>	If the element has any descendant in <code>[list]</code> ?

gard, we can think of `xml2tex` more like a DOM (tree model) rather than SAX (event model). Indeed, `xml2tex` parses the entire XML tree in advance. This parsed tree has a form of SXML [5], a representation of the XML Infoset in the form of S-expressions. Even this bare SXML tree is available when defining a rule. It is sometimes necessary for elements which require transforming the original structure to define a reasonable mapping to  $\LaTeX$  syntax. Tables are one such formidable challenge, as we’ll see next.

### 4 Tables

To convert XML’s logical structure of tables into  $\LaTeX$  is a substantial problem. We think the root cause is probably the lack of a general model for tables in  $\LaTeX$ .<sup>2</sup>

Let us consider the conversion rule for typical HTML tables. If we use `tabular` environment as the basic  $\LaTeX$  construct for HTML tables, then the first attempt might be:

```
;; this doesn't work!
(define-tag table (define-rule
  "\\begin{tabular}\n"
  trim
  "\\end{tabular}"))

; put "\\" after each lines of table.
(define-tag tr (define-rule "" trim " \\\\\\"))

; put "&" after each cells in line.
(define-tag td (define-rule "" trim " &"))
```

<sup>2</sup> In contrast, `ConTeXt` has a standard model for tables and thus it’s easier to define mappings from XML tables [8].

Unfortunately, this does not work. First, the `tabular` environment requires an argument explicitly specifying the appearance of each column. To determine this information from the given HTML table, we have to look through the entire table contents in advance. Second, we don't want the following '&' for the last cell of each line. Third, we should be able to change the width and color of each cell, as well as to span columns or rows. This information could be found in the attributes of `<td>`.

What we need is a way to transform the raw SXML tree before applying the corresponding conversion rules. To do that, we pass a procedure to `define-rule` using the `:pre` keyword. Below is a (relatively) simple example to decide the column specs for the `tabular` without any additional information except the table itself.

```
(use xmltex.latextable)
(define-tag table (define-rule
  #"\begin{tabular}{|,($@ 'colspec)}\n"
  trim
  "\\end{tabular}\n"
  :pre calc-colspec))

(define (calc-colspec body root)
  (sxml:set-attr
   body
   (list
    'colspec
    (make-colspec
     (map
      (node-closure
       (ntype-names?? '(td th)))
      ((node-closure
        (ntype-names?? '(tr)))
        body))))))
```

The `make-colspec` function used in `calc-colspec` is one of the helper functions provided through an `xml2tex` package called `xmltex.latextable`, loaded at the beginning. With these helper functions and some understanding of Scheme and SXML, we have defined a conversion rule for a reasonable subset of HTML tables with `colspan` and `rowspan`. You can find the complete code in `xml2tex`'s repository [9].

## 5 Conclusion

Like it or not, more and more documents are created and stored in XML. Books which one can buy are no exception. Considering the changing circumstances regarding e-books and the Web, nearly any book may well be created in one of the XML-based formats. If you were to use a desktop publishing applications, you could go with some very nice WYSIWYG environments and not be bothered with the ill-reputed syntax of XML. However, of course, we must prefer

using `TEX` to such GUI software. This means, ultimately, writing a converter from XML-based formats to a `TEX`-flavored document.

We hope that `xml2tex` can help in this scenario. It works as a framework for using XML as a source for `LATEX`. All that's required is giving `xml2tex` a set of declarative mappings from each XML tag to an appropriate `LATEX` style. Aided by Scheme and SXML, you can even write a converter for a fairly complex XML document as needed.

To date, we have created dozens of commercial books using `xml2tex` while maintaining the manuscripts in a variety of XML-based formats.

## References

- [1] David Carlisle, "xmltex: A non-validating (and not 100% conforming) namespace aware XML parser implemented in `TEX`". <http://tug.org/TUGboat/tb21-3/tb68carl.pdf>
- [2] Ramon Casellas and James Devenish, "Welcome to the DB2`LATEX` XSL Stylesheets". <http://db2latex.sourceforge.net/>
- [3] Con`TEX`t Garden, "XML — Con`TEX`t wiki". <http://wiki.contextgarden.net/XML>
- [4] Shiro Kawai, "Gauche — A Scheme Implementation". <http://practical-scheme.net/gauche/>
- [5] Oleg Kiselyov, "SXML". <http://okmij.org/ftp/Scheme/SXML.html>
- [6] Håkon Wium Lie, "PhD Thesis: Cascading Style Sheets". <http://people.opera.com/howcome/2006/phd/>
- [7] Douglas Lovell, "`TEX`ML: Typesetting XML with `TEX`". <http://tug.org/TUGboat/tb20-3/tb64love.pdf>
- [8] Thomas A. Schmitz, "Getting Web Content and pdf-Output from One Source", <http://dl.contextgarden.net/myway/tas/xhtml.pdf>
- [9] Keiichiro Shikano, "k16shikano/xml2tex". <https://github.com/k16shikano/xml2tex>
- [10] World Wide Web Consortium, "Extensible Stylesheet Language (XSL) Version 1.1". <http://www.w3.org/TR/xsl11/>
- [11] World Wide Web Consortium, "Cascading Style Sheets (CSS) Snapshot 2010". <http://www.w3.org/TR/css-2010/>

◇ Keiichiro Shikano  
Tokyo, Japan  
k16.shikano (at) gmail dot com  
<https://github.com/k16shikano/xml2tex>

---

## MathBook XML

Robert A. Beezer

### Abstract

MathBook XML is an XML application to describe the structure of a technical document, such as a mathematics research article or textbook. This application is designed so an author can easily and clearly separate presentation from content. This then allows for direct XSL conversion to many formats, such as print, PDF, HTML, and EPUB.

L<sup>A</sup>T<sub>E</sub>X syntax is used to represent mathematics and clean L<sup>A</sup>T<sub>E</sub>X source is the result of one of the conversions, which is the precursor to print and PDF outputs. HTML output allows for variable granularity of the resulting pages and provides presentation and navigation interfaces appropriate for both small and large screens.

### 1 Introduction

There is a need for a source format that explicitly expresses the structure of a technical document, while making no assumptions whatsoever about presentation. This format should enhance an author's ability to specify this structure and free the author from eventual decisions about presentation. With such a format it is possible to then create versions for readers in a variety of formats, such as print, PDF, HTML, and EPUB, in addition to new formats not yet imagined.

### 2 The case for XML

eXtensible Markup Language (XML) is an extremely simple specification, with few reserved characters and a handful of rules about syntax. It suffers from a reputation of being verbose and overly-complicated. But this is primarily an artifact of its employment as a data-interchange language written by programs, not people. It is possible to design an XML application (the set of elements and attributes) which is natural for authoring and editing. MathBook XML is a case in point.

Technical documents such as mathematics research papers and mathematics textbooks are very structured. Chapters break into sections and sections may break into subsections, all with a hierarchical numbering scheme. There are numbered definitions, theorems, lemmas and corollaries. Figures, tables and references are all numbered. Then there are extensive cross-references to these items. XML naturally allows for an easy specification of the tree-like structure of such a document and cross-referencing is well-supported.

The eXtensible Stylesheet Language (XSL) provides a powerful declarative language for transforming XML into text, HTML or XML formats. In our case, it is possible to write out L<sup>A</sup>T<sub>E</sub>X source (resulting in PDF or print) and to write out HTML (resulting in web pages or as the basis of other formats such as EPUB). So the ease of authoring in XML with a carefully designed set of elements, along with the transformational power of XSL, make an XML application a natural choice.

### 3 MathBook XML source format

The elements in MathBook XML will feel natural to an author. To begin with, `<chapter>`, `<section>`, `<title>`, `<definition>`, `<theorem>`, and `<example>` are all exactly what you would expect them to be. At the paragraph level, frequently-used items have short abbreviations, familiar to anyone who knows HTML: `<p>`, `<em>`, `<q>`.

L<sup>A</sup>T<sub>E</sub>X includes at least four systems for cross-references: theorems (and similar environments), references, figures and tables, and equations. In MathBook XML, there is only one system, since a cross-reference is able to inspect the *type* of object it is pointing to and format the pointer in the way a reader expects (bare number, in brackets, or in parentheses). Optionally, you can choose to have prefixes on cross-references, such as “Theorem” or “Corollary” added automatically, with support for different languages, similar to what the `cleveref` package does.

#### 3.1 Mathematics

While one of the principal intents of MathBook XML is semantic markup, we have not chosen to go into this rathole for mathematics. L<sup>A</sup>T<sub>E</sub>X syntax works very well between the dollar signs and is understood by many authors. Indeed, it is the MathJax JavaScript library that makes this project possible. MathJax is able to do a very good job of expressing a wide range of L<sup>A</sup>T<sub>E</sub>X syntax (e.g. the `amsmath` package) in a web browser. So we can have quality output of displayed equations in both print/PDF and in HTML. MathBook XML is configured so an author need only specify macros once, and they can be employed in both the L<sup>A</sup>T<sub>E</sub>X and HTML output. Furthermore, they can be employed in the source code for diagrams described by the `TikZ` and `Asymptote` languages.

Inline math, single displayed equations (numbered or not) and multiline displays (numbered lines or not) are all supported. As an example of how authoring in XML is different, it is worth noting that in a multiline display, each line needs to be

delimited by an XML element. This then supports switching numbering on and off on a per-line basis, in addition to marking individual lines as targets of cross-references, all in a manner consistent with the syntax used for cross-references elsewhere in the document.

### 3.2 Graphics

For documents tracked by version control, or published with open licenses, or where the consistency of notation is controlled by simple macros, it is desirable to specify diagrams with a graphics language, rather than employing opaque raster formats. MathBook XML supports diagrams specified in *TikZ*, *Asymptote* or *Sage*, with full support for macros in the first two (and may be supported soon within *Sage graphics*). Diagrams are specified directly in the source, and then it is easy with XSL to isolate each one, in order to wrap it in the appropriate syntax so that it can be transformed to a standalone image. This step, and the subsequent processing, are all controlled by a single Python script, *mbx*. Whenever possible, the result is in Scalable Vector Graphics (SVG) format, so for example, the image scales fluidly on a web page when the reader zooms in or out.

### 3.3 Widgets on the web

There is a wide variety of JavaScript and Java applets which may be embedded on a web page. For example, *GeoGebra* is a sort of playground for exploring concepts and results from Euclidean geometry. It can provide a prepared interactive demonstration or a blank canvas with tools. One of our primary motivations has been to support the inclusion of the *Sage Cell Server* to allow readers the use of this extensive open source system for mathematics directly in a textbook. The *Sage Cell* is a text area, typically pre-loaded with *Sage* code from the author, which can be evaluated by pressing a button that sends the code to a publicly available server that then returns the results back into the reader's page. The reader is then generally free to edit the code to examine new situations, learning new mathematics or new *Sage* commands in the process. *WeBWorK* homework exercises are another conceivable addition for textbooks.

But a question remains: what to do with a dynamic element (such as a *GeoGebra* demonstration) in an output format that is inherently static, such as print. PDF readers can generally accommodate hyperlinks, but typically it is only Adobe's own *Acrobat Reader* that can display more complicated items such as animations.

## 4 $\LaTeX$ output

A principal feature of the  $\LaTeX$  output from MathBook XML is the isolation of style parameters in the preamble, and a body that is generic enough that it can be styled differently through style files or adjustments to the preamble. Parameters affecting style, such as margins, font size, and numbering depth can be controlled by parameters to the XSL processor and are not part of the MathBook XML specification of the source. With limited exceptions, the numbering of items is accomplished in the usual way, by letting the  $\LaTeX$  processing do that work. In other words, numbers of theorems, sections, figures, are not hard-coded into the  $\LaTeX$  created by the XSL transformation.

## 5 HTML output

HTML output from MathBook XML source allows for the creation of web pages at variable granularity ("chunks"). So a book can be broken up into one web page per subsection, if desired. Then CSS and JavaScript are employed to provide natural navigation interface elements, such as a clickable Table of Contents, plus "Previous", "Up" and "Next" buttons. On smaller screens (such as phones) the interface elements adjust to the limited space. The HTML output is also meant to be divorced from presentation, though the limited capabilities of CSS mean more information must be hard-coded, such as hierarchical numbering. But there is a good separation between the HTML and the CSS, which allows for different presentations.

The various numbering schemes that result from  $\LaTeX$  processing are identical to the numbering hard-coded into the HTML. For example, equation numbers in multi-line mathematics displays are identical in a PDF created via  $\LaTeX$  to the numbers that appear in a web page version rendered by *MathJax*. References to equations look and behave the same for both types of output.

## 6 Other output formats

*Sage* worksheets, *iPython* notebooks and *Sage Math Cloud* worksheets are all formats for web applications that allow a user to execute commands relevant to scientific computing, while maintaining a written record of input and output, with rich tools for annotating the results. Underneath, these documents are primarily HTML, adorned with CSS or JSON. We have experience with conversions of MathBook XML source to each of these formats, and have a usable conversion to *Sage* worksheets publicly available. With a new *MathJax* tool for rendering mathematics as



embedded SVG images, it appears that it is now possible to manufacture EPUB output that is as capable as PDF.

## 7 Philosophy

The primary purpose of MathBook XML is to make it easy for authors to capture their writing in a structured format. Then processing tools can be applied to create various output formats, while also insulating the author from a variety of technical details. While the supplied XSL conversions are meant to be usable and of high-quality, they are primarily a demonstration of the utility of the XML specification of the source. The conversions are modular enough that others can create new conversions by making small additions or changes, or by starting at a lower level and converting to some entirely different format.

## 8 Development

Code for MathBook XML is open-source (GPL license) and lives in a GitHub repository [1]. The latest changes are on the `dev` branch. There is a sample article in the examples directory of the distribution, which is heavily annotated and tries to contain at least one example of every feature possible, and so serves as accurate documentation. Support

questions and feature requests take place on a Google Groups forum. Pointers to these resources, and others, can be found at the website [2]. The Gallery at the website contains links to large examples of MathBook XML in production use.

## 9 Acknowledgements

This is joint work with David Farmer, Steve Blood, and Michael DuBois. Financial support has come from the UTMOST project (National Science Foundation DUE-1022574), a Shuttleworth Foundation Flash Grant, and the University of Puget Sound.

## References

- [1] Robert A. Beezer. GitHub repository: `rbeezer/mathbook`. Available at <https://github.com/rbeezer/mathbook>.
- [2] Robert A. Beezer. MathBook XML. Available at <http://mathbook.pugetsound.edu>.

◇ Robert A. Beezer  
 Dept. of Mathematics and  
 Computer Science  
 University of Puget Sound  
 Tacoma, Washington, USA  
`beezer (at) pugetsound dot edu`  
<http://buzzard.pugetsound.edu>

## Can $\LaTeX$ profiles be rendered adequately with static CSS?

William F. Hammond

### Abstract

MathJax demonstrates that heavy customization of CSS with JavaScript and webfonts provides good platform-dependent rendering. The issue with MathJax is speed, not quality. There has been and continues to be intense development with CSS. One may speculate that, as CSS continues to evolve, static CSS may entirely suffice not only for HTML documents with mathematics but also for the direct online rendering of profiled  $\LaTeX$  documents when presented using XML syntax.

My purpose is to report on some of what can now be done, to indicate how I would like to see things develop, and to stir interest in the  $\LaTeX$  community for incorporating the ideas of CSS into print typesetting.

### 1 Background

This article is an elaboration of what I said in my talk at TUG 2014 (slides are available at <http://www.albany.edu/~hammond/presentations/tug2014>). About half the time in the talk was devoted to showing  $\LaTeX$  profiles styled with CSS in a web browser. In fact, the slides themselves were such. Figure 1 is a screenshot of static CSS styling for math in a  $\LaTeX$  profile. At this point satisfactory results are being obtained with three of the major browsers using three different wide-coverage Unicode fonts. (Corresponding to the fonts there are three different stylings that differ from each other, aside from font invocations, only in handling a few things.)

For those who have less than satisfactory interaction with the XML slides, there is an HTML version provided at the previous url. Aside from the slideshow poster, which was written directly in HTML, the slides and all of the materials linked therefrom originated as source prepared for the  $\LaTeX$  profile of the GELLMU Didactic Production System, <http://www.albany.edu/~hammond/gellmu/>, which may be regarded as a base for spawning other profiles and which will be used as the profile of discourse here.

CSS [1] stands for *Cascading Style Sheets*. It is the standard design language used in presenting HTML (Hypertext Markup Language), as well as XML (eXtensible Markup Language) [2] applications, in web browsers.

A  $\LaTeX$  profile [8] is a dialect of  $\LaTeX$  [4] with a fixed command vocabulary, where all macro expan-

sions must be effective in that vocabulary, having a well-defined XML shadow.  $\LaTeX$  profiles are suitable domains for defining reliable translations to other profiles and, where sensible, to other markup languages.

The author recalls a conversation at a meeting in 2002 on the question of whether CSS might someday suffice for rendering segments of MathML (Mathematical Markup Language) in web pages. The participants agreed that it might be possible, but a sufficiently wide deployment of sufficiently capable CSS engines would then be the issue.

By early 2006 I had begun writing a CSS stylesheet for the XML guise of the  $\LaTeX$  profile of the GELLMU didactic production system [5, 6, 7]. Especially where mathematics is concerned, the richness of the vocabulary of profiled  $\LaTeX$  compared to the vocabulary of MathML makes it easier to think about<sup>1</sup> rendering math with CSS. The CSS rendering of profiled  $\LaTeX$  in 2006 was quite limited, not remotely close in quality to the rendering of MathML in a browser like Firefox, but still of some use.

By 2010 Davide Cervone’s MathJax, <http://www.mathjax.org>, if not Cervone’s earlier project *js-math*, had demonstrated that with heavy (and slow) JavaScript-based customization for available fonts, web browser, and computing platform, MathML could be rendered with CSS.

During the time that MathJax was being developed George Chavchanidze of Opera Software had been pursuing the idea of basing native browser support for MathML solely on static CSS.<sup>2</sup> Following that work the W3C Math Working Group in 2011 produced a W3C recommendation entitled “A MathML for CSS Profile” [9] that suggested restricted use of MathML by content generators interested in having their content rendered with CSS.

In early 2014 I learned about Frédéric Wang’s idea, following the earlier work of Chavchanidze and quite apart from MathJax, of providing “fallback” rendering of MathML in web browsers lacking native support for MathML, taking advantage of relatively new ideas in CSS, particularly CSS flexible boxes [10], without heavy customization for particular circumstances.

I have spent the last few months seeing how these new ideas in CSS could be used to improve the static CSS presentation of GELLMU’s  $\LaTeX$  profile. It is work in progress. My purpose is to report on some of what can now be done, to indicate how I

<sup>1</sup> (but probably not actually easier)

<sup>2</sup> Followers of  $\LaTeX$ 3 might think of this as an effort, unlike that in  $\LaTeX$ 3, to leverage the *design layer* without much in the way of apparent new support from the *programming layer*.

$$\frac{1}{1 + \frac{e^{-2\pi\sqrt{5}}}{1 + \frac{e^{-4\pi\sqrt{5}}}{1 + \frac{e^{-6\pi\sqrt{5}}}{\dots}}}} = \left( \frac{\sqrt{5}}{1 + \sqrt[5]{5^{3/4} \left( \frac{\sqrt{5}-1}{2} \right)^{5/2} - 1}} - \frac{\sqrt{5}+1}{2} \right) e^{2\pi/\sqrt{5}}$$

Figure 1: Static CSS styling of profiled L<sup>A</sup>T<sub>E</sub>X

would like to see things develop, and to try to stir the interest of the L<sup>A</sup>T<sub>E</sub>X community in incorporating CSS ideas into print typesetting.

## 2 Processing

In another talk at TUG 2014, by S. K. Venkatesan and C. V. Rajagopal, one of the slides had this poetic line:

T<sub>E</sub>X is poured into the XML mould, and DTD  
is used as the sieve.

“DTD” refers to the document type definition. Document type definitions are in one-to-one correspondence with L<sup>A</sup>T<sub>E</sub>X profiles. There are at least two reasons for sifting:

1. To know that a correctly written processor will reliably produce correct results.
2. To put the L<sup>A</sup>T<sub>E</sub>X under a framework that facilitates processing by any of the many software libraries, written in various programming languages, that operate on XML.

Those interested will be able to find more information about this in many places including, for example, *The L<sup>A</sup>T<sub>E</sub>X Web Companion* [3].

For rendering a L<sup>A</sup>T<sub>E</sub>X profile with CSS a small amount of “server-side” processing (independent of fonts, browser, and platform) may be used to dress the regular XML guise of the L<sup>A</sup>T<sub>E</sub>X so that it may be more easily addressed with CSS. In this report the main concern is with mathematics, but I should note that the whole of a profiled L<sup>A</sup>T<sub>E</sub>X document must be styled with CSS for presentation in a web browser.

### 2.1 Source

I will illustrate briefly with a tiny example under GELLMU’s L<sup>A</sup>T<sub>E</sub>X profile. This is the source `tg.glm`:

```
\documenttype{article}
\title{test}
\begin{document}

One has
\[ \Gamma(3) = 2! \]
\end{document}
```

### 2.2 Author-level XML

Under main track GELLMU processing, based only on syntax, not vocabulary, this resolves to the following XML instance `tg.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css"
  href="gellmuart.css"?>
<!DOCTYPE article SYSTEM "axgellmu.dtd">
<article stem="tg">
<preamble><title>test</title></preamble>
<body>
<parb>One has
<displaymath>
<Gamma/>(3)<eqs/>2<exc/>
</displaymath>
</parb>
</body>
</article>
```

The XML tags should be self-explanatory except for `<parb>`, which indicates the paragraph begun with a blank line, the empty element `<eqs/>` coming from the “=” (that makes it possible for downstream decisions to be made on the “=”), and the `<exc/>` coming from the “!”.

### 2.3 Elaborated XML

In main track processing under the GELLMU Didactic Production System this “author-level” XML that closely shadows the original source is processed, preparatory to translation toward either regular L<sup>A</sup>T<sub>E</sub>X or HTML, to the following elaborated XML instance `tg.exml`:

```
<?xml version="1.0" encoding="UTF-8">
<?xml-stylesheet type="text/css"
  href="gellmuart.css"?>
<?centralStyled?>
<!DOCTYPE article SYSTEM "uxgellmu.dtd">
<article stem="tg">
<preamble><title>test</title></preamble>
<body>
<parb>One has
<displaymath>
```

```
<Gamma/>(3)<equals/>2<exc/>
</displaymath>
</parb>
</body>
</article>
```

The only change noticeable in this very simple example is that the “=” has now become `<equals/>` because it is within a mathematical container. (Other changes that would happen with a more complicated document would be resolution of cross-references and assignment of section numbers.)

## 2.4 Dressed XML

To prepare for CSS it is necessary that every nugget of character data inside math zones be wrapped in a tag indicating whether the nugget is numeric, word-like, or operator-like, so that the questions of whether to use an upright or italic font and how to space may be addressed.

Such “dressing for CSS” leads to `tg-lm.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css"
  href="gellmualm.css"?>
<!DOCTYPE article SYSTEM "vxgellmu.dtd">
<article stem="tg">
<preamble><title>test</title></preamble>
<body>
<parb>One has
<displaymath mlvl="1"
  mchld="me|bal|me|mx|me">
<me name="Gamma" mlvl="2" mchld="Gamma"
  ><Gamma/></me>
<bal mlvl="2" mchld="mx"
  ><mx type="number">3</mx></bal>
<me name="equals" mlvl="2" mchld="equals"
  ><equals/></me>
<mx type="number">2</mx>
<me name="exc" mlvl="2" mchld="mx"
  ><mx type="character">!</mx></me>
</displaymath>
</parb></body>
</article>
```

One will see that the mathematics has been greatly elaborated. The elaboration creates hooks for possible use in CSS selectors. Note, in particular, the values of the attribute `type` on the element `<mx>`. The attribute `mlvl` on almost every element inside a math zone indicates how deep that element is in the math zone (a selection issue not foreseeably addressable by CSS). The attribute `mchld` on math containers lists the names of the child elements (another selection issue not currently addressable in CSS).

### 2.4.1 Balancing parentheses

Another thing to notice is that

```
<Gamma/>(3)
```

has become (with some simplification for human clarity):

```
<Gamma/><bal><mx type="number">3</mx></bal>
```

That is, at this stage of processing the parentheses have been replaced with the element `<bal>...</bal>`, “bal” for “balanced”, which at the source level is `\bal{...}`, corresponding to `\left(...\right)` in regular L<sup>A</sup>T<sub>E</sub>X.

### 2.4.2 A bit of CSS for `<mx>`

By way of example, relevant code from the linked CSS, available from <http://www.albany.edu/~hammond/webstyle/gellmualm.css>, includes:

```
mx[type="letter"] {
  font-style: italic;
}
mbox mx[type="letter"] {
  font-style: normal;
}
mx[type="number"] {
  font-style: normal;
  font-size: 0.92em;
}
[mlvl="1"]>mx[type="number"] {
  font-style: normal;
  font-size: 1em;
}
}
```

Many things in the style sheet are debatable, even questionable. In particular, the font-size adjustment for `mx[type="number"]` is based on a personal judgement that numbers at levels greater than 1 were too large compared to letters.

As mentioned above, the name of the dressed XML instance is `tg-lm.xml`. The pre-suffix “-lm” is a mnemonic reference to the “Latin Modern” font, matching the “lm” in the name `gellmualm.css` of the linked style sheet. Because at this time there is a certain lack of standardization in fonts used on the web, I am providing a different style sheet for each font. This must mean, of course, that the style sheet is being used to control the font, which, in turn, means that I must be serving web fonts.

## 3 Fonts

For most of the history of the World Wide Web the fonts used in web browsers have been the fonts found on the user’s computer. Relatively recently what are called “web fonts” have appeared and have gained support in major web browsers. Web fonts are

fonts served through the web, usually from the site where an HTML document is posted. The MathJax project has made a great deal of use of web fonts. My understanding is that web fonts may be made from either OpenType fonts or TrueType fonts.

I know relatively little about fonts, whether in the world of the web or in the L<sup>A</sup>T<sub>E</sub>X world. A few of the things that I have learned include:

- Unicode fonts, i.e., fonts whose glyphs are referenced by the Unicode value for the corresponding character, are becoming the standard both on the web and for L<sup>A</sup>T<sub>E</sub>X with the new T<sub>E</sub>X engines (*xetex* and *luatex*).
- OpenType fonts seem to be becoming the standard.
- A free program by Jonathan Kew of Mozilla (the originator of *xetex*) called *sfnt2woff*<sup>3</sup> will convert an open type font file, say `foofont.otf`, or a true type font file, say `foofont.ttf`, to a web font (`foofont.woff`).
- A CSS `@font-face` directive is used to tie a web font on your server to a `font-family` name, possibly also with a `font-weight` and `font-style` specification.

#### 4 Casting for glyphs

There comes the question in styling a L<sup>A</sup>T<sub>E</sub>X profile with CSS of what one is to do with the four “casting” commands: `\mathbb`, `\mathcal`, `\mathfrak`, and `\mathscr`. In L<sup>A</sup>T<sub>E</sub>X, as one knows, each of them takes as argument a Latin letter and produces, respectively, a double-struck, calligraphic, Fraktur, or script version. Apart from Fraktur, it can be argued that these are mere stylistic variations. But most mathematicians are inclined to regard the original Latin letter and these four casts as five semantically distinct symbols. While in a translation to MathML, it might be reasonable to generate appropriate Unicode points for the casts,<sup>4</sup> that would be a bit out of scale for styling the XML guise of a profiled L<sup>A</sup>T<sub>E</sub>X document with CSS where the author has used one of these commands.

Of course, the author could just put the appropriate Unicode points in the source. Aside from that, an off-track approach is to specify a sequence of matching old fonts such as `msbm10`, `cmsy10`, `eufm10`, and `rsfs10` for the four casts.

<sup>3</sup> C source available from <http://people.mozilla.org/~jkew/woff/woff-code-latest.zip>—I found it quite easy to build with `gcc`.

<sup>4</sup> This ignores that the Unicode standard seems to have merged calligraphic and script, and the standard provides only three incomplete “alphabets”.

If the Unicode standard were amended to fill the gaps left in the three alphabets that correspond to previously assigned glyphs (not characters in the abstract sense), then in the “dressing process” one could use offsets to those alphabets for `bb`, `frak`, and `scr`.

Finally, by way of trying to nudge the guardians of Unicode, I would like to note that in the document “The STIX Package” (`stix.pdf`) accompanying the 2014 release of the STIX fonts and found in T<sub>E</sub>X Live 2014, the table in section 3 indicates support for a number of `\mathxx` commands beyond what is provided in Unicode. The author has been shown browser-private CSS properties that may be used to access alternate glyphs for Unicode points in OpenType fonts. This gives hope that eventually there will be better ways to use CSS to access such alternate glyphs.

#### 5 Fractions

Fractions may be rendered reasonably well using CSS tables. There are, however, two important things to note. First, each of the numerator and denominator must be the sole table-cell in a table-row. The bar that separates numerator and denominator may be provided as the “collapsed” border-bottom of the numerator with the border-top of the denominator. This arrangement for the bar works with table-rows. However, table-rows cannot contain essentially arbitrary content, while table-cells can. Thus, the dressing of the profile must re-arrange things to be as if `\frac{a}{b}` had been marked up as `\frac{{a}}{{b}}`. This much is a slight inconvenience but not a problem.

The second thing to note is a problem. There does not seem to be any reasonable way at present to have horizontally adjacent fraction bars align with each other. It is not an obstruction to comprehensibility, but it leaves an appearance one does not want. It is part of a much larger concern with vertical alignment for the math in L<sup>A</sup>T<sub>E</sub>X profiles.

The author understands that vertical alignment is a subject of continuing work within the CSS community.

#### 6 Borders as balancers

Previously (in section 2.4.1) I mentioned `\bal{...}` as the profile’s version of `\left(...\right)`. At the stage of dressing when all character data in math zones is being wrapped in tags the character “(” is replaced with `<bal>` and the character “)” is replaced with `</bal>`. Aside from the fact that this is important for trapping the author error of having unbalanced parentheses because the output will not

parse correctly without balance, it is important because CSS (not unlike L<sup>A</sup>T<sub>E</sub>X) is largely about boxes. Every XML element gives rise to a box. CSS properties control that box.

One might eventually hope that perfectly sized parentheses for a given box might be provided using CSS as a border segment object. What works now is the following:

```
bal {
  align-self: center;
  display: inline-block;
  margin-left: 0.15em;
  margin-right: 0.15em;
  padding: 0.2ex 0.2em 0.2ex 0.2em;
  border-left: 0.2ex solid;
  border-right: 0.2ex solid;
  border-radius: 0.5em;
}
```

Here is a screenshot that illustrates this handling of `\bal{}`:

Notice how each pair of parentheses, allowing for CSS-specified padding of boxes, fits its box precisely. There is no limit to the number of sizes. How easy it could be for an author to omit one of the parentheses at the end if they were all of the same size as here:

$$\frac{\pi}{2} \left( 1 + \left( \lim_{n \rightarrow \infty} \left( \log(n+1) - \left( \sum_{k=1}^n \frac{1}{k} \right) \right) \right) \right)$$

Among my wishes for the future of CSS are new border-decoration properties for the four sides of a box that would enable one to have precisely fitting parentheses, braces, and brackets. Further one might wish eventually to be able to attach to a point on the border of a box a small piece of drawing, similar to a picture environment drawing in L<sup>A</sup>T<sub>E</sub>X.

Thus, for example, radicals (from `\sqrt{[]}`), which are presently rather fragile with the CSS styling now available (except for the top line that presents well when styled as the `border-top` of the radicand), would be handled better using the `border-top` of the radicand and a segment of the `border-left` of the radicand with a radical hook drawn from the bottom of the segment on the left. Failing that one can even now make Orwellian “victory radicals” consisting simply of the top and left borders of the radicand.

William F. Hammond

## 7 Flexible boxes

### 7.1 *underset*

The L<sup>A</sup>T<sub>E</sub>X profile in use here does not, though it certainly could, provide `\lim` as a command. The limit in the previous display is generated as an *underset*. Explicitly, the corresponding source is:

```
\underset{n \rightarrow \infty}{\mbox{lim}}
```

In the XML everything needs a name. The first argument of *underset*, the “decoration”, has the name *deco*, while the second argument, the “base”, has the name *expr*.<sup>5</sup>

It would be rather heavy-handed to style an *underset* as a table. Instead it uses a new concept in CSS [10]: the flexible column.<sup>6</sup>

The command *underset* is treated in this segment of the CSS stylesheet:

```
underset {
  display: inline-flex;
  flex-direction: column;
  align-items: center;
  vertical-align: text-top;
  justify-content: flex-start;
}
underset > expr {
  order: +1;
  padding: 0;
  margin-top: -0.2ex;
  margin-bottom: -0.3ex;
}
underset > deco {
  font-size: 0.8em;
  padding: 0;
  order: +2;
}
```

The `order` property for the children indicates the order from top to bottom in which the children should be displayed. In this case the first child is to be displayed second and the second child first.<sup>7</sup> The ability to arrange the order of children is useful. (Its usefulness is made less important, however, if the XML is processed for “dress”.)

This segment of CSS above is not very robust. The `vertical-align` specification may or may not be what one will ultimately want. If it is used, the idea is to align the *underset* box within the parent. It is not supposed to be effective if the parent is a flexible box (row or column), in which

<sup>5</sup> The argument order for MathML’s corresponding *munder* is reversed.

<sup>6</sup> As an exercise, the reader might ponder why it would not work to style a fraction as a flexible column.

<sup>7</sup> With MathML’s *munder* it would be first first and second second.

case a property called `align-self` with different permitted values can be used. It's not fully clear which display types of the parent are appropriate if `vertical-align` is to be effective. Moreover, it's not clear what point on a flexible column is being aligned, say, in a parent of type `inline-block`. I can report that for one browser the display changed for *underset* and *overset* with a version upgrade during July, 2014. The top and bottom margin settings for *expr* are a font-dependent attempt to adjust for the inadequacies of `vertical-align`.

## 7.2 *sum*-like operators

The sum in the previous display also involves flexible boxes, in this case two of them, one a row and one a column. In the  $\text{\LaTeX}$  profile at hand, sums, integrals, and products are handled together for most purposes, and I call them the “sip” (the first letters of sum, integral, and product) elements. (There could be more of these: unions, intersections, coproducts, . . . , but no others are presently in the profile.) Where in regular  $\text{\LaTeX}$  the corresponding commands reference the operator symbols, in the profile they reference the whole structure. Nonetheless, the markup is almost the same as with regular  $\text{\LaTeX}$  except that an explicit termination of the object of the operation is required. For the example in the display of section 6, the markup is

```
\sum_{k=1}^n\frac{1}{k}\sum:
```

This could be marked up in a manner close to its XML guise:

```
\sum{\siphead{\lower{k=1}\upper{n}}
  \sipbody{\frac{1}{k}}}
```

The CSS scheme used is that *sum* is a flexible row, while its first child *siphead* is a flexible column given that its parent is a *displaystyle* sum. At the stage of dressing an *mx* containing the summation symbol is inserted between the *upper* and the *lower*. The *sipbody*, which is the second child of the *sum*, contains the object of the summation — in this case the fraction  $1/k$ . A listing of the relevant CSS code follows. Note that the property `align-self` should govern vertical alignment of the *sum* in its parent in the case that the *sum* is itself inside a flexible row, while the property `vertical-align` should govern the case that the *sum* is inside an inline block (which I have taken to be the default display style for any expression more complicated than a single symbol).

```
sum, int, prod {
  align-self: center;
  vertical-align: middle;
  margin-left: 0.2em;
  margin-right: 0.2em;
```

```
  display: inline-flex;
  flex-direction: row;
  justify-content: flex-start;
  align-items: center;
}
siphead {
  vertical-align: middle;
  align-self: center;
  display: inline-flex;
  flex-direction: column;
  justify-content: flex-start;
  align-items: center;
}
siphead > upper {
  font-size: 0.6em;
  order: -1;
  line-height: 2.5ex;
  min-height: 1.5ex;
  margin-bottom: 0.15ex;
}
siphead > mx {
  order: 0;
}
siphead > lower {
  order: 1;
  font-size: 0.6em;
  line-height: 2.5ex;
  min-height: 2.5ex;
  margin-top: 0.15ex;
}
sipbody {
  display: inline-block;
  align-self: center;
  padding-left: 0.05em;
}
```

## 8 Why?

It is important to explore all avenues for making mathematical content fully available online in proper online formats. This includes the world of small screens as found on “smart phones” and the world of “e-books”.

One hopes that the design concepts in CSS<sup>8</sup> for online content eventually become adequate for handling mathematics — they are not far from that now — and that those concepts come to be supported in all major web browsers.

With fully robust static CSS for  $\text{\LaTeX}$  profiles the gain for the online presentation of mathematics could be:

1. Faster browsing of math online.

<sup>8</sup> general concepts not particularly tied to mathematics

2. The potential for greater control in the presentation of math online.
3. Elimination of the need for special handling of math in the online world.

If this future is realized, there are two other things to note:

- A L<sup>A</sup>T<sub>E</sub>X profile can be robustly translated (on the server side) to HTML with MathML. In that process the output can be “dressed” so that it can be styled with static CSS in a way that has an almost identical presentation in web browsers to that of the profile itself presented with its static CSS. There should be no need to observe the restrictions of “MathML for CSS” [9].
- Web-served HTML with MathML will have the advantage over web-served L<sup>A</sup>T<sub>E</sub>X profiles of having mathematical expressions that can be “pasted” into a computer algebra system.

## References

- [1] Bert Bos, Tantek Çelik, Ian Hickson, & Håkon Wium Lie, *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification* World Wide Web Consortium Recommendation, 7 June 2011, <http://www.w3.org/TR/2011/REC-CSS2-20110607>.
- [2] Tim Bray, Jean Paoli, C.M. Sperberg-McQueen, Eve Maler, & François Yergeau, *Extensible Markup Language (XML) 1.0 (Fifth Edition)*, World Wide Web Consortium Recommendation, 26 November 2008, <http://www.w3.org/TR/2008/REC-xml-20081126>.
- [3] Michel Goossens and Sebastian Rahtz et al., *The L<sup>A</sup>T<sub>E</sub>X Web Companion*, Addison-Wesley, 1999.
- [4] Leslie Lamport, *L<sup>A</sup>T<sub>E</sub>X: A Document Preparation System*, 2nd edition, Addison-Wesley, 1994.
- [5] William F. Hammond, *The GELLMU Manual*, 2007, <http://mirror.ctan.org/support/gellmu/doc/glman.pdf>, or <http://mirror.ctan.org/support/gellmu/doc/glman.xhtml> (XHTML+MathML).
- [6] William F. Hammond, “GELLMU: A Bridge for Authors from L<sup>A</sup>T<sub>E</sub>X to XML”, *TUGboat*, vol. 22 (2001), pp. 204–207; also available online at <http://www.tug.org/TUGboat/tb22-3/tb72hammond.pdf>.
- [7] William F. Hammond, “Dual presentation with math from one source using GELLMU”, *TUGboat*, vol. 28 (2007), pp. 306–311; also available online at <http://www.tug.org/TUGboat/tb28-3/tb90hammond.pdf>. A video recording of the presentation at TUG 2007, July 2007, in San Diego is available at <http://river-valley.zeeba.tv/conferences/tug-2007/>.
- [8] William F. Hammond, “L<sup>A</sup>T<sub>E</sub>X profiles as objects in the category of markup languages”, *TUGboat*, vol. 31 (2010), pp. 240–247; also available online at <http://www.tug.org/tugboat/tb31-2/tb98hammond.pdf>. A video recording of the presentation at TUG 2010, June 2010, in San Francisco is available at <http://river-valley.zeeba.tv/conferences/tug-2010/>.
- [9] Bert Bos, David Carlisle, George Chavchanidze, Patrick D. F. Ion, & Bruce Miller, “A MathML for CSS Profile”, World Wide Web Consortium Recommendation, 7 June 2011, <http://www.w3.org/TR/mathml-for-css/>.
- [10] T. Atkins, fantasai, & Rossen Atanassov, ed., “CSS Flexible Box Layout Module Level 1”, World Wide Web Consortium, last call working draft (work in progress), March 25, 2014, <http://www.w3.org/TR/2014/WD-css-flexbox-1-20140325/>, (latest: <http://www.w3.org/TR/css-flexbox-1/>).

◇ William F. Hammond  
 University at Albany, Albany,  
 New York  
 and San Diego, California  
 hammond (at) albanys dot edu  
<http://www.albany.edu/~hammond/>



---

**TUG 2014 abstracts**

Editor's note: Slides and other related information for many of the talks are posted at <http://tug.org/tug2014/program.html>.

— \* —

**Kaveh Bazargan**

*Creating a L<sup>A</sup>T<sub>E</sub>X class file using a graphical interface*  
 Writing L<sup>A</sup>T<sub>E</sub>X class files is a very specialized job, needing intimate knowledge of T<sub>E</sub>X. This job can be simplified by separating parameters from the T<sub>E</sub>X commands themselves. Using such parameterization, a user can simply change the parameters and obtain the change required. The technique becomes much more useful if a graphical user interface is used to modify the parameters interactively, and if the user gets instant feedback by seeing the result in the typeset document. I will demonstrate one such system, namely Batch Commander, created using LiveCode. I will show that many uncomplicated class files can be created by a user with minimal knowledge of T<sub>E</sub>X.

**Kaveh Bazargan**

*PDF files both to print and to read on screen*

PDF started life as a reliable format for distributing a document for printing. In the electronic age when most documents are never printed, many people have predicted the demise of PDF, in favour of other formats such as EPUB. But PDF has remained resilient and is likely to remain so for the foreseeable future. However, in general, a print PDF is not ideal for reading on screen, and vice versa. So publishers either compromise, by putting minimal links and enhancements, or by producing two completely separate PDFs.

I will present methods by which we can use T<sub>E</sub>X to create PDF files that print perfectly, but that are enhanced for online reading too.

**Karl Berry**

*Building T<sub>E</sub>X Live*

An overview of how T<sub>E</sub>X Live is constructed and updated, both so-called packages and programs. Here, packages contain only material which is interpreted, such as L<sup>A</sup>T<sub>E</sub>X add-ons, fonts, and scripts; these are updated throughout the year, as updates are released to CTAN. In contrast, programs are compiled binaries and are, almost always, updated only for the annual release. A document with details on all these topics (and much more) is available at <http://tug.org/texlive/doc/tlbuild.html> (or [.pdf](#)).

**Dave Crossland**

*Metapolor: Why Metafont is finally catching on*

This presentation follows up on my paper, "Why

didn't METAFONT catch on?" presented at TUG 2008 and in *TUGboat* 29:3. In March 2013, Dave contacted the developers of the Metaflap web application, and met with designer Simon Egli in New York. Simon proposed a radical idea to enable typeface designers to harness the power of METAFONT's parametric capabilities without requiring them to write any METAFONT code. After a year of prototyping, Dave and Simon secured financial support from the Google Fonts project to fully develop such a tool with a team of collaborators from around the world. They call it Metapolor.

**Ward Cunningham**

*Another wiki? OMG why?*

We will reflect on the first wiki, what problem it solved, and what problems it sidestepped. We'll acknowledge the most famous wiki, Wikipedia, and especially recognize what they found the need to change. We will then describe the technological and social opportunity for another wiki, again a service nobody asked for, but now built with technology twenty years further along in the evolution of the web. For this conference: <http://tug.fed.wiki.org>.

We've made wiki servers simpler by moving rendering and sharing into the browser. We've used federation to solve some problems that plague other implementations, and to open an innovation space unserved by web technology until now.

Finally we'll declare our love for those who write, even a little, for thoughtful writing inoculates us from the ravages of consumerism.

**Paulo Ney de Souza**

*A call for standards on the way to internationalization of T<sub>E</sub>X*

Even though T<sub>E</sub>X is able to write and process documents in hundreds of languages, its own internal organization regarding use of language is next to nil. On this talk we will show how the introduction of standards like ISO-639, ISO-15924, ISO-3166-1 and RFC-5646 are producing real cross-pollination among projects like Babel, Polyglossia, BIBL<sup>A</sup>T<sub>E</sub>X and CSL and driving the internationalization of T<sub>E</sub>X further.

**Michael Doob**

*Using animations within L<sup>A</sup>T<sub>E</sub>X documents*

T<sub>E</sub>X has always been in its heart a program for creating beautiful books. As T<sub>E</sub>X matured, markup was simplified using L<sup>A</sup>T<sub>E</sub>X, packages for creating beautiful tables of contents and indexes appeared, and colour was added. Graphics packages were also added allowed beautiful illustrations, but the resulting output was in essence a book. This era of progress is disappearing.

We now read novels on our smart phones and mathematical papers on our tablets. The objects being viewed are less like traditional books and have potential to be much more exciting. Fortunately,  $\text{\TeX}$  is flexible enough to adapt to the changing environment. As a step supporting this change, we survey the varied techniques available to create animations within  $\text{\LaTeX}$  documents.

### David Farmer

*Converting structured  $\text{\LaTeX}$  to other formats*

I will describe a project which converts research papers and textbooks in mathematics from  $\text{\LaTeX}$  to HTML, providing an alternative to online PDF documents. This project specifically targets journal papers and books, as examples of structured documents. Making use of the underlying structure of the  $\text{\LaTeX}$  document, the output more closely preserves the reader's ability to visually scan the document's contents and seamlessly incorporates references and citations.

### Doug McKenna

*Liberate  $\frac{\text{\TeX}}{\text{C}}$  Top part: JSBox*

A work in progress, JSBox is a self-contained library — written in portable C — that instantiates sandboxable,  $\text{\TeX}$ -language interpreters within the memory space of any C, Objective-C, or C++ 32- or 64-bit client program.

Built and documented anew, JSBox is faithful to the  $\text{\TeX}$  language's primitives, syntax, typesetting algorithms, measurements, data structures, and speed. At the same time, it fixes — in an upwardly compatible manner — a variety of important problems with or lacunæ in the current  $\text{\TeX}$  engine's implementation. These include integral support for 21-bit Unicode, namespaces, OpenType font tables and metrics, job-specific 8-bit to 21-bit Unicode mapping, run-time settable compatibility levels, full 32-bit fixed-point math, and more. Especially pertinent to interactive applications — such as an eBook reader — is that all of a document's pages can optionally be kept as  $\text{\TeX}$  data structures in memory after a job is done, with direct random access of any requested page exported to the client program's screen without file I/O or DVI or PDF conversion if unneeded.

Tracing (notably including recursive expansion, the re-tracing of interrupted commands, alignments, math, etc.) and all error messages have been significantly improved over what  $\text{\TeX}$  does. (A detailed article on JSBox's tracing appears in this issue, pp. 157–167.)

The author will demo what JSBox can do now, and discuss what it could do in the future.

### Doug McKenna

*Liberate  $\frac{\text{\TeX}}{\text{C}}$  Bottom part: literac*

`literac` is a command-line program that converts source code written in C, C++, Objective-C, Swift, Go, or other languages that use C-style commenting syntax (i.e., `//` and `/* ... */`) into a  $\text{\LaTeX}$  document.

Computer code is typeset verbatim (with optional line numbers). Comments, on the other hand, are stripped of their delimiters, presented in different styles based on context, and merged into paragraphs if possible. A significant amount of attention is paid to ensuring that  $\text{\TeX}$  does “the right thing” in numerous edge cases. Within comments, a few special commands support common typesetting tasks, including verbatim and auto-verbatim code quotes, macros, moving source material forward for typesetting later in the document, inserting arbitrary  $\text{\TeX}$  code for math displays, tables, or footnotes, suppressing both code and comment lines from being typeset, and visual cues for dividing a large program in one source file into chapters, sections, subsections, a README file, etc. This fosters better documented C code without imposing an intermediate CWEB (CWEAVE/CTANGLE) step on the source code's developer/tester.

The single implementation file, `literac.c`, is documented in the `literac` style it implements. It typesets itself into a 200+ page document that is its own user manual, its own implementation, the explanation of the program's internal design, and an excellent test suite for itself. Its author built `literac` to typeset the 90,000+ lines of source code — half of them comments — of the JSBox library, as commented using `literac`'s rules and commands.

### Frank Mittelbach

*Regression testing  $\text{\LaTeX}$  packages with Lua*

For many years,  $\text{\LaTeX} 2_{\epsilon}$  has used a custom Perl script to perform regression testing on a large number of test files to ensure that any changes to  $\text{\LaTeX}$  do not break anything. This tests have also been useful in highlighting changes in  $\text{\TeX}$  Live. One of the first steps in the modern development of the  $\text{\LaTeX} 3$  code was to produce a similar system to ensure that the development process was as smooth as possible. This build system, which also handled documentation typesetting and CTAN archiving, was written with a Makefile system, and eventually a Windows batch script was written as well.

These days, all  $\text{\TeX}$  distributions ship with Lua $\text{\TeX}$ , which includes a standalone Lua interpreter; for the first time,  $\text{\TeX}$  users can write platform-independent scripts that run without needing to install any additional frameworks. (E.g., Perl under

Windows.) Joseph Wright, accordingly, has rewritten the  $\LaTeX$ 3 build scripts in Lua (uploaded to CTAN in June 2014) to avoid maintaining two separate systems. As an added bonus, this new system, named `l3build`, is flexible and extensible enough to be used for any  $\TeX$  package, and we hope the community will now take advantage of having an easy-to-use regression test suite available.

In this presentation, we will discuss how we use the `l3build` system, how we hope others will use it, and why regression testing is so important.

### Ross Moore

*“Fake spaces” with pdf $\TeX$  — the best of both worlds*

When Donald Knuth wrote  $\TeX$  he chose to omit space characters from the output, but instead carefully position the start of each word and punctuation character. This was to be able to better handle the idea of full-justification, as done by clever typists on manual typewriters.  $\TeX$ 's visual output seems clearly superior because of this.

Nowadays, however, other word-processing and text-presentation software seems to have largely abandoned full justification. Instead, window sizes can be resized causing the text to reflow on-the-fly. The presence of a space character as a word delimiter is important for this to work properly.

With the 2014 version of  $\TeX$  Live, new primitives such as `\pdffakepace` are included within pdf $\TeX$  that allow a “fake space” to be inserted into the PDF content stream, occurring between words and at the end of lines. This is done only at the final output, so it does not affect the high-quality positioning of words. Now when the textual content is extracted from the PDF, by Copy/Paste or other means, a space character is indeed included in the extracted content. This is a requirement to meet PDF/A archival standards.

The author will demonstrate examples of the use of this `\pdffakepace`, and the other new primitives that control when and where it is used (e.g., not needed in mathematical content) for producing PDF/A and “Tagged PDF” for both archivability and accessibility. Also to be shown is how a fake space allows extra material, such as the  $\LaTeX$  source of inline or displayed mathematics, to be included invisibly within the PDF. With a Select/Copy/Paste of the mathematical expression, this included source coding comes along with the pasted text.

### Dan Raies

*$\LaTeX$  in the classroom*

The  $\LaTeX$  skill-set required of a student or a researcher is vastly different than that required of a teacher. As a result, when one writes a test or homework assignment it often happens that the  $\LaTeX$

we know dictates the questions we can ask. In this talk we examine some  $\LaTeX$  techniques that will allow us to improve the materials that we use in the classroom. These techniques include ways to organize documents, ways to write solutions, and ways to create diagrams. We will examine some basic  $\LaTeX$  strategies as well as some particular packages that are of use for teachers.

### Will Robertson and Frank Mittelbach

*$\LaTeX$ 3 and `expl3` in 2014: Recent developments*

The `expl3` programming layer for  $\LaTeX$ 3 has stabilised and is now being used by many people “in the wild” and for many different package types. In this talk we’ll discuss the emerging popularity of `expl3` and some thoughts we have for rounding out its feature set. One area of recent development is case-changing in the Unicode era;  $\TeX$ 's `\uppercase` and `\lowercase` don’t fulfil our needs when case changing is language-dependent and in some cases no longer a one-to-one mapping. On the other hand, those primitives are used extensively for various tricks in  $\TeX$  programming, and one of `expl3`'s philosophies is to avoid ‘tricks’, so we can try to do something about that too.

### Etienne Tétreault-Pinard

*Plotly: Collaborative, interactive, and online plotting with  $\TeX$*

$\TeX$  was designed with the goal of allowing anyone to produce high-quality documents with minimal effort, and to provide a system compatible on all computers, now and in the future (A. Gaudeul, Do Open Source Developers Respond to Competition?: The ( $\LaTeX$ ) $\TeX$  Case Study, Social Science Research Network, 2006).

Plotly applies the same core principles to graphics. Plotly lets users collaboratively make and share interactive graphics online using Python, MATLAB, R, Excel data and  $\TeX$  (MathJax) for free. Additionally, Plotly allows users to easily embed and export graphics for publication, while backing up your graphics, data and revisions in the cloud. This tutorial outlines Plotly’s features and demonstrates how using Plotly creates unique workflows with emphasis on collaboration and reproducibility. More information is at [bit.ly/1vdF6Kp](http://bit.ly/1vdF6Kp).

### Alan Wetmore

*A quarter century of naïve use and abuse of  $\LaTeX$*

In this talk I will recount my introduction to and experiences with  $\LaTeX$ . Throughout this time installing and maintaining  $\TeX$  and friends has become ever so much easier while the capabilities have grown enormously. Along the way I learned about many subjects I hadn’t even known existed, and experimented with many facets of  $\TeX$ .

---

## Let's Learn L<sup>A</sup>T<sub>E</sub>X: A hack-to-learn ebook

Parthasarathy S

### Abstract

*Let's Learn L<sup>A</sup>T<sub>E</sub>X* is an innovative ebook which aims to speed up your L<sup>A</sup>T<sub>E</sub>X learning experience using the “hack-to-learn” approach. It offers many lessons which you can copy and modify freely, to create your own L<sup>A</sup>T<sub>E</sub>X document. Each lesson highlights some specific aspects of L<sup>A</sup>T<sub>E</sub>X.

### Description

Perhaps you have a basic knowledge of L<sup>A</sup>T<sub>E</sub>X, or have heard about it vaguely or seen it at work earlier. You are in a hurry to try your hand at L<sup>A</sup>T<sub>E</sub>X. You are overwhelmed by the complexity of L<sup>A</sup>T<sub>E</sub>X. Reading traditional books and tutorial material to understand L<sup>A</sup>T<sub>E</sub>X seems to be too cumbersome for you. You believe in D-I-Y, even for learning. If any or all of this applies to you, now you have a way out [6]. The ebook *Let's Learn L<sup>A</sup>T<sub>E</sub>X* gives you a total of 170+ pages in 25+ PDF documents (called lessons), each highlighting some aspect of L<sup>A</sup>T<sub>E</sub>X usage.

This book recommends (and encourages) hacking as an effective method of learning L<sup>A</sup>T<sub>E</sub>X. In fact, this book starts with two articles devoted to hacking as a means of learning. It gives 25 such lessons, each highlighting certain aspects of L<sup>A</sup>T<sub>E</sub>X. The L<sup>A</sup>T<sub>E</sub>X source of all the lessons is also given in easily locatable files (hyperlinked to the main text), so that you can hack them and experiment with them. True to the spirit of FLOSS, this book is distributed under a liberal license — Creative Commons Attribution-ShareAlike 4.0 [1]. The book can be freely downloaded from [6].

This book does not pretend to teach you L<sup>A</sup>T<sub>E</sub>X ab initio. Neither does it claim to answer all your questions about L<sup>A</sup>T<sub>E</sub>X. This is a D-I-Y approach to learning L<sup>A</sup>T<sub>E</sub>X. It aims to jump-start your L<sup>A</sup>T<sub>E</sub>X learning experience. The 25 lessons can be read in any order, and You can see the rendered version (pdf) as well as the source version (L<sup>A</sup>T<sub>E</sub>X) of each lesson, by clicking on the hyperlinks (shown in wine-red). This book is therefore best read on the computer screen. A printed version of this book will have all these texts printed and inserted at the appropriate places. Those using the printed version will still need the

book in a softcopy form (such as a DVD) to access the L<sup>A</sup>T<sub>E</sub>X sources. The softcopy version of this book makes best use of ebook format created by L<sup>A</sup>T<sub>E</sub>X.

### Availability

*Let's Learn L<sup>A</sup>T<sub>E</sub>X* was prepared and tested under a GNU/Linux system, and will work best there. In any case, you will have to ensure that your PDF reader, your text editor, and your web browser are properly installed, configured, and are working satisfactorily. You will need a live connection to the web to browse the web links in the book. To get the book with all its lessons, download the provided zip bundle, unzip, and follow the instructions in the `aboutme.txt` file.

Finally, this book is not expected to replace any of the traditional and popular books on L<sup>A</sup>T<sub>E</sub>X. It is a L<sup>A</sup>T<sub>E</sub>X practitioner's attempt to make L<sup>A</sup>T<sub>E</sub>X accessible to all who want to get proficient quickly in the use of L<sup>A</sup>T<sub>E</sub>X. You may refer to any of the three books [2, 3, 4], or use any of the many resources accessible on the web [5]. Indeed, keep these resources handy as you work on L<sup>A</sup>T<sub>E</sub>X using this book.

### References

- [1] Creative Commons Attribution-ShareAlike 4.0 Unported license, <https://creativecommons.org/licenses/by-sa/4.0/legalcode>
- [2] H. Kopka, P. Daly. Guide to L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>, fourth ed. Addison-Wesley, 2003.
- [3] L. Lamport. *L<sup>A</sup>T<sub>E</sub>X: A document preparation system*, 2nd ed. Addison-Wesley, 1994.
- [4] F. Mittelbach, M. Goossens, J. Braams, D. Carlisle, C. Rowley. *The L<sup>A</sup>T<sub>E</sub>X Companion*, 2nd ed. Addison-Wesley, 2004.
- [5] S. Parthasarathy. Dr. Partha's L<sup>A</sup>T<sub>E</sub>X starter, <http://www.freewebs.com/profpartha/startlatex.htm>
- [6] S. Parthasarathy. Let's Learn L<sup>A</sup>T<sub>E</sub>X. <http://www.freewebs.com/profpartha/teachlatex.htm>

◇ Parthasarathy S  
 Secunderabad, India  
 drpartha (at) gmail dot com  
<http://profpartha.webs.com/profpartha.htm>

---

**Book review: *Fifty Typefaces That Changed The World***

Jeffrey A. Barnett, jbb (at) notatt dot com

John L. Walters, *Design Museum Fifty Typefaces That Changed The World*, Alison Starling Publisher, 2013. ISBN 978-1-84091-629-4



This book is part of a Design Museum series including the titles “*Fifty x That Changed The World*”, where  $x$  is one of Bags, Bicycles, Cars, Chairs, Dresses, Shoes, Typefaces, or others. The book reviewed herein was originally published by Design House in association with Conran Octopus and John L. Walters, the Editor of *Eye* magazine, is the presumed author but is only credited by the phrase “*Text written by*”. N.B. There is no human name, e.g., an author’s name, on the spine of the book or on its cover and publishing credits, copyright claims, publishing dates, etc., are deferred to the last page of the book; in other words, a rather enigmatic structure and I may have some of the credits wrong.

The contents are simple to describe: The front matter consists of two copies of a title page separated by a page-sized slightly abstract picture of a type slug for an ampersand, a simple table of contents (no indentations, everything at the same level), then a one-page introduction accompanied by a picture of some characters. The main contents are fifty two-page articles, each describing a different typeface. The left-hand pages include descriptions with some history, inventor(s), motivation for introduction, and some example characters in the typeface. Right-hand pages show pictures, usually in color, of *applications* of the typeface, e.g., the pictures for Transport, aka the House Style for Britain, show actual road signs set in the Transport typeface. The end matter consists of four parts: a glossary defining some terminology used in the book, a subject index, picture credits, and other credits—publication dates, copyrights,

and additional information usually found near the front of a book.

The fifty typeface descriptions are arranged in chronological order. The first, circa 1455, is the Blackletter used to set the 42-line *Gutenberg Bible*. This type was based on the lettering used by monks to copy manuscripts. The accompanying pictures include a page of *Gutenberg* and a more modern version used for the naming sign on the Chicago Tribune building. The number of typefaces, by centuries, in this book are: 2 in 15th century, 3 in 16th, 1 in 17th, 4 in 18th, 10 in 19th, 27 in 20th, and 3 in 21st. The fiftieth typeface discussed is Ubuntu with a 2011 date so we are certainly in the computer age. Ubuntu aims to grow and provide a method to print languages based on roman as well as other alphabets. The accompanying picture shows a collection of characters that I do not recognize with markup comments, e.g., “*a bit narrow*”, meant to show the extensibility concept in action through design.

The book itself is printed on thick sheets of a very white paper and is set in DM Schulbuch, an invention of Design Museum. It is a rather natty looking sans serif face that is easy to read. The pictures are in the best traditions of the commercial artist showing off and give the book a hypermodern feel. In my mind this inexpensive book—less than \$16 at Amazon.com—qualifies as coffee table art. It’s very good looking.

My specific interest in typography is for author-centric generation of manuscripts for publication. Thus, I am specifically interested in mechanisms that individuals can use on computers. The inventions of eight of the fifty typefaces included in the book were motivated by considerations of digital usage. I believe the selector of that material was negligent by omitting the single most important contribution to digital typesetting to date. First let’s see which eight made the book and the reasons given for their inventions:

1. Zapf Dingbats (1978) — 1st for computers to make figure drawing easier
2. Beowolf (1989) — Exploits PostScript inconsistencies for random results
3. Scala (1991) — Old style serifs for a digital world
4. Meta (1991) — The anti-Helvetica, i.e., not bland
5. Blur (1991) — Desktop software could warp and twist typefaces into new fonts
6. Comic Sans (1994) — Inspired by Microsoft Bob inappropriately speaking New Times Roman
7. Georgia (1996) — A Microsoft-commissioned computer screen destination serif typeface
8. Ubuntu (2011) — Extensibility without limit for the computer age

I must review my own history<sup>1</sup> vis-à-vis typesetting in order to identify what I believe to be the glaring omission. I worked at the System Development Corporation (SDC) starting in the mid 1960s. This was the first time I was involved in preparing papers for external publication. Original text was handwritten since neither our computer terminals nor line printers had lower case or math characters. At some point after editing and hallway reviews, papers were ready to prepare for submission. I could now use the best, smartest typesetting engine I have ever had access to—Ethyl Erich! Lou (as we called her) had emigrated from Australia where she had been a court stenographer. And could she type too. One hand held the math ball for an IBM Selectric Typewriter, while the other typed. She would change balls with one hand as necessary and was able to sustain a rate of several words a minute through math-laden text.

Once Lou came into my office while preparing a lengthy document coauthored by several of us and said “On page 12 you have such and such but look at page 143.” There was a subtle inconsistency that took a meeting of the coauthors many hours to haggle out. As I said, this was the smartest typesetting system I ever used. Eventually we had terminals and line printers with lower case that were usable for initial paper preparation but were quite worthless for preparing “final” submissions.

The next stop for me was USC ISI during the late 1970s and early 1980s. ISI had a Xerox XGP printer with multiple typeface capability. ISI was also one of the beta test sites for Brian Reid’s Scribe typesetting system developed at Carnegie Mellon University as the substance of his PhD dissertation. Scribe and the XGP together provided a substantial typesetting capability that was quite exciting for its time. However there were three serious problems: 1) The XGP would only allow six different typefaces/fonts per page; 2) The available typefaces/fonts were wildly inconsistent, e.g., 12pt roman and italic had different baselines, and other problems of this sort, because computer science grad students sans commercial art backgrounds generated fonts du jour; 3) Scribe proclaimed but did not have decent mathematics capabilities, e.g., putting both a subscript and a superscript on the same character was a complicated process. I was writing a paper that ran into all of these problems and decided to fix things: 1) I “fixed” various faces in two sizes to be consistent;

---

<sup>1</sup> I’m aware that reviews are supposed to be about the object reviewed and not about the reviewer but this one is based on my opinion and I want to justify it. Or I’m declaring my interest as British MPs but not US Congress members do.

2) I combined several small special-purpose character sets into one to mitigate the XGP’s load limit; 3) I wrote a macro package to simplify typesetting simple mathematics. I tried to get ISI management to hire a part-time grad student with relevant typesetting experience to improve the Scribe/XGP combination into real usefulness. I failed in that attempt.

My final stop on the path to typesetting enlightenment in the digital world was the Northrop Research and Technology Center in the mid-1980s. Nothing to note for a while, then progress: first we worked our way onto the emerging Internet and second we received a T<sub>E</sub>X suite including T<sub>E</sub>X, L<sup>A</sup>T<sub>E</sub>X, and METAFONT! It was everything that Scribe proposed to be and much more. Math was insanely good, publication quality was almost guaranteed, the engines were fast given the hardware of the day, and the package dealt head-on with typeface problems.

Computer Modern was available in a plethora of sizes, boldness, slants, serif choices, etc. and all were consistent. It wasn’t long before the engine would generate a character set automatically when needed and absent. Nowadays many typefaces have been defined within the T<sub>E</sub>X suite and these definitions are distributed via systems such as MiK<sub>T</sub>E<sub>X</sub> and the CTAN repository system. A question came to mind at the time: has every font in the Computer Modern typeface that can be auto-generated been generated at least once somewhere? If the question is expanded to the totality of typefaces distributed through METAFONT, the answer is surely no.

I think the book’s omission must now be obvious: Computer Modern, by leveraging computers, was the world’s first truly complete typeface in any serious sense. Its appearance brought the vague promise of the computer in publishing into sharp focus. Here was software that could run on desk-sized computers and produce copy that would satisfy the most finicky publishing houses. And if the credit is not given to Computer Modern, consider METAFONT as the object of praise. If Ubuntu is to be honored for providing flexibility and growth, why not pin the gold star on something else which does the job better and did it earlier?

Though I’m sure the book’s omission is the result of ignorance, that isn’t an excuse when producing a title such as *Fifty Typefaces That Changed The World*. T<sub>E</sub>X and its companions have changed the entire publishing landscape and that should be acknowledged.

I must end what began as a short review with my excuse, a paraphrase of Mark Twain and Blaise Pascal: I am not knowledgeable enough to write you a short review, so I am providing a long one.

---

## 2015 T<sub>E</sub>X Users Group election

Kaja Christiansen  
for the Elections Committee

The positions of TUG President and nine members of the Board of Directors will be open as of the 2015 Annual Meeting, which will be held in July 2015 in Darmstadt, Germany.

The directors whose terms will expire in 2015: Barbara Beeton, Karl Berry, Susan DeMeritt, Michael Doob, Taco Hoekwater, Ross Moore, Cheryl Ponchin, Philip Taylor, and Boris Veytsman.

Continuing directors, with terms ending in 2017: Kaja Christiansen, Steve Grathwohl, Jim Hefferon, Klaus H\"oppner, Arthur Reutenauer, David Walden.

The election to choose the new President and Board members will be held in Spring of 2015. Nominations for these openings are now invited.

The Bylaws provide that "Any member may be nominated for election to the office of TUG President/ to the Board by submitting a nomination petition in accordance with the TUG Election Procedures. Election . . . shall be by written mail ballot of the entire membership, carried out in accordance with those same Procedures." The term of President is two years.

The name of any member may be placed in nomination for election to one of the open offices by submission of a petition, signed by two other members in good standing, to the TUG office at least two weeks (14 days) prior to the mailing of ballots. (A candidate's membership dues for 2015 will be expected to be paid by the nomination deadline.) The term of a member of the TUG Board is four years.

A nomination form follows this announcement; forms may also be obtained from the TUG office, or via <http://tug.org/election>.

Along with a nomination form, each candidate must supply a passport-size photograph, a short biography, and a statement of intent to be included with the ballot; the biography and statement of intent together may not exceed 400 words. The deadline for receipt of nomination forms and ballot information at the TUG office is **1 February 2015**. Forms may be submitted by FAX, or scanned and submitted by e-mail to [office@tug.org](mailto:office@tug.org).

Ballots will be mailed to all members within 30 days after the close of nominations. Marked ballots must be returned no more than six (6) weeks following the mailing; the exact dates will be noted on the ballots.

Ballots will be counted by a disinterested party not affiliated with the TUG organization. The results of the election should be available by early June, and will be announced in a future issue of *TUGboat* as well as through various T<sub>E</sub>X-related electronic lists.

## 2015 TUG Election — Nomination Form

Only TUG members whose dues have been paid for 2015 will be eligible to participate in the election. The signatures of two (2) members in good standing at the time they sign the nomination form are required in addition to that of the nominee. **Type or print** names clearly, using the name by which you are known to TUG. Names that cannot be identified from the TUG membership records will not be accepted as valid.

The undersigned TUG members propose the nomination of:

**Name of Nominee:** \_\_\_\_\_

Signature: \_\_\_\_\_

Date: \_\_\_\_\_

for the position of (check one):

**TUG President**

**Member of the TUG Board of Directors**

for a term beginning with the 2015 Annual Meeting, **July 2015**.

1. \_\_\_\_\_  
(please print)

\_\_\_\_\_ (signature) \_\_\_\_\_ (date)

2. \_\_\_\_\_  
(please print)

\_\_\_\_\_ (signature) \_\_\_\_\_ (date)

Return this nomination form to the TUG office (forms submitted by FAX or scanned and submitted by e-mail will be accepted). Nomination forms and all required supplementary material (photograph, biography and personal statement for inclusion on the ballot) must be received in the TUG office no later than **1 February 2015**.<sup>1</sup> It is the responsibility of the candidate to ensure that this deadline is met. Under no circumstances will incomplete applications be accepted.

- nomination form
- photograph
- biography/personal statement

T<sub>E</sub>X Users Group **FAX:** +1 815 301-3568  
**Nominations for 2015 Election**  
P. O. Box 2311  
Portland, OR 97208-2311  
U.S.A.

---

<sup>1</sup> Supplementary material may be sent separately from the form, and supporting signatures need not all appear on the same form.

## TUG Institutional Members

American Mathematical Society,  
*Providence, Rhode Island*

Aware Software, Inc., *Midland Park, New Jersey*

Center for Computing Sciences, *Bowie, Maryland*

CSTUG, *Praha, Czech Republic*

Fermilab, *Batavia, Illinois*

Google, *San Francisco, California*

IBM Corporation, T J Watson Research Center,  
*Yorktown, New York*

Institute for Defense Analyses, Center for  
Communications Research, *Princeton, New Jersey*

Marquette University, Department of Mathematics,  
Statistics and Computer Science,  
*Milwaukee, Wisconsin*

Masaryk University, Faculty of Informatics,  
*Brno, Czech Republic*

MOSEK ApS, *Copenhagen, Denmark*

New York University, Academic Computing Facility,  
*New York, New York*

Springer-Verlag Heidelberg, *Heidelberg, Germany*

StackExchange, *New York City, New York*

Stanford University, Computer Science Department,  
*Stanford, California*

Stockholm University, Department of Mathematics,  
*Stockholm, Sweden*

University College, Cork, Computer Centre,  
*Cork, Ireland*

Université Laval, *Ste-Foy, Québec, Canada*

University of Ontario, Institute of Technology,  
*Oshawa, Ontario, Canada*

University of Oslo, Institute of Informatics,  
*Blindern, Oslo, Norway*

University of Wisconsin, Biostatistics &  
Medical Informatics, *Madison, Wisconsin*

VT<sub>E</sub>X UAB, *Vilnius, Lithuania*

## 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<sub>A</sub>T<sub>E</sub>X, we are your microtypography specialists. We take special care to typeset mathematics well.

Not that picky? We also handle most of your typical T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X typesetting needs.

We have been typesetting in the commercial and academic worlds since 1979.

Our team includes Masters-level computer scientists, journeyman typographers, graphic designers, letterform/font designers, artists, and a co-author of a T<sub>E</sub>X book.

### Latchman, David

4113 Planz Road Apt. C  
Bakersfield, CA 93309-5935  
+1 518-951-8786

Email: [david.latchman \(at\) texnical-designs.com](mailto:david.latchman@texnical-designs.com)

[texnical-designs.com](http://www.texnical-designs.com)

Web: <http://www.texnical-designs.com>

L<sup>A</sup>T<sub>E</sub>X consultant specializing in: the typesetting of books, manuscripts, articles, Word document conversions as well as creating the customized packages to meet your needs.

Call or email to discuss your project or visit my website for further details.



**Peter, Steve**

+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  $\text{\TeX}$ , I have typeset books for Pragmatic Programmers, Oxford University Press, Routledge, and Kluwer, among others, and have helped numerous authors turn rough manuscripts, some with dozens of languages, into beautiful camera-ready copy. In addition, I've helped publishers write, maintain, and streamline  $\text{\TeX}$ -based publishing systems. I have an MA in Linguistics from Harvard University and live in the New York metro area.

**Sievers, Martin**

Klaus-Kordel-Str. 8, 54296 Trier, Germany

+49 651 4936567-0

Email: [info \(at\) schoenerpublizieren.com](mailto:info@schoenerpublizieren.com)Web: <http://www.schoenerpublizieren.com>

As a mathematician with ten years of typesetting experience I offer  $\text{\TeX}$  and  $\text{\LaTeX}$  services and consulting for the whole academic sector (individuals, universities, publishers) and everybody looking for a high-quality output of his documents. From setting up entire book projects to last-minute help, from creating individual templates, packages and citation styles ( $\text{\BIBTeX}$ ,  $\text{\biblatex}$ ) to typesetting your math, tables or graphics — just contact me with information on your project.

**Sofka, Michael**

8 Providence St.

Albany, NY 12203

+1 518 331-3457

Email: [michael.sofka \(at\) gmail.com](mailto:michael.sofka@gmail.com)

Skilled, personalized  $\text{\TeX}$  and  $\text{\LaTeX}$  consulting and programming services.

I offer over 25 years of experience in programming, macro writing, and typesetting books, articles,

**Sofka, Michael (cont'd)**

newsletters, and theses in  $\text{\TeX}$  and  $\text{\LaTeX}$ : Automated document conversion; Programming in Perl, C, C++ and other languages; Writing and customizing macro packages in  $\text{\TeX}$  or  $\text{\LaTeX}$ ; Generating custom output in PDF, HTML and XML; Data format conversion; Databases.

If you have a specialized  $\text{\TeX}$  or  $\text{\LaTeX}$  need, or if you are looking for the solution to your typographic problems, contact me. I will be happy to discuss your project.

**Veytsman, Boris**

46871 Antioch Pl.

Sterling, VA 20164

+1 703 915-2406

Email: [borisv \(at\) lk.net](mailto:borisv@lk.net)Web: <http://www.borisv.lk.net>

$\text{\TeX}$  and  $\text{\LaTeX}$  consulting, training and seminars. Integration with databases, automated document preparation, custom  $\text{\LaTeX}$  packages, conversions and much more. I have about eighteen years of experience in  $\text{\TeX}$  and three decades of experience in teaching & training. I have authored several packages on CTAN, published papers in  $\text{\TeX}$  related journals, and conducted several workshops on  $\text{\TeX}$  and related subjects.

**Young, Lee A.**

127 Kingfisher Lane

Mills River, NC 28759

+1 828 435-0525

Email: [leeayoung \(at\) morrisbb.net](mailto:leeayoung@morrisbb.net)Web: <http://www.thesiseditor.net>

Copyediting your  $\text{\.tex}$  manuscript for readability and mathematical style by a Harvard Ph.D. Your  $\text{\.tex}$  file won't compile? Send it to me for repair. Experience: edited hundreds of ESL journal articles, economics and physics textbooks, scholarly monographs,  $\text{\LaTeX}$  manuscripts for the Physical Review; career as professional, published physicist.

**TUG2015**  
**Darmstadt, Germany**  
**July 20–22, 2015**  
[\*\*http://tug.org/tug2015\*\*](http://tug.org/tug2015)

## Calendar

### 2014

- Oct 3 *TUGboat* **35**:3, submission deadline.
- Oct 3–5 Oak Knoll Fest XVIII, and Fine Press Book Association annual meeting, New Castle, Delaware. [www.oakknoll.com/fest](http://www.oakknoll.com/fest)
- Oct 31–Nov 4 ASIS&T 2014, 77<sup>th</sup> Annual Meeting, “Connecting Collections, Cultures, and Communities”, American Society for Information Science and Technology, Seattle, Washington. [www.asis.org/asist2014](http://www.asis.org/asist2014)
- Nov 8–9 The Twelfth International Conference on Books, Publishing, and Libraries, “Disruptive Technologies and the Evolution of Book Publishing and Library Development”, Simmons College, Boston, Massachusetts. [booksandpublishing.com/the-conference](http://booksandpublishing.com/the-conference)
- Nov 13–14 The Printing Historical Society’s 50<sup>th</sup> Anniversary, “Landmarks in Printing: from origins to the digital age”, St Bride Institute, London, UK. [printinghistoricalsociety.org.uk/forthcoming\\_phs\\_events/#144](http://printinghistoricalsociety.org.uk/forthcoming_phs_events/#144)
- Nov 14 TYPO Day, “Business Typography Talks”, München, Germany. [typotalks.com/day/muenchen-2014](http://typotalks.com/day/muenchen-2014)
- Nov 28–29 5<sup>th</sup> Meeting of Typography, “Ubiquitous”, Escola Superior de Tecnologia–IPCA, Barcelos, Portugal. [www.atypi.org/events/5th-meeting-of-typography](http://www.atypi.org/events/5th-meeting-of-typography)

### 2015

- Feb 1 **TUG election:** nominations due. [tug.org/election](http://tug.org/election)
- Mar 7–9 Typography Day 2015, “Typography, Sensitivity and Fineness”, Industrial Design Center, Indian Institute of Technology, Bombay, India. [www.typoday.in](http://www.typoday.in)

- Mar 9 *TUGboat* **36**:1, submission deadline (regular issue)
- Mar 19–21 “Publish or Perish? Scientific periodicals from 1665 to the present”. The Royal Society, London, UK. [royalsociety.org/events/](http://royalsociety.org/events/)
- Apr DANTE Frühjahrstagung and 52<sup>nd</sup> meeting, Stralsund, Germany. [www.dante.de/events.html](http://www.dante.de/events.html)
- Apr 29–May 3 BachoT<sub>E</sub>X 2015: 23<sup>rd</sup> BachoT<sub>E</sub>X Conference, Bachotek, Poland. [www.gust.org.pl/bachotex](http://www.gust.org.pl/bachotex)
- Apr 30–May 1 TYPO San Francisco, Yerba Buena Center for the Arts, San Francisco, California. [typotalks.com/sanfrancisco](http://typotalks.com/sanfrancisco)
- May 21–23 TYPO Berlin 2015, “Character”, Berlin, Germany. [typotalks.com/berlin](http://typotalks.com/berlin)
- Jun 29–Jul 3 Digital Humanities 2015, Alliance of Digital Humanities Organizations, “Global Digital Humanities”, Sydney, Australia. [dh2015.org](http://dh2015.org)

---

### TUG 2015

#### Darmstadt, Germany.

- Jul 20–22 The 36<sup>th</sup> annual meeting of the T<sub>E</sub>X Users Group. [tug.org/tug2015](http://tug.org/tug2015)

- 
- Aug 9–13 SIGGRAPH 2015, “Xroads of Discovery”, Los Angeles, California. [s2015.siggraph.org](http://s2015.siggraph.org)
- Aug 24–28 SHARP 2015, Society for the History of Authorship, Reading & Publishing, Jinan, Shandong Province, China, [www.sharpweb.org](http://www.sharpweb.org)
- Oct 19–20 The Thirteenth International Conference on Books, Publishing, and Libraries, University of British Columbia, Vancouver, Canada. [booksandpublishing.com/the-conference-2015](http://booksandpublishing.com/the-conference-2015)

*Status as of 15 September 2014*

For additional information on TUG-sponsored events listed here, contact the TUG office (+1 503 223-9994, fax: +1 815 301-3568. e-mail: [office@tug.org](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).

## Introductory

- 135 *Michael Sharpe* / Recent additions to T<sub>E</sub>X's font repertoire
  - Garamonds, Baskervilles, typewriter fonts, sans serif, and more
- 139 *Jim Hefferon* and *Lon Mitchell* / Experiences converting from PDF-only to paper
  - offering an online textbook in print for the first time
- 142 *Joseph Hogg* / Texinfo visits a garden
  - producing an annotated, indexed, plant list for the Huntington's Herb Garden

## Intermediate

- 195 *Leyla Akhmadeeva* and *Boris Veytsman* / Typography and readability: An experiment with post-stroke patients
  - serif vs. sans serif for readers with cognitive impairments from stroke
- 209 *Robert A. Beezer* / MathBook XML
  - writing technical documents with many possible output formats and online integration
- 168 *Julian Gilbey* / Creating (mathematical) jigsaw puzzles using T<sub>E</sub>X and friends
  - educational puzzles creating using T<sub>E</sub>X, Python, YAML, et al.
- 198 *SK Venkatesan* and *CV Rajagopal* / T<sub>E</sub>X and copyediting
  - copyediting markup to improve consistency and communication

## Intermediate Plus

- 192 *David Allen* / Dynamic documents
  - using R's `tikzDevice` to generate graphical output in L<sup>A</sup>T<sub>E</sub>X
- 173 *Pavneet Arora* / SUTRA — A workflow for documenting signals
  - using YAML and ConT<sub>E</sub>Xt tables for generalized signal documentation
- 145 *Richard Koch* / MacT<sub>E</sub>X design philosophy vs. TeXShop design philosophy
  - Global vs. LocalT<sub>E</sub>X PrefPane for the Mac, and Apple histories
- 152 *Adam Maxwell* / T<sub>E</sub>X Live Utility: A slightly-shiny Mac interface for T<sub>E</sub>X Live Manager (`tlmgr`)
  - a Mac OS X graphical interface for `tlmgr`
- 179 *Andrew Mertz*, *William Slough* and *Nancy Van Cleave* / Typesetting figures for computer science
  - practical packages for drawing stacks, byte fields, trees, automata, and more

## Advanced

- 212 *William Hammond* / Can L<sup>A</sup>T<sub>E</sub>X profiles be rendered adequately with static CSS?
  - using pure CSS to handle math from L<sup>A</sup>T<sub>E</sub>X profiles in, e.g., GELLMU
- 157 *Doug McKenna* / On tracing the `trip` test with JSBox
  - accurate and complete tracing as part of developing a new T<sub>E</sub>X interpreter
- 205 *Keiichiro Shikano* / `xml2tex`: An easy way to define XML-to-L<sup>A</sup>T<sub>E</sub>X converters
  - a Scheme program to use L<sup>A</sup>T<sub>E</sub>X as an effective XML presentation layer
- 202 *Boris Veytsman* / An output routine for an illustrated book: Making the *FAO Statistical Yearbook*
  - when illustrations are primary and text is secondary

## Reports and notices

- 126 TUG 2014 conference information
- 127 TUG 2014 conference program
- 128 TUG 2014 photos
- 130 *David Latchman* / TUG 2014 in Portland
- 134 *Tracy Kidder* / Visiting TUG 2014
- 219 TUG 2014 abstracts (Bazargan, Berry, Crossland, Cunningham, de Souza, Doob, Farmer, McKenna, Mittelbach, Moore, Raies, Robertson, Tétreault, Wetmore)
- 222 *S Parthasarathy* / Let's Learn L<sup>A</sup>T<sub>E</sub>X: A hack-to-learn ebook
- 223 *Jeffrey Barnett* / Book review: *Fifty Typefaces That Changed The World*, by John Walters
  - review of this art book, with personal commentary on the omission of Computer Modern
- 225 *Kaja Christiansen* / TUG 2015 election
- 226 Institutional members
- 226 T<sub>E</sub>X consulting and production services
- 228 Calendar