TUGboat

Volume 33, Number 2 / 2012
TUG 2012 Conference Proceedings

TEX Users Group

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

Memberships and Subscriptions

2012 dues for individual members are as follows:

- Ordinary members: $95.
- Students/Seniors: $65.

The discounted rate of $65 is also available to citizens of countries with modest economies, as detailed on our web site.

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

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

Institutional Membership

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

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 trademarks which commonly appear in *TUGboat* should not be considered complete.

TEX 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: August 2012]

Printed in U.S.A.

Addresses

TEX 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`

Submissions to *TUGboat*, letters to the Editor:
`TUGboat@tug.org`

Technical support for TEX users:
`support@tug.org`

Contact the Board of Directors:
`board@tug.org`

2012 Conference Proceedings

TEX Users Group
Thirty-third Annual Meeting
Boston, Massachusetts
July 16–18, 2012

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP
TUGBOAT EDITOR BARBARA BEETON
PROCEEDINGS EDITOR KARL BERRY

VOLUME 33, NUMBER 2 • 2012
PORTLAND • OREGON • U.S.A.

TUG 2012

Boston, Massachusetts USA

July 16–18, 2012

Sponsors

TEX Users Group ▪ DANTE e.V.
CSLI Publications ▪ O'Reilly ▪ River Valley Technologies
with special assistance from individual contributors. *Thanks to all!*

Conference committee

Karl Berry ▪ Robin Laakso ▪ Steve Peter ▪ Dave Walden

Bursary committee

Steve Peter, chair ▪ Jana Chlebikova ▪ Bogusław Jackowski ▪ Alan Wetmore

Workshop leaders

Sue DeMeritt ▪ Cheryl Ponchin

Participants

Robert Alps, Evanston, IL
Pavneet Arora, Bolton, Canada
Rohan Attele, Chicago State University
Michael Barr, McGill University
Nelson Beebe, University of Utah
Barbara Beeton, American Mathematical Society
Karl Berry, TEX Users Group
Bill Cheswick
Bart Childs, College Station, TX
Daniel Comenetz, Belmont, MA
Julie Conroy, IDA/CCS
Jack Creilson, American Meteorological Society
J. Michael Dean, University of Utah
Sue DeMeritt, Center for Communications Research, La Jolla, CA
Michael Doob, University of Manitoba
Julie Doring, Project Euclid–Duke Univ. Press
Sadeq Elbaneh, Lackawanna, NY
Peter Flynn, University College Cork
Federico Garcia, Alia Musica Pittsburgh
Steve Grathwohl, Duke University Press
Matthew Hacker, American Meteorological Society
Michele Hake, American Physical Society
Jim Hefferon, Saint Michael's College
Troy Henderson, University of Mobile
Amy Hendrickson, TEXnology Inc.
Klaus Höppner, DANTE e.V.
Ned Hummel, Indiana University–Purdue University Indianapolis
John Kitzmiller, Norwich, VT
Richard Koch, University of Oregon
Robin Laakso, TEX Users Group
David Latchman, TEXnical Designs
Philippe Baril Lecavalier, Concordia University
Richard Leigh, St Albans, UK
Sherif Mansour, Cairo University, Egypt
Wendy McKay, California Institute of Technology
Frank Mittelbach, LATEX3 Project
Robert Moody, Victoria, BC
Bob Neveln, Widener University
Brian Papa, American Meteorological Society
Steve Peter, TEX Users Group
Cheryl Ponchin, Center for Communications Research, Princeton, NJ
Norbert Preining, JAIST
Peter Pupalaikis, Ramsey, NJ
Thomas Ratajczak, German Army
Will Robertson, University of Adelaide
Herbert Schulz, Naperville, IL
Heidi Sestrich, Carnegie-Mellon University
Charles Shooshan, Newington, CT
William Slough, Eastern Illinois University
Michael Sofka, RPI
David Tellet, Alexandria, VA
Christina Thiele, Carleton Production Centre
Didier Verna, EPITA
Boris Veytsman, George Mason University
Bruno Voisin, CNRS and University of Grenoble
Herbert Voß, DANTE e.V.
David Walden, E. Sandwich, MA
Alan Wetmore, US Army Research Lab
Patrick Weyer, Medfield, MA
Christine Wujick, Alexandria, VA

TUG 2012 — program and information

Sunday, July 15, 4–6 pm: *opening reception and registration.*
Monday, July 16: *concurrent LaTeX workshop*, Sue DeMeritt & Cheryl Ponchin.

Monday July 16

8:00 am	*registration*	
9:00 am	Steve Peter, TUG	*Opening*
9:15 am	Amy Hendrickson	*The wonders of* `\csname`
9:55 am	*break*	
10:15 am	Frank Mittelbach	*E-TeX: Guidelines for future TeX extensions, revisited*
11:15 am	Steve Peter	*Metafont as a design tool*
11:35 am	Will Robertson	*Lineage and progeny of fontspec and unicode-math*
12:15 pm	*lunch*	
1:20 pm	*group photo*	
1:30 pm	Sherif Mansour & Hossam Fahmy	*Experience with Arabic and LuaTeX*
2:10 pm	Norbert Preining	*Typesetting with Kanji — Japanese typography*
2:50 pm	*break*	
3:05 pm	Federico Garcia	*TeX and music*
3:45 pm	David Walden	*My Boston: Some printing and publishing history*
4:25 pm	q&a	
(after	Herbert Schulz	*Workshop: Introduction to TeXShop*
hours)	Steve Peter, et al.	*Workshop: Installing LuaTeX*

Tuesday July 17

9:00 am	Troy Henderson	*User-friendly web utilities for generating LaTeX output and MetaPost graphics*
9:40 am	Richard Koch	*The MacTeX install package*
10:00 am	Bill Cheswick	*An iTeX update*
10:20 am	*break*	
10:40 am	Peter Flynn	*A university thesis class: Automation and its pitfalls*
11:20 am	David Latchman	*Preparing your thesis in LaTeX*
noon	*lunch*	
1:00 pm	Boris Veytsman	*TeX and friends on a Pad*
1:40 pm	Bart Childs	*LaTeX source from word processors*
2:20 pm	*break*	
2:40 pm	Federico Garcia	*Documentation in TeXnicolor*
3:20 pm	Jim Hefferon & Michael Doob	*Reaching for the stars with Asymptote*
4:00 pm	David Walden, moderator	*Roundtable discussion: TeX consulting* Flynn, Hendrickson, Latchman, Peter, Thiele, Veytsman
6–10 pm	banquet	at Oceanaire (`theoceanaire.com/Locations/Boston`)

Wednesday July 18

9:00 am	Pavneet Arora	*Sleep de(p)rived typesetting*
9:40 am	Bob Neveln & Bob Alps	*Adapting ProofCheck to the author's needs*
10:20 am	*break*	
10:40 am	Christina Thiele	*Almost 30 years of using TeX*
11:20 am	Will Robertson & Frank Mittelbach	*LaTeX3: From local to global — a brief history and recent developments*
noon	*lunch*	
1:00 pm	Boris Veytsman & Leyla Akhmadeeva	*Towards evidence-based typography: First results*
1:40 pm	Norbert Preining	*TeX Live 2012: Recent developments*
2:20 pm	*break*	
2:40 pm	Didier Verna	*Star TeX: The Next Generation*
3:20 pm	TUG meeting; q&a	
≈ 4:00 pm	*end*	

TUG 2012: A first-time attendee

David S. Latchman

On July 16–18, 2012, I attended the 33rd annual meeting of the TEX Users Group in Boston, MA. It was not only my first time attending such an event but my first time presenting. Though I did not know what to expect, I found my experience to be both enjoyable and educational. The 2012 conference was held at the Omni Parker House located in the heart of downtown Boston at the corner of School and Tremont Streets. This hotel has the distinction of being the longest continuously operating hotel in the US and fits right in with the rich history of downtown Boston. The hotel lies along the Freedom Trail, a 2.5 mile mostly-red-brick path that connects Boston's most significant historic sites starting from Boston Commons and ending at the *USS Constitution*.

My trip to the conference was a red-eye flight from Los Angeles. By the time I arrived in Boston, I was as well-rested as anyone could be after such a flight. Getting to the hotel from Logan International was made easy by Dave Walden's clear travel instructions on the TUG website. I took the Silver Line bus to South Station. From there, I took the Red Line subway (the T) to the Park Street station which was a short walk to the hotel.

(a) Silver Line bus stop

(b) Train at South Station

Figure 1: From Logan International airport to downtown Boston

Figure 2: Markers at the King's Chapel Burying Ground

Downtown Boston and the Omni Parker House

School Street is so named because it was the site of the first public school in the United States, the Boston Latin School. The Boston Latin School is also the oldest existing school in the United States and has moved several times from its original location. About half-a-mile away is the Boston Common, another record for the city of Boston as it is the oldest city park in the US. A first time visitor to the city can't help but be in awe of the history that surrounds them.

Just across from the Omni Parker House and also on the corner of School and Tremont Streets is King's Chapel. In addition to being a historic landmark that lies along the Freedom Trail, King's Chapel is also a place of worship. The Chapel is open to visitors except during times of worship. As it was Sunday, I didn't have the opportunity to see inside the chapel but did manage to visit the nearby cemetery — the King's Chapel Burying Ground.

Though the cemetery shares a name with the nearby church, it is in fact the property of the city and was founded in 1630, making it Boston's first cemetery. It remained Boston's only burial site for thirty years. The church itself didn't come into existence until some time later, in 1689, and was built on the burial grounds as no one in Boston wanted to sell land to a non-Puritan church. There are a few notable historic figures who can call the King's Chapel Burying Ground their final resting place. They include John Winthrop, Massachusetts's first governor and William Dawes, Paul Revere's companion on his famous ride. Though regular burials ceased in 1896, the occasional burial still takes place.

Not far from the King's Chapel Burying Ground is the Granary Burying Ground, another tourist attraction located on Tremont Street along the Freedom Trail. This cemetery is the final resting place of three signers of the Declaration of Independence and all five victims of the Boston Massacre in 1770. Also prominent in the cemetery is an obelisk erected in 1827 to the parents and relatives of Benjamin Franklin. The obelisk was constructed from granite obtained from the Bunker Hill Monument quarry to replace the original family gravestones which had deteriorated over time.

One of the notable men buried at the Granary Burial Ground is John Hancock, most remembered not because he served as President of the Second Continental Congress or that he was the first and

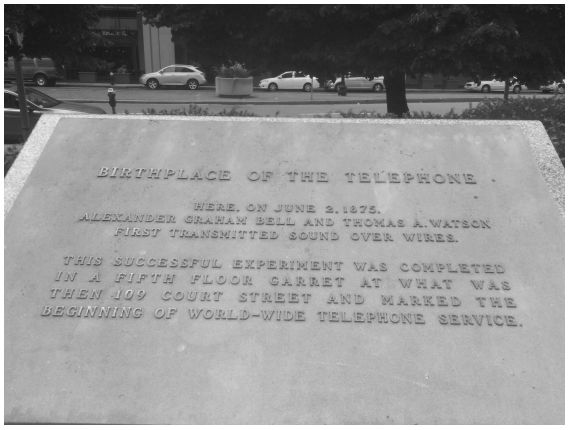
(a) The John Hancock Memorial

(b) The Paul Revere Memorial

(c) The Samuel Adams Marker

Figure 3: Markers at the Granary Burying Ground

(a) Boston City Hall



(b) The first telephone marker in front of the JFK building.

Figure 4: Government sites in Boston

third Governor of Massachusetts but rather for the prominence of his large and stylish signature on the Declaration of Independence. It is believed that Hancock made his signature so large as an act of defiance to King George III. This story couldn't be further from the truth.

As the President of the Second Continental Congress, Hancock was the first to sign the now historical document. At the time, the Declaration of Independence wasn't a formal declaration of independence from Britain but rather was intended to be copied and distributed among the colonies to explain the need to declare independence. As a result, Hancock thought he would be the nearly the only person to sign the document (along with the Secretary of the Congress, Charles Thompson), and so made his signature large. As the Declaration of Independence gained support, other delegates stated adding their names over a period of weeks and months.

For the beer drinkers reading this, it may be surprising to note that the face featured on a Sam Adams beer isn't actually Sam Adams but Paul Revere. It is said that Samuel Adams was not a good-looking fellow while Paul Revere, on the other hand, was considered a handsome chap. Both men are buried at the Granary Burying Ground.

Downtown Boston doesn't just feature historical sites and landmarks. It is also the center of the city's government, the section of which is, appropriately, named the Government Center. Not far from the hotel is both the City Hall Plaza and the John F. Kennedy Federal Building. In front of the JFK Building on Cambridge Street sits a plaque commemorating the first sound transmission over wire, an event that led to the practical invention of the telephone.

The event the plaque commemorates isn't the one we are all familiar with that took place on March 10th, 1876, when Alexander Graham Bell shouted the words, "Mr. Watson, come here, I want to see you".

In 1875, Bell under the advice of another scientist, Joseph Henry, stated working on an electrical multi-reed device to transmit the human voice over wires. Bell didn't have the necessary skill or knowledge to work on such a device, but a chance meeting with Thomas Watson changed all that.

The principle behind the reed design operated on the same principle as the human ear. The pinna collects sound and concentrates it as it travels down the auditory canal. Vibrations on the tympanic membrane then transmit energy to the ossicles or ear bones which is then converted into electrical impulses by the cochlea to be interpreted by the brain. The ear bones are a system of levers that amplify the force on the cochlea. This allows the tympanic membrane to be relatively small. Bell reasoned that if he had a larger membrane, he would be able to collect more sound and move a steel reed placed in a magnetic field. This would turn sound energy into an electrical current as the reed moves through the magnetic field.

On June 2, 1875, while troubleshooting the multi-reed device, Watson plucked one of the reeds to check the tension believing it to be too tight. Bell, who happened to be listening at the right time in the next room, heard the metallic twang and the first working model of the telephone was born.

The Conference

The Attendees and Presenters There were 61 participants and 23 presenters (of which I was one) listed in the TUG program, with participants coming from Australia, Canada, UK, Germany and Egypt. Attending the conference was a bit of a surreal experience as I saw so many people who were as passionate as I was about TEX and its future development. Where most of the conferences I have attended in the past tend to be specialized events geared toward one topic and where everyone — more or less — is in the same field, I found this to be quite different. Instead I found people with a wide array of specialties and talents.

Day One The conference started with an opening from the TUG president, Steve Peter, which was then followed by a presentation by a LATEX consultant, Amy Hendrickson of TEXnology Inc. Amy's talk focused on the use of `\csname` to create LATEX macros and dynamically generate a series of definitions. Amy demonstrated its use to redefine the footnotes command to produce endnotes and to dynamically create online reports. This is something I definitely have to look into and learn about.

After a break, Frank Mittelbach talked about LATEX3 and the direction of the TEX typesetting engine. When TEX was started over two decades ago, computing power was far more limited compared to what is available today and, as a result, so too were some of the algorithms. Frank compared the limitations of TEX's algorithms to LATEX3 and how they were overcome, to what degree and what still needs to be done. Given the increase in computing power and the work that has been done in recent years, the present system could be a viable one and remain so for some time.

Next was a presentation on font design by Steve Peter. Steve is a linguist and font designer and showed how font design can be done using the METAFONT package.

Will Robertson next came on to show his work on the `fontspec` and `unicode-math` packages. Will's talk focused on his experiences in developing these packages while at the same time learning how to program — one of the consequences of which was contributing code to the LATEX3 project.

Will's talk was followed by lunch and a group photo of the conference attendees.

The next group of talks focused on using the TEX engine to typeset foreign languages. Unfortunately, the talk on using LATEX to typeset Mayan hieroglyphics by Bruno Delprat and Stepan Orevkov had to be cancelled. It was a little disappointing as I have always found it interesting how the TEX typesetting engine can be used in so many ways, from typesetting chess boards to creating Sudoku puzzles. Fortunately, the next talk by Sherif Mansour on typesetting Arabic proved just as interesting.

Sherif is a graduate student from Cairo University whose research is focused on improving Arabic typesetting using the AlQalam font in LuaTeX. One of the problems Sherif faces is the typesetting of right-to-left fonts in LuaTEX. Each written line on a page in the Arabic language is about the same length and this makes line breaking difficult. This makes for a difficult problem to solve as the various shapes can also change in subtle ways for the same characters. Though I don't envy Sherif I am interested in his future work. Maybe I can find a way to use LATEX to help me learn Arabic.

The next presentation by Norbert Preining focused on the problems of typesetting in Japanese. Japanese typesetting differs from English in that there are four different writing systems: Kanji, Hiragana, Katakana and Roman letters, the difficulties of which are further compounded as Japanese employs both vertical and horizontal typesetting styles.

Federico Garcia next talked about music typesetting. As a developer working on writing a professional music typesetting system in TEX, Federico

highlighted some of the problems experienced with typesetting music. Though I am not a musician, I could appreciate some of the problems he faced. One of these problems is getting the TEX engine to automatically (and correctly) place the beams when creating a sheet of music. Federico compared his system with other music typesetting systems. While many problems need to be solved before he has a fully functional system, he believes they can be solved.

Day Two The first talk of the day by Troy Henderson focused on some of the web-based utilities he had developed to generate and plot functions in METAPOST. I haven't paid too much attention in the past where METAPOST or PSTricks is concerned, choosing to favor Ti*k*Z instead. I may have to reevaluate that position as the online utilities may help me learn and pick up the code quicker.

The next talk by Richard Koch focused on installing TEX on the Mac using the MacTEX install package. The key feature of this package, like everything Mac, is its ease of use and LATEX can be installed with a single click of the button. Richard demonstrated a full TEX installation during the course of his talk; we were all assured that an actual installation took place and no trickery was involved.

If you are an iPad owner then Bill Cheswick's talk on iTEX should be of interest. The iTEX app is a LATEX reader for the iPad and shouldn't in any way be confused with Donald Knuth's proposed XML-based successor to TEX. When typesetting a book, or a page or any document, the final page size is generally known down to the nanometer. TEX then uses some clever algorithms to optimize the presentation for a high standard of quality output. Unfortunately, this poses some problems for ebook readers. As there are varying screen sizes as well as the different ways one can hold a reader, it would be difficult and time-consuming to run TEX over a document every time a reader shifts position.

The iTEX application solves this by using TEX to create precomputed images for portrait and landscape layouts in both standard and large type versions. These images are then stored in a container file that the app can read. Besides being a reader, the app also converts text into LATEX typeset output. Bill demonstrated this by importing and converting text from Project Gutenberg and `arXiv.org`. The app is free on Apple's app store for anyone interested to try.

LATEX and Thesis Talks The next two presentations after the break focused on theses and dissertations. While my talk dealt with the various packages students can use to make their lives easier, Peter Flynn's talk dealt with the creation of a LATEX class file for the University College Cork (UCC). Given the number of thesis class files present on the CTAN server, and that we can assume there are far more unofficial class files elsewhere, the question we must ask is, "do we really need another thesis class file?" According to Peter Flynn, we do.

Generally speaking, a thesis class file is supposed to meet the formatting requirements of a particular university by ensuring page dimensions and margins are set correctly. But this is not the only requirement. Students are also required to enter the formal names of their departments and the colleges they belong to. Given that a university might have a complex departmental infrastructure and even stranger naming standards, the best way to ensure that students enter this information correctly is to enter them as options to the `\documentclass` command rather than allow authors to enter this information themselves. The Cork class file was tested on January 2010 over the course of 18 months and so far the response has been favorable — so favorable, in fact, that other institutions are looking to adopt or base their class file on Cork's.

Like most university class files in my experience, the UCC class file was designed to be minimalist in nature but at the same time meet the needs of as many students as possible. Any package that is added must be done carefully so as not to break any other existing packages and must be done to meet the needs of as many students as possible. Unfortunately, with the number of disciplines and in some cases, cross-disciplines, this means that all the needs of every student cannot be built into a single class file and students will need to add packages as they see fit. This was the subject of my talk.

Typically, when a student looks for a LATEX consultant they are under a lot of stress. Generally, any attempts to compile a document freezes and ends up with hundreds of errors. Part of my job as a consultant isn't to just fix these problems and get the document compiling again but to offer solutions and make lives easier. Once a project is concluded, students still need to make edits before the final submission.

LATEX has a multitude of packages and, at a quick glance, it almost seems as if you can do anything. By knowing a student's discipline or what their thesis is about, a consultant can often give advice on the best package to optimize the writing process. Students in engineering or the sciences, for example, may find it easy to take advantage of the `siunitx` package which will allows them to enter

mathematical units easier rather than having to enter math mode then enter a confusing list of symbols. Science students who deal with chemical equations may also take advantage of the `mhchem` package for the same reason. I talked about some past projects and how proper package use helped my clients.

Post-Lunch Talks The next talk by Boris Veytsman was on using TEX on the iPad: "TEX and friends on a Pad". Though I can't imagine actually typing LATEX code on a flat screen I am nonetheless excited at the possibility. Maybe I work too much and need to get out once in a while. Who knows? But, bottom line, it is possible to have LATEX on an iPad.

Bart Childs then talked about the problems of automating the process of converting text from word processors into LATEX. The problems that some converters face is that they usually attempt to make the final output look like the original typewritten document as much as possible. This introduces complex and, oftentimes, terrible LATEX code. Bart's goal was to find a conversion process that would produce code that was accurate, clean and maintainable.

Bart primarily used a hybrid process based on the Writer2LATEX plugin for OpenOffice and macros written in Emacs Lisp. Bart talked about his tests on converting a book on rotordynamics (which was heavy on the mathematical side), a C++ programming text, a memoir written by a friend that contained portions in the Czech language, and a novel.

Final Talks for Day Two Frederico Garcia's second talk focused on his `colordoc` package. This package was based on Frank Mittelbach's `docstrip` package. It highlights braces and other code delimiters and makes for slightly more readable code. This package is sure to come in handy if you have ever need to troubleshoot code and aren't sure if there is an extra brace or bracket lurking somewhere.

The final talk for the day by Jim Hefferon and Michael Doob focused on the Asymptote graphics program. Hefferon and Doob showed some of the features of the package and its ease of use in generating graphics.

Roundtable Discussion The day concluded with a roundtable discussion moderated by Dave Walden with some of the TEX consultants present as panelists, of which I was one. The other consultants included Peter Flynn, Amy Hendrickson, Christina Thiele, Steve Peter and Boris Veytsman. Questions asked by the audience focused on the business aspects of TEX consulting, such as how consultants got business and how they dealt with problem clients, just to name a few.

Figure 5: Conference room at the Omni Parker House where TUG 2012 took place

After the roundtable discussion, proceedings concluded for the day. Participants met for a banquet at the Oceanaire Seafood Room for dinner; the restaurant is located across City Hall Plaza.

Day Three The first talk of the day, by Pavneet Arora, was titled "Sleep de(p)rived typesetting", and focused on the process of typesetting. The second talk by Bob Neveln and Bob Alps looked at a Python program they created to check the syntax of mathematical proofs: ProofCheck. Mathematical proofs were checked against the syntax developed by A. P. Morse in his book, *A Theory of Sets*. The authors presented recent updates to the system that are designed to make proof checking easier to users.

For the next talk, Christina Thiele talked about her experiences using TEX, first to typeset articles and journals in academia to the creation of her company, Carleton Production Center. Christina also highlighted some of the changes in TEX that took place when she first started using it almost thirty years ago as well as the changes in software and hardware she underwent in her career as a consultant.

The following talk by Will Robertson and Frank Mittelbach looked at the origin, the development and recent changes to LATEX3. The key aspects and ideas behind LATEX3 were developed in the early 1990s but it wasn't until recently that code came into widespread use. The talk focused on the what LATEX3 can currently do and their plans for the future.

After Lunch The next talk by Boris Veytsman and Leyla Akhmadeeva focused on the results of a recent study conducted by the two authors to test whether typographic style influenced a reader's ability to comprehend and remember contents in a passage. It

is generally believed among typographers that typography is more than an art as it can influence a person's comprehension and reading speed. To test this hypothesis, the authors gave university students a one-page passage and tested their comprehension. LaTeX was used to control various typographic features from fonts, page layout and justification. This preliminary study was intended for textbook designers where comprehension of text is very important. The preliminary study indicated that typography does not play a part in reading speed or comprehension and the human brain may be flexible enough to allow us to read even badly designed pages.

While this first study may indicate that typography does not have an effect, I definitely would like to see more research on the area. Though short passages using bad typography may not show an effect on comprehension, perhaps the effects of typography and longer passages may affect a person's ability to understand a passage.

Norbert Preining followed with a talk about the recent changes and additions to TeX Live. One of the biggest changes is the extension of the TeX Live manager to read multiple repositories, something that has been a feature of MiKTeX for some time. Norbert also gave an overview of the other changes to TeX Live 2012.

The final talk of the conference was by Didier Verna and looked at possible modern implementations of TeX. Didier said his current project came about from a discussion he had with Donald Knuth, the creator of the TeX typesetting system. TeX was initially designed to be a simple system as the computer resources of the time were limited and this meant that a full programming language could not be implemented.

In the time that has passed, we have seen an exponential growth in computing power along with our skills in language design and implementation. Could this be a way to modernize TeX? Didier says this is possible and can be done using an old, but very established modern programming language, Common Lisp. Didier focused on the features of the language that made it ideally suited to the task of modernizing TeX and gave some demonstrations.

Concluding Remarks and Observations

I have always known somewhere in the back of my mind that the development of LaTeX is community-based, much like many open-source projects. Though I am just a user of the system, it was great to meet others like me as well as some of the developers who are going to continue building and contributing to the evolution of LaTeX. As a first-time participant I truly had a wonderful time and attending future conferences is something that I definitely look forward to. Attending gave me the chance to meet others who were just as interested in LaTeX as I was and also gave me the chance to see the future of TeX. Attending is definitely something I can recommend to any TUG member who hasn't done so.

◇ David S. Latchman
texnical dot designs (at) gmail dot com

Roundtable: TEX consulting

http://tug.org/consultants

There was a roundtable on TEX consulting on the second day of the conference, moderated by David Walden (DW). He requested background information from the panelists, and circulated it to all the conference participants earlier in the day. Thus, the panel discussion itself could focus on questions more about consulting rather than just who they were and their basic businesses.

Most of the panelists are listed on the TUG consultants page, http://tug.org/consultants; most also have web sites that are easily found.

Peter Flynn (Silmaril Consultants) I was introduced to TEX around 1980 and have been using it daily since about 1985. I started consulting in 1990, although I had done a few odd TEX-related jobs for organisations as diverse as the Church of England and the Local Government Computer Services Board before that. Most of the work is writing document classes and modifying packages (rather than writing whole new packages), but we also proofread and typeset books, and advise on the adoption of TEX systems. Clients include individual authors, one-person training companies, learned societies, auction houses, printers and typesetters, government offices, manufacturers, and publishers.

Amy Hendrickson (TEXnology, Inc.) I've been doing LATEX consulting for nearly 30 years, since 1983. My steady client is John Wiley & Sons publishing. I provide book class files for Wiley and then assist their authors with their particular needs or questions. Other than Wiley, I have many and varied other clients coming to me, finding me through my website.

In the last few years I've worked for quite a few overseas companies, including an Italian climate change organization, a Swiss software company, a German university, where I did the design as well as the LATEX implementation on an ejournal for their Institute of Advanced Study, and a Brazilian economics journal. I'm working now for a Swedish scientific organization, to produce a scientific journal package and documentation, and have been asked to come teach in Stockholm in December. The other parts of my consulting include teaching LATEX two or three times a year, and occasionally doing book production, for Wiley authors, or for others that come to me. An example is a client last year who asked me to design a book style and implement it for an 800-page book on oil drilling that he is self-publishing.

But the usual kind of work I do is LATEX macro writing, commonly for book or journal class file development, with tabbed documentation in PDF; but I also do macro writing in response to specific requests. A recent example is a macro set for an online financial report generator that uses LATEX in the background.

David Latchman (TEXnical Designs) I first started using LATEX about ten years ago, mainly to do assignments. As I did physics, using LATEX made the typing of equations easy (if only I had known about this as a physics undergrad). I slowly came to appreciate writing in TEX and started using it more and more until I started using it almost exclusively.

Soon others came to me with their problems in LATEX. That grew when I noticed that people were posting their problems online. I started with typesetting articles and dissertations but the projects eventually grew in scope. Most of the subject matter I deal with is still technical in nature; I focus on the sciences, engineering and mathematics. Soon I was typesetting books; I have typeset some books for companies for their internal use, as well as creating of reports. I also modified existing style and class files, leading to the eventual creation of style and class files for clients.

My clients to date have included students (mostly graduate), writers and businesses. About a year ago, I decided to start my company, TEXnical Designs, in the hopes of expanding my services. My most recent project uses Sweave to pull and analyze data into a report, a project I am looking forward to.

Steve Peter (Beech Stave Press) I've been using TEX and friends since 1997 and have been consulting since the turn of the century. There have been three distinct areas of the consulting work: first, I write document classes and packages (or in many cases, customize existing ones); second, I support authors using these and other packages to write for publishers for whom I consult; and, third, I provide editorial and typesetting services outright for students writing theses, authors publishing in online journals (which now often require the authors to hire copyeditors and compositors), NGOs, and various publishers.

Christina Thiele (Carleton Production Ctr.) I started using TEX in 1983, on a mainframe, with no previewer. The bulk of my involvement with TEX has been hands-on, just-me, typesetting work. I've worked on at least a dozen journals, ranging from a few years to well over two decades of their issues. Almost all have been in the humanities, but not all

in English; there were journals in French, Spanish, German, Italian, with small stretches of more exotic fare, such as Arabic, Korean, and Hebrew.

But I don't think I did anything that was really and truly "consulting work in TEX" until 1998, to help the National Research Council's Research Press start using TEX for the Canadian Journal of Physics, a small-format single-column monthly. The macros were originally done by Robin Fairbairns — I was the local contact, doing training, first-response troubleshooting, and answering questions of all sorts. I was a consultant at last!

Boris Veytsman I have used TEX since 1994 and consulted since 2005. I mostly do customized LATEX packages, troubleshooting, and teaching. Among my customers have been No Starch Press, Israel Journal of Mathematics, Annals of Mathematics, Philosophers' Imprint, American Chemical Society, Association for Computing Machinery, US Army Corps of Engineers, US Census Bureau, UN Food and Agriculture Organization, and many individual authors, publishers, etc.

Panel discussion

DW: We've already met the panelists, so we won't go through the bio stage. And you've already heard from five of them, giving presentations; one more will happen tomorrow. Another thing I'd like to say is that we're having this panel discussion on consulting because we're curious, and possibly we'll learn something that will be useful in our amateur work. I'm sure we're not here to somehow create competitors for the consultants.

I've collected some questions in advance, and I'll start by asking a few of those. The panelists should feel free to quiz each other, or to answer each other's questions, or to follow on to each other's questions. And of course in a few minutes we'll begin to take some questions from the group. The first question is a little bit of a self-serving TUG question: Raise your hands — How many of you get business from your advertising in *TUGboat*? [Four hands go up.] And, where else do you get your business?

DL: I've gotten some from Facebook. And there have been a few on-line free-lancing sites that I've tried.

BV: Mostly word of mouth, besides *TUGboat*. Somebody, a former customer, recommends some friends.

CT: Word of mouth from editor to editor, which usually means the humanities; professor to professor, same department. I'm a member of the Society for Scholarly Publishing, SSP; many years ago I did a poster session on the use of TEX for linguistic journals, well, not just linguistics, and eventually someone from OUP got in touch with me — it took them about 14 months to track me down; they obviously don't know about TUG — they found me and I got two books out of that.

I had a query that came through the TUG office which had some through a publisher in Sicily; they had a book that had some in in very old LATEX, and they needed it redone. I haven't gotten work through any ads in *TUGboat* because I don't have an ad there; it's not that nobody looks at my ad.

I've had work come through poster sessions; membership in the SSP, where you're allowed one little advertising blurb for free, so your name is out there; and a number of clients came as a result of something I'd done when I was still at Carleton University. Another editor and I went around and polled profs to see who edited journals; and we found over 40, most in the humanities, but not all. So I would say that a university is a really good source of clientele. You put out an ad in their publications or their newsletter — academics have a newsletter, students have a newsletter — and that probably works just as well.

AH: I think the place where I get most work is through my website. I just sit with my website out there, and somebody will come to me. I've had people from all over the world ask me to do various things; that's been pretty useful. *TUGboat* has been useful as well, and also word of mouth. I started out working at MIT, so I have some MIT/Lincoln Lab connections. That's my general stomping ground. More than that, it's just people who come to my website.

PF: Like Christina, I don't have an ad in *TUGboat*, but I do have a website. I get a lot of traffic through the website. Also, going to conferences is a major source. In my case, not just TEX but XML as well. Being there and meeting the people is probably about 65% of the battle.

BV: I have a story about word of mouth. You know, I have several jobs — an evening job at a university, a daytime job at some company. And once I was recruited for a permanent position about TEX; somebody read about me in *TUGboat*, they wanted to recruit me; I said I like my positions and don't want to move, and this was a very wise decision, because, after some talk, I understood that I was being recruited by another department of my own company. So my answer that I was pretty happy where I am was probably a good thing.

DW: A follow-up question was, how do you communicate with your customers? I assume phone and

e-mail. All of you? [all nod] In person, some of you? ["occasionally"]

BV: If you do teaching, seminars or classes, obviously you meet your students in person. For everything else, e-mail is good.

PF: If you are using a website, then having a client subdomain is basically an extranet with user name and password access for clients. You can use it to download, to place copies of things, to keep in contact with the client.

DW: To what extent does anybody's consulting go beyond LaTeX, or is it primarily LaTeX?

PF: XML...

SP: Copyediting, editing ...

DW: Proofreading ...

AH: Design ...

DW: Well, that's my follow-on question. How much of the time does your TeX work take you into design, either of the book or of fonts?

AH: Not as often as I'd like. The last few years I've had the opportunity to do some design work, and actually it's been a lot of fun, although also time consuming, and perhaps not very profitable. For an e-journal, I did the design as well as the implementation of it. I've also done design work for small companies who wanted to use their logo in a particular way. I find that's kind of fun, but it *is* really time consuming. I don't know that I recommend it.

BV: For me, it's the opposite story, because some clients want me to do design more often than I wanted — I never tried — but now I know to whom to recommend it.

DW: There's a followup question over here — the same topic?

Pavneet Arora: In what form does the design reach you?

BV: My favorite form is specs. You have so much baseline, so much space in the margins, and so on. Unfortunately, many designers don't do this; they just send me a sample, and I have my trusty ruler with the different scales, where I measure. Unfortunately, sometimes they just do this.

CT: I usually just have journals, so once you set the design, all you do is plug the data in each issue. But when I'm asked to start up a new journal, which has happened three or four times, I prefer *not* to be given the specs, because most of the editors are completely unaware of what those mean, and I would much rather have them go cruising the shelves or go cruising through Muse[1] and, although you have to be a subscriber to get the current issue, you go in there, you can go by category, mainly humanities as I said, and then if you go into any specific journal, go all the way down to the bottom of the list of their collection — the oldest issue is very often free. So you open it up, and you take it — that's what I've done — you look at the title page, the general pages, the bibliography, and after about five or seven of them, you realize that a lot of them don't have a lot of the information that you think could go in there. A lot of journal design is very static; it hasn't been reviewed in a long time. So they don't have things like e-mail address, contact info, the main author to connect with, ... That's very common in science journals, but not in the humanities; they don't seem to want to talk to anybody, just talk *to* you in the article, but please, don't come back to me. It's very different in the sciences. So Muse is a really great source for inspiration, and motivation to do better. As I said, after five or six journal title pages, you can see where you can do better. Of course, techniques are so simple, it's great.

AH: My specs for that job I was talking about, the e-journal, the specs were the color scheme and some graphics, and that was it. So you can imagine, there can be a lot of choices, so what do you want to do — do you want to have the colors graded through the page, how do you want to link this page to the other page. And also, I found my own graphics, by going to the web and finding things that didn't have copyrights on them, and then using those graphics. Similarly, for doing book design, I was able to do a few book designs recently, either because the authors didn't know what they wanted exactly, or they just let me loose with it. One was a book that dealt with "the theory of everything", so I was able to find a graphic that matched that, and used it for the chapter headings. Another one — it was kind of a weird job — a guy came to me who was from the oil fields of Texas; he had done work with oil wells, and he wanted to write a book on oil wells to share the information he had gained over the years. So I found some graphics that had to do with oil wells and put them in the book, and he was happy, and I was able to use whatever colors I wanted to use, and so forth, and produce the book for him.

So I think, if you're given a set of specs, that's trivial; you just implement them. I've done that plenty of times; it's not terribly interesting, but I can do it. But it *is* kind of fun when you have a little

[1] Muse is an electronic warehouse, with electronic subscriptions, mainly to humanities journals. It's run out of Johns Hopkins. *Ed.*

bit of leeway so that you can do something more interesting.

PF: My initial reaction is that the design, everything should be black, shiny, shiny black. [SP: and lucrative] And lucrative. And corrupt. I have not decided. I have ten designs that I need to design. To answer the question, most clients will provide a PDF of the sample, oversize, big borders, and with what they fondly believe to be measurements. Unfortunately, the way they do these is frequently very ambiguous. They get the dimensions wrong, they get the scale wrong, you try to print it and it's nowhere even vaguely near what they request, because they've test printed it with Acrobat with the scaling turned off. They haven't got it right to start with. And, like Boris, I'd much rather they'd give me the spec, a positive spec. But if they do, and even if they try to do it in the design, they leave out stuff. They don't specify chapter starts, and they don't specify heading 1 and heading 2, and then no heading 3 layout, even though the book contains heading 3s. The book will contain, for example, bulleted lists, but they won't specify how much they're indented. Many, many of these specifications are grotesquely defective. So there's a constant stream of going back to the client, and saying "What do you want this to look like?" And they go back to the designer, and the designer says, "Oh, well."

DW: Steve, you also work on the design of the parts, yes?

SP: Yes, but not anything I've gotten through this. What I get is a surprising number of organizations that have things like occasional papers, and they've already got a Word template that they give out to authors, and then they want to support TEX users. I'll get a copy of the Word template, which in some cases is surprisingly tough. I had one recently where the display math was supposed to be left justified if it was fewer than three lines, but centered if it was more than three lines. And my reaction then was "Why??" And they said, "Well, that's the way we do it. We just press the button and you either center it or left justify it." I could probably do it for you, but do you really want that? I think I've talked them out of it; they've reserved the right to come back to me and demand it to be implemented that way. I'm hoping they avoid that.

BV: Speaking about specs, once I had beautiful specs! Absolutely beautiful; it was a journal, and the author was a professor of typographic design. It was absolutely beautiful, exactly stated, it was very difficult to implement because it was grid based of course, and you know how to do grid-based typesetting in TEX, it's . . . but it was beautifully done.

DW: This brings to mind another question: you've talked about how the customers will come back to you, you go back to them. While I'm sure that sometimes happens in the slightly longer run, therefore my question is, How often do you get change requests when you maybe think the project is done? How do you *know* it's done? What do you do with those?

BV: Nobody wants this? Okay. I made a mistake in the beginning of my work as a consultant. and it's too late to correct it. The mistake is I said that you have free lifetime support. [laughter] And you wouldn't believe how many times I have requests for this support. What I do is I say, "If you want me to correct my bugs, it's free. But if you want me to redesign new features, I'd rather be paid for this." But still, if there are some bugs, some situations that are not bugs, but I didn't realize what they wanted, if you start doing it, you are going to do it for a long time. Sometimes it's reasonable to ask money for this; many times you just do it for free because . . . just because.

PF: I did very nearly the same that Boris said. I offer support for fixing bugs, but not enhancements. I did get, I think, one argument from a learned American society for whom I'd done an SGML-to-XML-to-LATEX conversion program, which was running in batch on their web server, running happily for years without any problem. And they rang me in a panic on a Sunday night at 8 p.m. to say that it was broken, my program. I suspected the bug was in their data, not the program, but by the time I'd gone and checked it out, I discovered they had a new sysadmin, who had *removed* some key features from their operating system, like `rm` [laughter] because he thought they were unsafe. But the script I'd written on several occasions removed its temporary files, and was falling flat on its face. Trying to explain that to the customer, that their own staff member has accidentally fouled things up, taught me a valuable lesson: you can't trust a customer to provide you with a platform on which your stuff will execute. So you have to write your scripts so that before they start, they test for the presence of everything that you need, even system facilities, and they will issue an error message if something is not there. You *do* have to be incredibly careful, if you're going to undertake this, to make sure that stuff that's supposed to be there is actually there.

AH: I deal with a whole lot of clients. I will make a macro package with documentation and template files and so there's a solid piece of work there. And

then lots of different authors will use that work, and as the authors use it, they will say, this needs to be changed, or I would like this additional feature, and so on. So it's just ongoing support of authors I think of, rather than redoing the macro file. I find as a consultant that this can sometimes be tedious, but on the other hand, it's kind of nice because it's kind of steady at the same time, so it's kind of nice to even out your funds with this kind of work. That does tend to be useful, I think, to support authors as they're using your macro package.

CT: I only have one client like that, a very long term with the National Research Council in Ottawa, the Research Press, which about a year or so ago got sort of spun off into a non-profit, so they pay bills faster, they get back to me faster, and they don't go off on very peculiar adventures because of principle or personal hobby-horses. We did it for quite some time, until about 2005 — and then they went away from TEX to XML direct with Word and 3B2. So one of the journals is coming back to TEX, which has meant that we had to ... we said, yes, we can give you support again, starting with a new version of TEX, because they were using Y&Y, which was great, but now defunct. So we are in the process of taking macros that work perfectly, even now, under Y&Y, amd we have to upgrade and haul them forward to use PC TEX and that structure is different, so now they're saying, "Well, can you make it more like..." Well, no! It's a different product, different company, different GUI, ... Well, they're not keen on the editor. "Oh, well, that's not the upgrade, that's going to be the support and maintenance", so we're having to make a distinction between the get-us-going again phase, and separating those tasks from "and could we do", which is what we say is the future support for their maintenance, and that's a different contract. So we're trying to distinguish those two. That, of course, is different pay.

SP: One thing that I've noticed, most of my ongoing work is author support, things of that sort, which is charged by the hour. But I have noticed a definite correlation between the amount that they're originally willing to pay, and how much they want to come back to you for free changes. So if they're going to nickel-and-dime you at the beginning, they're going to come back and ask for an amazing amount of free work.

DW: Well, a question that I think has come up here, that is the second part of the previous question, is: First, how do you deal with people who, they have different distributions and tools than you do, and for these instances where you go back and forth with them, perhaps after a long time, how do you archive the setup that you were using when you worked with them before, so that you can come back to them? Or do you?

SP: Personally, I've got five different computers set up in my office, and a number of virtual machines. I try to match a large number of setups. Obviously, I don't have VMS or anything like that running. Rarely do I have a request to do that. But either I can match directly, or ... Really, TEX has very few problems that I have hit with flat version dependence. ... Knock on wood.

CT: I have some very, very old files that every so often have to be revisited. I've been doing the *Canadian Journal of Linguistics* for many years. It started as plain TEX, then eventually it went to 2.09, now it's 2_ε. But now, for Muse, we're going to try to put all of the back issues up in electronic form, and PDF. So I'm going to be revisiting files from volumes 34 to 48, which go through all those three. And so on my machine, much to my husband's dismay, I have something called "oldTEX", and I have something called "RnewTEX" — that's "really" new TEX — and I'm hoping that I can still use oldTEX with plain and .09, and ε with RnewTEX, which is actually 2003. (I know; we'll talk about that tomorrow.) So I have some very old installations. I have Y&Y running under Win98. (You see, when you're a one-person operation, you're a very old archival type, historical.) The fellow I work with for the NRRC, Paul Mailhot (he's in Halifax; he's also a consultant like this), he has four or five machines with various TEXs — PC TEX, old and new, Y&Y, I think he has a couple of the public domains as well — so one just collects, Mostly they stay on the old machine that they lived on first, and when you advance, you advance your TEX, and if you have to, you cross borders, but otherwise, they're dedicated to certain clients and certain jobs, until you run out of room.

AH: I find that fonts are the thing that's most problematic, in terms of not being transportable. So I'm so grateful to Will Robertson for the `fontspec`. That's just great, and it also gives us access to all those nice Adobe fonts. That's a terrific improvement in *my* life. Other than that, I haven't had much problem with version dependence, really.

PF: I don't have a problem with version dependence, because most of my customers are keeping up to date. What I've got is a few who are no longer customers who may well want to update stuff, and if they come back to me still running LATEX 2.09, then I strongly suggest that they upgrade. I *do* make a rigid rule that they use an SVN repository.

DW: I have a couple more from the audience, and then I have a couple more that we made up just in case there weren't any. Suppose I want to write a LaTeX package, but have no experience. This person knows how to do LaTeX, but doesn't know how to do a package. Where would this person start — are there tutorials, textbook chapters, other suggestions? We may have someone here who would like some consulting, but free consulting, I'm pretty sure.

BV: There is a guide, `clsguide`, in "old TeX" how to do it. For other people, it will probably be different, but for me, I started by reading source code by other people. This is probably the best way to learn to look at packages, to look at `.dtx`, just look how people did things. Some people prefer textbooks, but for me, reading source code is the most educational thing.

PF: Yes, I'd agree with that — reading the source code, but reading it very *critically*, because some of it is a bit fake. The `clsguide` document is definitely a good place to start. Also the three chapters at the end of the latest *Companion*, which describe how to write packages. Those are very good. There are some serious problems with the documentation for `.ltx doc` because it doesn't explain terribly clearly how you embed multiple output files into your `.dtx` file. The documentation is ambiguous and that key piece is missing, or not terribly clear. As I said earlier, in my talk, I cheat, in the sense that my `.dtx` files are all generated from an XML document, so I can retain control of the way they're output. If I make a change, I can just do a regression test on all the documents and they all come out looking the same. But that's just me.

AH: My suggestion is Victor Eijkhout, *TeX by Topic*. That's very helpful, because the other basic reference guide is *The TeXbook*, and it's not well organized, really, in terms of, if you want one bit of information. So I think Victor's book, which is free to download, is a real good help. And also just starting with an existing `.cls` file, see how running heads are handled, for instance, you can make a variation on that pretty easily.

BV: [Victor's book] is free to download, but if you want to save your eyes and if you want Victor to have a couple of bucks, I would suggest buying it from Lulu. I did it myself, and it's one of my favorite books, to tell the truth, when I do TeX.

DW: It doesn't get Victor any money, but they're also for sale, used, on Amazon.

Another, very specific, question from a member of our group. He's working on a textbook, copyediting it, which has two `.bib` files. When he compiles it to PDF, sometimes the relevant `.bbl` file changes, and sometimes it doesn't. He wants a glimmer of, what makes that file change, and what makes it not change.

AH: Did he run BibTeX on it?

DW: I presume BibTeX was run on it? [Yes.] My own experience in compiling, and I'm far from an expert, is every once in a while it seems to pay to delete all the `.aux` files and somehow make TeX think that it needs to start from scratch.

CT: I've done that.

BV: I would look into the logs, the `.blg` file (which is the BibTeX log), the main TeX log, because sometimes BibTeX just dies under you, and if you are not careful, if you use an IDE which does not tell you, I prefer Make because Make just stops when something goes wrong. But if you use an IDE, sometimes it just dies and you don't know this, and you happily go further, which would be wrong. So just look at the log files and they will tell you a lot of the story.

SP: Sacrificing a chicken under the full moon is the best way.

PF: If the file is changed, and sometimes not, my immediate reaction is to run Diff. *What* is TeX'ed? The only thing I can think of that can immediately affect it is if you have two works by the same author, one in each `.bib` file, of the same year, so on alternate occasions, it would be computed as "1994a" and "1994b", or something like that. And if you updated a part of the document that did not cause that to change, it would not trigger that addendum.

Nelson Beebe: I've probably done more BibTeX than anybody else. Here's an example that I use extensively in my bibliographic work. You have an article on a particular topic, and six months later, an erratum is published. So I put in the BibTeX entries a pointer to the original to be read so if you cite the original the first time through, you get the original. There's still a new reference dumped into the `.aux` files if you need to run BibTeX twice. So the reality is, for heavily cross-referenced bibliographic needs, you really have to run BibTeX every time you run LaTeX, and then probably one more time. If you're using something like Make, you end up with something that can be set up with the right number of steps consistently.

Unfortunately, all of these people work with GUIs that click a button; they can't remember how many times they have to run the index program and BibTeX, and other programs. They get it wrong.

BV: Yes. If you have several entries from the same book, then the threshold is such that the book might

be good, or it might not, then unfortunately it is done only on the second run of BibTeX.

DW: Do the panel members have any questions for each other?

SP: One thing I want to bring up, one of the reasons I suggested we have this discussion is, about three weeks ago, I had a call out of the blue from another consultant listed on the `tug.org` site. He just wanted to chat, and said, basically, "How do you guarantee that you get paid?" Because he had a couple of cases recently; one of them he described, and I know who it was because the guy contacted me at one point, and I wasn't able to do the work at that point, so i passed him on, I said just look at the `tug.org` site. But I personally have had a couple of jobs and it seems to be an increasing case in the past few years of people getting work, and not getting paid. In general, if I've got somebody coming to me who says "I need you to do x, y and z and I need it by next week", and it's going to be a $200 job, you can't really say "Send me a check for $100 first, before I'll start on it." If it's a small amount and a short-term job, I'll just go ahead and do it. And those are rarely the problems. But it's the ones that are $1500 and they want things, what do other folks do?

DL: I know some people who have had this problem. One way to combat it is to ask for a partial up-front payment. In some cases maybe 10–15%. In cases where you have to collect the rest of your money, unfortunately, hopefully you have enough information on the project client to give to some collection agency. Beyond that, there isn't really much that you can do.

BV: If you have organization or corporate customers, they always pay. You just sign an agreement. Maybe there are some corporations that don't pay, but with individual people, ... If it's $1500, you can always say, pay me 25% up front. But when it's a couple of hundred, you cannot do this, and several times I was burned. You cannot do much when it's a couple hundred. Well, maybe it's a cost of doing business.

I'm sorry if some of you are economists, but I have a very low opinion about economics students, because it seems they are taught that morally it's something which should not be done, but business efficiency is very important, and one part of business efficiency is not to pay when you owe. I have a problem with business students, unfortunately. Maybe it's just my bad luck. When it's a couple of hundreds, you can't do much. You could go to small claims court, but you'll spend more time. What can you do? But to say the truth, lots of students are very good. I had one student who said "I cannot work with you unless I pay you up front. Let me just pay you my bank balance." I said "Fine." There are many people who understand this.

Robin Laakso: I have a suggestion. Sometimes when you go to a restaurant, especially a small one, and they have a bounced check, they'll put it on the wall. They'll have a list, and these days, a list on the wall or a blog.

BV: They could try to sue you for defamation. I wouldn't. It's very interesting, but...

DW: I think when they put it on the restaurant wall, their excuse is not that I'm exposing this person to the world, but I'm telling my staff not to let this person come in any more. So they've got kind of a fake excuse.

CT: I've actually had two cases. One time I got to a point where I had too much work, and I asked Steve if he could help on the front end grunt work. So I couriered the stuff down, it was for a university prof, not from my university, and it took so long to even get messages and responses back from him. amd we lost money. Steve did all the way up to the PDF proofs, I gave it to the author, and it finished. We had no recourse, I couldn't find him any more, the material was actually time sensitive, and five years after economic forecast papers, that's pointless. So it was a total loss. And then there was another where it was a very large book, and it simply took a lot more time than the two authors had found funds for. So whatever they *were* able to scrape up, I gave to the fellow I co-worked with, and I got zip. But it was for one of the authors who was editor of a publishing house for which I had done at least two series, at least 20–30 volumes over time, so I felt morally obliged to bring this book to its conclusion. And it took a very long time, there were two rounds of very thorough edits that came from other colleagues, yet more time. There *is* no recourse, because there was no money. That's one of the problems with getting jobs through university profs who apply for funding: they underestimate savagely, because they don't think a large sum would get them money. Somehow, magically, things will work out. Of course, if you feel morally obliged, they *do* work out.

AH: I actually have good experience with economics people! I've worried plenty of times. This is my way of making a living; it's *not* a luxury. One time I was working for a guy with a journal in Brazil, and I think, I have no recourse! If the guy doesn't pay me, he just doesn't pay me. So you're kind of relying on the people's sense of honor and the fact that you work with them in a nice way. And they

usually come across. There was one example recently where I had a guy who sort of had a temper tantrum. He said "Oh, I can't use what you did, so I'm not going to pay you." Oh, well, that's not going to work. So I just had to talk to him, long distance, for like four hours, and soothed him through whatever his problem was, and showed him how to get the results he was looking for, and eventually that all worked out. However, I will say the other problem I've had is that people don't necessarily pay quickly. When this is your rent money, that's an issue. So I always suggest that anybody doing consulting have a good cache of money in your background so that if somebody plays around with you for a couple of months without giving you the money they owe you, you can still pay your rent.

BV: [...] For me the US government was a very good customer. They paid on time, and very well and very fast. Basically, the UN was also fast; they also paid well. Second, what I wanted to say, there is some asymmetry here. The secret is this: even if the customer doesn't have any money, and there are some problems, it is very difficult when you have — I'm speaking about myself — some sort of moral feeling that, well, the customers don't want to pay, but for us, not to finish the job, not to do the job, is very difficult. I don't remember ever saying that I'm going to drop this client.

DW: Nelson, you had a question?

NB: Two related questions. They have to do with the adoption of what we consider technology futures. One is color, the other is input [...]

DW: Let me repeat the question. Color is getting easier to use, color laser printers are getting inexpensive. The other part of the question is, Unicode is out there but it's somehow mixed up sometimes with other codes. To the panel: what is your experience with color and with Unicode?

PF: Unicode, yes, because most of what I do, whether I end up doing it with TEX or something else, comes through XML. The customer has all kinds of files and all kinds of mixed formats. I turn it around and tell him, I'm sorry, you have to provide [better input]. I can't do it — it won't process!

AH: Color's great, and that's the wonder of PDF files.

CT: Just a very small comment. I offer PDF in color even before their electronic versions because the journals now want both hard copy and ...

SP: The question is, do you get an opportunity to put in a colophon on your work?

SP: So far, no, because most of the consulting work I'm doing is class files, and they really don't want to call attention to the fact that they're using TEX, especially since, lots of times, they're matching Word. I've had one book that I was requested to do the typesetting on that I was able to put a colophon on. The one commercial book that I did in ConTEXt, I put it in and it got printed the first time, and for whatever bizarre reason, when the press, Cambridge University Press, did the second edition, they retypeset it and left off the colophon altogether. But you could tell it was no longer my typesetting.

CT: For the job that paid nothing, the editor had several series going — there are some Renaissance plays, all sorts of stuff in that vein, Italian plays, Spanish plays, English plays, and so on — and I *always* had a colophon, because he gave me free rein. I always had a colophon that said the Barnaby Rich Series is typeset using TEX with Palatino fonts. Every single one. Always printed. And then I thought, every book that I did in between, all those books in that series, I would always generate a colophon page, and I would leave it to the publisher, with or without the editor, to make the decision about including it. So it makes no never mind to typeset that one page with a little block of text right in the middle and then let someone else decide. A lot of them don't even know what it is; they have to look up what is a colophon. That says something about publishers. But I would say, produce it, make it look good, let someone else decide if they want to include it, but you've done a good job by saying this is it.

PF: Sometimes the author or editor will mention in the foreword or preface that it was typeset with TEX. I leave it to the author. Other times I put a colophon. The publisher will frequently take it out because the publisher will re-typeset the legal blurb — that is the title verso — because they have their own stuff that needs to be included; so frequently the colophon will disappear because that page has been redone.

BV: Of course a good book should have a colophon. I have a publisher; I designed a style for this, and it has a colophon; and the colophon mentions fonts, everything, and my name. I don't understand why, but in e-books — they produce e-books and hard copy books — my name is on e-books, and somehow they delete it from the hard copy. Nobody has explained why to me!

SP: I always like the e-books that say they're printed on acid-free paper. [general laughter]

PF: How do you get an author to sign the e-book?

DW: I'd like to thank our panelists, all six of them. Thanks very much for taking the time.

My Boston: Some printing and publishing history

David Walden

During the four summers before each of my college years, I worked in a large cardboard box printing plant (big letterpresses and lithography presses) in an industrial town 40 miles east of San Francisco. Thus began my fascination with printing. I was also an avid reader of books and of *The New Yorker* magazine, to which my father subscribed. I dreamed of eventually living in a big city with big libraries and thick newspapers. Thus, after college, I moved in 1964 to the Boston area (where I have remained ever since).

As I explored Boston and Cambridge in the 1960s, I became aware of a number of publishing and printing activities, often by walking or driving by their then-current locations. I also began to use the libraries and to frequent the bookstores. Compared with the small town in the Central Valley of California in which I had grown up (and even compared with San Francisco where I went to college), Boston was a mecca for someone interested in books, magazines, and the related printing, publishing and distribution world.

With the TUG 2012 meeting (in some sense a publishing event) being held in Boston, I got to thinking about and then looking into the history of printing, publishing, libraries, bookstores, and so forth in Boston. In this note, a written variation on my presentation at the conference, I sketch what I have learned.

The presentation is at `http://walden-family.com/texland/tug2012`. *Numbers in braces in the following text (for example, immediately after this sentence) are the numbers of PDF pages in the presentation.* {1,2}

A lot of this printing and publishing history happened close to the conference hotel because Boston was once essentially a small island (the neck of land to the mainland was sometimes under water at high tide [Krieger99, Whitehill68]). {3} The location of our conference hotel was close to the center of this small almost-island. Thus, anything that happened in early Boston took place near the hotel location.

There are several maps which a reader might look at while reading this note (the conference hotel is at the southwest corner of Tremont and School Streets on all three maps): {4}

- 1772 Bonner map of Boston
- Boston Freedom Trail map
- Literary Boston, 1794–1862 map

All three maps are at `http://walden-family.com/texland/tug2012`.

Space and time do not allow a thorough presentation of the Boston-region history of printing, publishing, and the like. In particular, I have mostly omitted the author part of the literary world [Wilson00]. Also, this is not a scholarly piece of research (my narrative is based on what I have read in secondary sources, been told by someone, or found in Wikipedia). It also glosses over many details, for example, calling the early college in Cambridge Harvard and ignoring its early name. I hope my fragmentary narrative is suggestive of the actual history of events.

Colonial period, 1630–1775

{5} The Pilgrims, who previously had left England to go to Holland in the Netherlands, came to Plymouth, just south of Massachusetts Bay, in 1620. Another Massachusetts-based outpost was attempted at Cape Ann in 1624. In 1628–1630 a succession of largely Puritan settlers arrived in the Massachusetts Bay Colony settling in locations from Salem to Boston. (This section is substantially based on Thomas70 and Blumenthal89; see also Reese89.)

The Puritans came to the Massachusetts Bay Colony fleeing what they felt was the incorrect approach of the theology of the Church of England and the relationship of church and state (King James's approach subordinated the church to the state). In particular, in 1630 Governor John Winthrop and other Puritan leaders arrived with a charter allowing the Massachusetts Bay Colony to be governed from the colonies rather than from England. {6} Both Boston and Cambridge (a few miles up the Charles River) were settled by the Puritans around 1630. These largely Puritan immigrations to the Massachusetts Bay Colony continued for the next 10 years.

{7} The arriving Puritans were idealistic about their new home, and John Winthrop gave a sermon quoting the Sermon on the Mount and saying that they in the Bay Colony would be a "citty on a hill," watched by people throughout the world for the purity of their religious practice (and the way it was supported, i.e., enforced, by the government).

For all their concern to be free to practice their own religious reformations, the Puritans were not supportive of reformations by others. Roger Williams, among many others, was banished from the Bay Colony. In 1630 to 1658, several Quakers who refused to remain banished were hung, including Mary Dyer whose statue is on the grounds of the Massachusetts State House.

{8} The Puritans believed in education so that their citizens could study the Bible and read the laws and acts that governed them. By 1635 they had established the first public school in English North America, Boston Latin (there are signs on both sides of School Street outside the side door of the conference hotel noting early locations of the Boston Latin school). A couple of years later, Harvard College was established in Cambridge.

*

{9} In 1638 Rev. Joseph Glover contracted with Stephen Daye for the two of them and their families to travel from England to Cambridge (in the colonies) along with a printing press, type, and printing materials, where Daye would be responsible for setting up and running the printing press in Glover's home. This printing press, the first in British North America (the church in colonial Mexico had a Spanish language printing press a hundred years earlier), was at least nominally operated under the auspices of Harvard. Glover died before their ship reached Massachusetts, and Daye carried out his contract for the widow Elizabeth Glover. Stephen's son Matthew was also involved with the printing activity. Stephen was a locksmith and Matthew had apprenticed as a printer, so historians suspect Matthew did most of the actual printing. In any case, there was probably a lot of on-the-job learning about printing.

{10} After printing a couple of other documents of which no copies remain, in 1640 Stephen Daye printed the so-called Bay Psalm Book, the first book written and printed in British America. In most of the churches in the Massachusetts Bay Colony, the Bay Psalm Book replaced the earlier Psalm books the Puritans had brought with them from England — hence the popular name "Bay Psalm Book." Its actual title was *The Whole Booke of Psalmes Faithfully Translated into English Metre*.

{11} In 1649 Matthew Daye died and Samuel Green took over the printing activity. Green also did a lot of on-the-job learning. By 1656 Green had two presses. According to Lawrence Wroth in his contribution to Lehmann-Haupt52 (p. 8), for 40 years this activity was the "press of Harvard College," although there was not really enough work over the years to keep Green working full time. Green stopped printing in 1692. After Green, printing in colonial Cambridge was finished.

Green had 19 children, 8 with his first wife and 11 with his second wife, and many of Green's descendants became printers, forming a dynasty of printers extending up and down the east coast.

In addition to no liberalism in religious practice, there was no freedom for printing (at least within the Massachusetts Bay Colony). The goal of the print shop operated by the Dayes and then the Greens was to support the church and the commonwealth.

(As I understand it, the original purpose of copyrights — circa 16$^{\text{th}}$ century — was to control printing of books. The authorities only gave the "right to copy" to a chosen few who were allowed to print only what the authorities liked. There was somewhat of an English tradition of freedom of expression, but this was primarily about no prior restraint. Post-speech or post-publication, the authorities could punish expression they didn't like.)

{12} Some of the publications over the year of existence of the press were [Wroth, ibid]: "a book of capital laws . . . ; small pieces relating to the scholastic activities of the college; annual almanacs; a second edition of the 'Bay Psalm Book'; catechisms; a document relating to the troubles with the Narragansett Indians; a platform of the prevailing Congregational faith; and numerous sermons and doctrinal treatises."

{13} "The press reached the highest point of its activity with the publication in 1663 of John Eliot's translation of the whole Bible into the Indian tongue . . . " [Wroth, ibid]. This was a massive effort, producing over 1,000 copies and requiring a special shipment of paper from England. It was the first Bible printed in the western hemisphere. On the title page, Samuel Green is listed as the printer, and his apprentice Marmaduke Johnson is also listed.

To slightly paraphrase Wroth, this Cambridge press did its job of being, over half a century, an intellectual force in a new and rude environment.

*

{14} From 1674 on, printing was also allowed in Boston, on a case-by-case basis. Marmaduke Johnson received permission to print in Boston, but died before he could do any printing. Some of the following Boston printers were [Thomas70]:

- John Foster, 1676–1680; licensed to do printing, he was the first person who actually did printing in Boston.
- Samuel Sewall, 1681–1684: he was a bookseller, licensed to do printing, who printed acts and laws and books for himself and others; Samuel Green Jr. was his printer.
- James Glen, who also printed for Sewall before going out on his own.
- Samuel Green Jr., who printed work both for himself and for booksellers and was allowed to continue printing after Sewall's death; Green died in 1690.
- Richard Pierce, 1684–1690, the fifth Boston printer, who printed for himself and booksellers.

- Bartholomew Green, who first worked for his father in Cambridge and then took over his brother's activity in 1690. In 1704 he started the *Boston News-Letter* for the postmaster, who somehow asserted a right to have a newspaper.

There were a number of other printers in colonial Boston, i.e., between 1700 and 1775 when the Revolutionary War started. All this is detailed in Thomas's book.

*

Looking beyond printing, there were no strong lines between trades. Printers worked for booksellers, booksellers did some of their own printing, printers published newspapers, binding was often a separate trade but not always, and printers publishing newspapers did some of their own writing.

{15} Isaiah Thomas lists about 90 booksellers between 1641 and 1771 [Thomas70]. Initially there was a bookseller or two in Cambridge. Next there were booksellers in Boston, particularly along the street and slope known as Cornhill. Hezekiah Ushel was the first in Boston, 1650–1771.

Booksellers sold (and sometimes printed) acts and laws, books on religion, school books, books on politics, imported books, and new printings of books pirated from Europe. The shops of booksellers were often also community meeting places.

*

{16} Colonial Boston also has a rich history of newspapers [Thomas70]:

- The *Boston News-Letter*, 1704–1776, was started by the Boston postmaster and printer John Campbell. This was the first newspaper in Colonies. It had lots of editors over the years, and was printed through the siege of the 1770s.
- The *Boston Gazette*, 1719–1798, was started by the next postmaster, William Brooker. (Apparently postmasters thought they had a right to have a newspaper.) Brooker hired James Franklin to do the printing. The *Gazette* had a long line of successor organizations.
- The *New-England Courant*, 1721–1727, was started by James Franklin, who had lost his job with the *Gazette* within a couple of years.

Isaiah Thomas lists another ten Boston newspapers prior to 1775, including his own *Massachusetts Spy*, published in Boston from 1770 through April 1775. I will touch more on this in the next section.

{17} Before leaving the topic of Boston's colonial newspapers, I'll say a little about the most famous colonial Boston-trained printer, Ben Franklin, who was born on Milk Street and baptized at the Old South Meeting House, and attended Boston Latin on School Street for two years [Franklin40, Isaacson04].

{18} Ben's much older brother James had gone to England to apprentice as a printer. He returned in 1717 with a press and a small amount of type. James's shop was at the corner of Court Street and Franklin Avenue (called Queen Street and Dorset Alley in pre-Revolutionary times [Drake70]). Ben was indentured at age 12 to his brother James to learn the printing trade. The indenture was to last until age 21.

{19} James started the *New-England Courant* in 1721, and it was the first "truly independent newspaper in the colonies and the first with literary aspirations" [Isaacson04]. For disagreeing in print with the authorities, James was imprisoned for a month in 1722. He was released on the condition that "James Franklin not publish the *Courant*," so Ben became the publisher in name. However, Ben couldn't be the publisher while still indentured to James, and so officially Ben's indenture was ended — although a follow-on secret document of indenture was made.

Ben contributed a lot to the *Courant*, including 14 humorous letters over six months under the name of the widow Silence Dogood. However, Ben and James didn't agree on things, and in 1723 Ben broke his secret indenture and went to Philadelphia, knowing that James could hardly admit that such a secret indenture existed.

By 1727, James, faced with continuing suppression of his press, had moved his printing business to Newport, RI.

Revolutionary War (1775–1783); transition

{20} The stamp act of 1765 was an incendiary event which produced resistance to British rule in the colonies. This was a tax by the British Parliament on printed materials in the colonies — the printed materials had to be produced on paper carrying an embossed revenue stamp. It was repealed a year later, but Parliament continued to assert its power to regulate the colonies and other taxes and regulations were imposed.

As resistance to British control increased, the colonial press participated and got in trouble. One of the printers involved in the resistance was Isaiah Thomas [Blumenthal89], to whose book I have frequently referred [Thomas70].

{21} Isaiah Thomas was born in 1749. His widowed mother could not support him, and at age 6 he was apprenticed to a printer with an indenture to age 21. He did both personal jobs for the childless printer and his wife and printing jobs. In particular, according to Thomas himself, he "set types, for

which purpose he was mounted on a bench eighteen inches high, and the whole length of a double frame which contained case of both roman and italic."

The printer was not too skilled, but Thomas nevertheless quickly acquired the craft. A decade or so later, he escaped from his indenture and went elsewhere on the east coast to try to learn more about printing. By 1770 he was back in Boston, initially briefly in partnership with his former master. In 1771, Thomas started his own newspaper, the *Massachusetts Spy*. {22}

{23} As time went on and discontent with England grew in the colonies, Thomas used his *Massachusetts Spy* to support the causes of the Founding Fathers against England, and his print shop became known as the "Forge of Sedition." as many resistance meetings were held there. Thomas himself refers to his press as the "sedition machine." A 1774 edition of the paper shows the famous "join or die" slogan (first published in a cartoon and essay by Ben Franklin in Philadelphia), meaning that the colonies must join together or they would die separately.

{24} Shortly before April 19, 1775, the day of the British march on Lexington and Concord, Thomas transported a press out of Boston to Worcester, 40 miles to the west. He snuck out of Boston on April 18, 1775, and briefly joined the Lexington militia. Then two days later he traveled to Worcester where on May 3 he restarted publication of the *Massachusetts Spy*, including the first printed accounts of the Battle of Lexington and Concord.

{25,26} After the war Thomas stayed in Worcester and, after some struggle, he began to develop a successful business. The business did well and Thomas became the "country's leading printer, publish, editor, and bookseller" [AAS12]. In Worcester, he had a big printing plant, a bindery, and a paper mill; he had branch offices, including one in Boston, and partnerships with a number of other publishing-world companies throughout the new country.

{27} "Thomas retired in 1802 and devoted the rest of his productive and long life to collecting, scholarship, and philanthropy" [AAS12]. He wrote the marvelous and comprehensive 1810 book, *The History of Printing in America*. "In 1812 he established the American Antiquarian Society to house his remarkable library of 8,000 volumes, with a mission to collect, preserve and make available the printed record of the United States for future generations. He served as president of AAS until his death in 1831" [AAS12].

Thomas's Old No. 1 printing press resides at the Antiquarian Society in Worcester.

{28} After the Revolutionary War, the press was no longer regulated, and the technologies of the industrial revolution were applied in the publishing and printing business. I have just mentioned the example of Isaiah Thomas and his success.

Initially rotary presses were available and, eventually, much more efficient typesetting machines. {29} Presumably using this technology, the *Boston Advertiser* was founded in 1813, Boston's first daily newspaper.

{30} However, in the years of the first third of the 1800s, Boston ceased to be the publishing center of what was now the new country. The big publishing centers were now New York and Philadelphia. Other cities such as Baltimore, Cincinnati and New Orleans also developed strong publishing activities.

Liberal elites of the mid-19th century

{31} Although Boston was no longer the country's center of publishing, in the 1800s Boston was the center of an important philosophical and literary movement [Barry12, Myerson80, Phillips06, Rose81, Wilson05]. (Look at Figure 1 in parallel with the following several paragraphs of description.)

The Puritans came here as Congregationalists, but individuals still needed to follow the doctrine and creed of their congregation (and to sign the Freeman's Oath to be a citizen of the Massachusetts Bay Colony). As mentioned earlier, if someone wanted to promote some other version of religion, the Puritans kicked that person out of the Colony with lethal punishment for not staying out.

As the Revolution drew near, the churches largely favored the revolution, and I suppose in some sense this was at least a partial departure from the conservatism inherent in the colonial Puritan churches.

In the early 1800s, the Unitarian approach became more popular: people could be religious using their own reason and not just reliance on doctrine and creed. Eventually Harvard appointed Unitarians as president and the divinity professor, and thus ministers coming out of Harvard were Unitarians, and in time a majority of the First Parish churches around the Massachusetts Bay region switched from Congregational to Unitarian.

Kings Chapel, across the street from the conference hotel, is a special case. Before the Revolution it was Anglican. After the Revolution it became Unitarian, but they liked their Anglican Book of Common Prayer and rewrote it to be consistent with Unitarianism. (Today they state, "We are Unitarian in theology, Anglican in worship service, and Congregational in governance.")

All this thought about reason, individual goodness and personal relationship with god, etc., helped

Church of England under the reign of King Charles

- Church subordinate to state (Biblical justification); much hierarchy, ritual, and decoration between the individual and God; anti-Calvinism (predestination)

Puritans in the Massachusetts Bay Colony

- Congregational governance without ritual or decoration; theocratic state; Calvinist; Biblical literalism; exclusive

Unitarians

- Each individual could find Christian truth through intellectual freedom, reason and empirical evidence (e.g., from the Bible but without literalism); one God (not Trinity); no original sin or predestination; concern with moral and social harmony of a diverse population; anti-revivalist; use of writings, lectures, etc., to reach out

Transcendentalists (several thrusts)

- Religion: intuition and spirituality rather than reason; Eclecticism instead of Christianity; practically, more of a philosophy and idealism than a religion; deliberately no component of being an organized religion
- Social movement: arguing for (and practicing) the goodness and self-reliance of man (partly as a reaction to urbanization and capitalism); impractical
- Literature: believed literature was a contributor to the betterment of man; called for a new, American, literature
  - wrote some now-venerated works
  - attitudes evolved to acceptance of the progress of the Industrial Age and international trade (and acceptance of capitalism)
  - further development of literature and a cultural environment
- Broader world of social improvement; active in the abolition movement, women's rights, education reform, and improving conditions of people in unfortunate circumstances

Figure 1: The evolution of spiritual doctrine from the Puritans to the Transcendentalists (my superficial, perhaps inaccurate, approximation of the transitions).

lead to the Transcendentalists and their idealism. (In national politics today, we sometimes hear about the Massachusetts/Cambridge/Harvard liberal elites. It's a long tradition. There were liberal elites in Cambridge, Boston, and Concord over 150 years ago.)

In addition to their philosophy and idealism, the Transcendentalists sought to create a new American literature.

{32} One of the important meeting places was Elizabeth Peabody's bookstore, which still exists today (as a restaurant) on West Street near the conference hotel. The plaque on the wall of the building gives a good summary of the importance of Elizabeth Peabody and her bookstore:

> Elizabeth Peabody, the first female publisher in Boston, maintained a home and business here in the 1840s. Her bookshop was the first in the city to offer books by foreign authors; and she published the periodical *The Dial* with Ralph Waldo Emerson. The shop was the meeting place for transcendentalists and intellectuals. Journalist Margaret Fuller [who lived on Winter Street, I believe, a couple of blocks away] gave lectures here called "Conversations," which were an important part of the early American feminist movement.

The Dial: A Magazine for Literature, Philosophy, and Religion was "an organ for the dissemination of Transcendental thought" [Wilson05]. The lectures were called "conversations" because women were not supposed to do public speaking in the 1840s.

{33} Another meeting place was the Old Corner Bookstore, which in the mid-to-late 1800s was both a publishing location for and meeting place of the transcendentalists and other intellectuals. I'll return to the Old Corner Bookstore in the next section.

{34} Founded in 1885, the Saturday Club at the Parker House hotel (our conference hotel) was another meeting place of intellectuals. (Not only did they do a lot of writing, they apparently liked to spend a lot of time in each other's company talking about their thinking.) Here Charles Dickens gave a preliminary reading of the Christmas Carol to the Saturday Club before he did the public reading next door at the Tremont Temple. Here they conceived and started the *Atlantic Monthly* magazine in 1887.

{35} It was in volume 1, issue 6, of the *Atlantic Monthly* that Oliver Wendell Holmes wrote in his series "Autocrat at the Breakfast Table" that Boston (specifically the State House) was the "Hub of the Solar System," suggesting that Boston was the center of everything commercial and intellectual at that time. The Massachusetts elites were not modest. At various times, the Old Corner Book Store and other famous downtown Boston locations have been called "the hub of the hub."

Mid-1800s on — Boston is just another big American city

{36} Despite Boston's claim as a intellectual and cultural center, by the mid-1800s, it was just another big American City, in general publishing terms. The history leading up to this brings me back to the Old Corner Book store [Hall10, Tebbel72, Tebbel75].

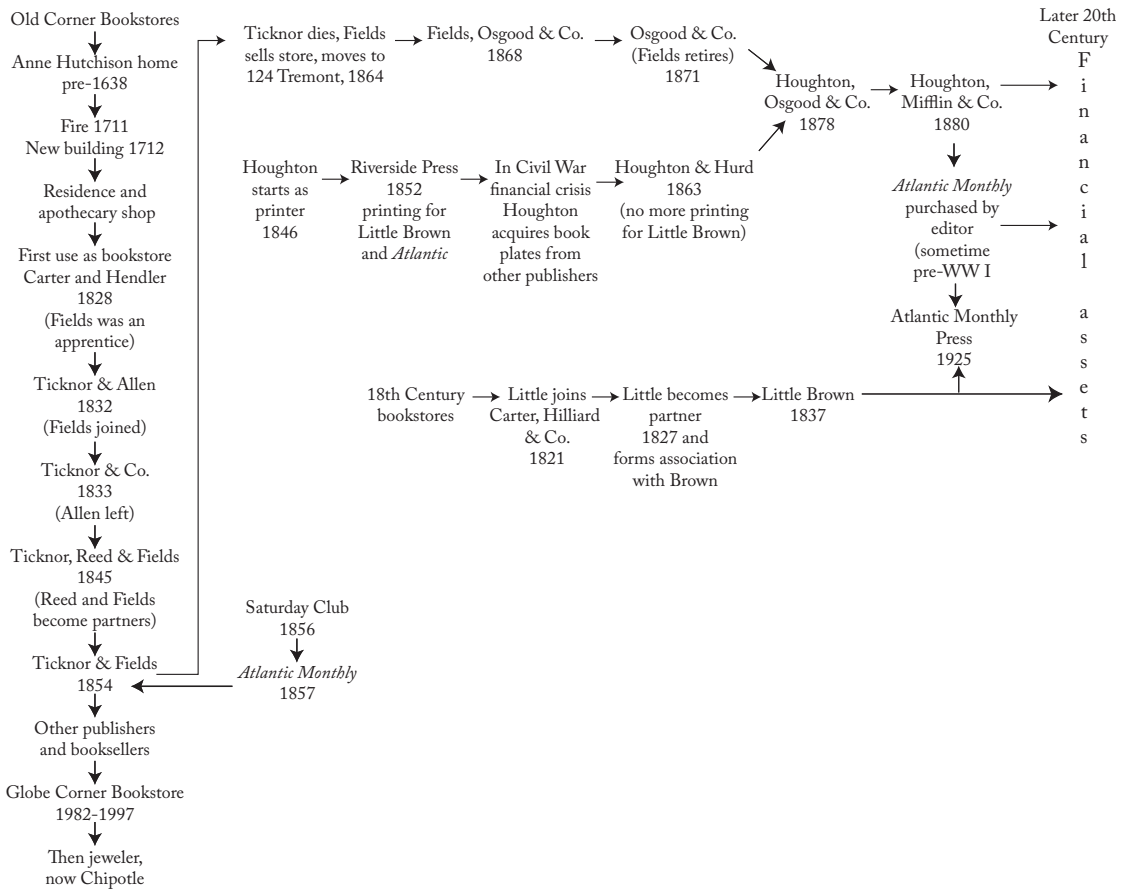


Figure 2: Some transitions in Boston publishing, from the middle years of the 19th century.

{37} Going down the left column of Figure 2, we have the history of the Old Corner Bookstore building, which in time became a building housing booksellers and publishers, including eventually the important publisher known as Ticknor and Fields. A little while after the *Atlantic Monthly* was established, Ticknor and Fields acquired the Old Corner Bookstore. Then, after Ticknor's death, Fields moved the publishing business and the *Atlantic* to Tremont Street.

Also shown in the figure, coming out of the 1700s, were the predecessor organizations to Little Brown.

Finally (also on the figure), Henry Houghton, just out of college, started work as a printer, eventually acquired his own business; and he established the Riverside Press in 1852 in Cambridge where he also did printing for Little Brown and later for the *Atlantic Monthly*. During the economic downturn resulting from the Civil War, Houghton acquired book plates from various failing publishers, and eventually went into publishing himself (with Melancthon Hurd as a New York partner), which caused Little Brown to drop Riverside Press as a printer.

In time the successor partners to Ticknor and Fields merged with Houghton's company, George Mifflin joined, and later, with Houghton growing old and Mifflin by that time a partner, the company became Houghton Mifflin.

In the early 1900s, the editors of the *Atlantic Monthly* bought the magazine from Houghton Mifflin.

From then until circa 1970, Houghton Mifflin and Little Brown were the "big two" Boston publishers, and the *Atlantic Monthly* was a Boston institution. At that point, these historic institutions increasing were important as financial assets to be bought and sold.

{38} Of course there were other publishers in Boston besides the Big Two, including specialty publishers. Two examples are the Beacon Press (1854–present) and Daniel Berkeley Updike's Merrymount Press (1893–1941).

The Beacon press was and is the publisher for the Unitarians, now UU, church. Its first location

was on Bromfield, the street parallel to School Street behind the hotel. It later moved to Washington Street, and then to Beacon Hill. Its current building is on Joy Street, a block from the Beacon Street headquarters of the UUs. In the 19th century it primarily printed sermons and other books related to Unitarian theology. Since the 20th century it has printed many non-religious books, consistent with its mission to publish works that "affirm and promote ... the inherent worth and dignity of every person; justice, equity and compassion in human relations; acceptance of one another; a free and responsible search for truth and meaning; the right of conscience and the use of the democratic process in society; the goal of the world community with peace, liberty, and justice for all; respect for the interdependent web of all existence; and the importance of literature and the arts in democratic life" [Beacon12].

{39} Daniel Berkeley Updike was a fine book publisher [Blumenthal89], who had previously gained experience for over a decade at Houghton Mifflin and its Riverside Press [Kelly11]. Updike also was greatly interested in the history of printing types, and in 1922 published his classic book, *Printing Types, Their History, Forms and Use: A Study in Survivals*. (Barbara Beeton has told me, in a 2012-06-21 email, that "Updike's archives from the Merrymount press are now in the special collections of the Providence Public Library; they include an incredible variety of specimen sheets and other material of interest to typographers and book designers.")

*

{40} There were lots of Boston papers in the 1800s and 1900s, for example: *Boston Daily Advertiser*, 1813; *Boston Journal*, 1833; *Boston Evening Traveler*, 1845; *Boston Herald*, 1846; *Boston Globe*, 1872; *Boston American*, 1904; plus smaller town papers. There were also lots of mergers and acquisitions, i.e., industry consolidation as the mid-1900s neared.

{41} Boston's newspaper row was on Washington Street, down the one-block length of School Street from the conference hotel. {42} In the days before radio and TV were common, crowds stood in Washington Street to hear the latest news, e.g., of an election or prize fight.

An alley off Washington Street, a short way from the School Street intersection, is named Pi Alley. Purportedly it is named Pi Alley because of all the newspapers in the area and the fact that a box of type dropped and scrambled on the floor was known as "pied type."

*

{43} In the colonial days, the printers typically had very little type — maybe one or two sizes, maybe not italic, maybe not two different typefaces. What type they did have came from Europe.

By 1800, Williams Caslon's types had made it to the new country in many variations. Isaiah Thomas's specimen book of types featured Caslon on its cover: "Being as large and complete an assortment as is to be met with in any one printing-office in America. Chiefly manufactured by that great artist William Caslon, Esq., of London" [Blumenthal89]. (As an homage to the popularity of Caslon type in the eras under discussion (for instance, it was also used for the Declaration of Independence), I used Adobe Caslon Pro for the text of my presentation slides and also have used it for Figure 2 of this article.)

Type foundries took a while to get going in the United States. The first successful one in Boston was the Dickinson type foundry, founded in 1839. There were a few other foundries by the time of the great Boston fire of 1872, in which all the type foundries were destroyed.

There were five type foundries in Boston by the time of the American Type Founders (ATF) consolidation of 1892: "the Dickinson Type Founders, Boston Type Foundry, New England Type Foundry, Curtis & Mitchell Type Foundry, and the H.C. Hansen Type Foundry. The H.C. Hansen Type Foundry was started in late 1872 after the fire (Hansen had been an employee of Dickinson). The New England and Curtis & Mitchell foundries soon disappeared. The Dickinson and Boston foundries were absorbed by ATF. H.C. Hansen, later with his sons, remained in existence until 1922 as an independent type foundry" [Devroye02].

No doubt a good bit of type design also went on in Boston. Two individuals well-known for their type designs were Bruce Rogers and Bertram Grosvenor Goodhue [Blumenthal89].

Goodhue (1869–1924) was a celebrated architect who also did book and type design, in particular the Merrymount type for the Merrymount press and Cheltenham type (originally known as Boston Old Style). Cheltenham was widely popular in the early decades of the twentieth century. Digital versions have been created in recent decades including a version for the New York Times designed by Matthew Carter.

{44,45} Rogers (1870–1957) is viewed by some as the greatest book designer of the 20th century. He worked at Riverside Press from 1895 to 1911 (he took over Updike's position [Kelly11]) where he created many fine editions. Two of the types he designed are Montaigne (at Riverside Press) and Centaur (for New York's Metropolitan Museum of Art). Centaur remains widely available (for example,

it comes with Microsoft and Adobe products). I particularly like the slanted hyphen of Rogers's original Centaur, which apparently and unfortunately is no longer slanted in the digital recreation. Rogers practiced so-called "allusive typography" where the type and ornamentation were matched to the content of the book.

Personal observations, 1964–present

{46} This last section, covering the period of time I have lived in Boston and adjacent towns, is about my personal experience and observations rather than trying to cover printing and publishing history.

As I mentioned at the start of this note, Boston seemed a literary mecca when I first arrived here in 1964. In Harvard Square, near where I originally lived, the Harvard Coop and the Paperback Booksmith were my primary bookstores, but there were a variety of other stores selling new books in the Harvard Square neighborhood. Also, there seemed to be dozens of used bookstores. The Out of Town News in the middle of Harvard Square and Nini's Corner across the street had lots of magazines and newspapers from around the world for sale. At the time Out of Town News had perhaps a hundred different foreign newspapers shipped in by air and would sell up to 600 London papers on Sundays and 1,500 Irish papers a week.

I had also not previously been where I could easily use the libraries of multiple communities. Most of the time since I arrived in Boston I have had library cards for three different library systems, e.g., originally Cambridge, Boston, and Belmont (on my way to and from work). Also, because I originally worked for MIT, I could use its libraries and occasionally found a reason and way to access Harvard libraries.

In my early years in the region, Boston also had many bookstores selling new books and even more used-book stores, it seemed, than in Cambridge.

{47} I also liked the major newspaper options: *Globe* (more liberal), *Herald* (more conservative), and *Christian Science Monitor* (a wonderful 6-day-a-week paper providing unbiased world news). Over time some interesting weekly newspapers were founded as well as the African-American *Bay State Banner*.

{48} As I got to know Boston, I also got to know the locations of such publishing institutions as the Riverside Press in Cambridge, {49} Little Brown at the corner of Joy and Beacon Streets on Beacon Hill facing the Boston Common, {50} the *Atlantic Monthly* on Arlington Street facing the Boston Public Garden, and {51} Houghton Mifflin which in 1972 moved into the high rise building diagonally across the intersection from the conference hotel.

*

Much has changed in the nearly 50 years I have lived in or around Boston (an average of 3.5 miles from the conference hotel and never more than 8.3 miles). At some point (circa 1980 perhaps) the big chains of bookstores began displacing the independent bookstores and small chains, for example, Barnes and Noble, Waldenbooks, Borders, and eventually Waterstones (from the UK) came to Boston.

A few years ago, Out of Town News stopped carrying most of its foreign newspapers.

As mentioned above, the big two publishers (Little Brown and Houghton Mifflin) have become substantially financial assets rather than institutions dedicated to the mission of publishing. The Riverside Press is gone from Cambridge, and the *Atlantic Monthly* is now part of a non-Boston publishing empire.

{52} A notable exception to the decline of publishing in Boston is "David R. Godine, Publisher." Godine started his publishing business printing his own books in nearby Brookline. Dedicated to publishing quality books, the business has slowly grown. When I first became aware of it, {53} it was based in the beautiful Victorian Ames-Webster mansion at the corner of Dartmouth Street and Commonwealth Avenue in Boston (it is now based about 400 feet from the conference hotel).

{54} Another exception is International Data Corporation, now part of the International Data Group. This company publishes business data and 300 magazines in 85 countries. It founded the "Dummies" series of books (later sold to Wiley). Its headquarters is across the street from the Boston Public Library in a high rise building (known locally as the Darth Vader building — not a compliment).

Of course, Godine and IDC are not the only positive notes, but they are an exception to the apparent general decline of publishing and printing in Boston. (During many of my years in Boston, Addison-Wesley, Knuth's TEX publisher was based just outside of Boston. I also now know that Bitstream was started in Cambridge; and Matthew Carter, a founder of Bitstream and now of Carter and Cone, lives in Cambridge. A few other type design activities are still based in Boston.)

As we have moved into the Internet era, the *Globe* and *Herald* newspapers were bought by out-of-town newspaper empires (the *Herald* has since become independent again but is struggling). The *Christian Science Monitor* has become an on-line newspaper. And most of the chain bookstores have succumbed to the competition of on-line bookstores such as Amazon and to the popularity of the e-book.

With our present American culture of "no new taxes," the city and town libraries have cut their hours.

Nonetheless, Boston still remains a major urban, educational, and cultural center with some pretty nifty literary resources, particularly its libraries. {55} Some notable libraries are at Harvard (founded 1636) libraries (with 80 libraries and 15 million books); the Boston Athenæum (founded 1807); the New England Historical and Genealogical Society (founded 1845); Boston Public Library (founded 1848, the first large city public library in the country including having a circulating library [Whitehill56], now with many branches and 15 million books); and MIT (founded 1865) libraries (divided into several sub-libraries with 3 million books).

In addition to being notable, these libraries are close enough together to require minimal travel time among them — so close together it is probably faster to take public transportation than to find parking. A walk passing each of them would only take about 90 minutes: one mile from the {56} Athenæum to the {57} NEHGS; two blocks from the NEHGS to the {59} BPL (the BLP was previously {58} on Tremont Street); 1.5 miles from the BPL to MIT (1 mile as the crow flies across the Charles River, or as the Tech dinghy sails), {60} whose original building was across the street from the current location of the NEHGS; and 1.7 miles from MIT to Harvard. {61}

The youngest of these libraries is almost 150 years old. So obviously it is possible for a literary institution to withstand and adapt to the evolution of culture and economics — I suspect because they have never viewed profit as a key aspect of their missions.

Furthermore, since 1964, accessibility to library materials has become even easier. There were always many town libraries and libraries at the dozens of other colleges and universities in and around Boston. Now lots of the libraries are in library networks, so one can ask for a book at any of the cooperating libraries to be sent to one's own library. Also, there is the Massachusetts Virtual Library which supports exchange of books among networks. There is so much exchange going on that there is a company which has made a business of vans driving among the various libraries and networks of libraries doing deliveries and returns of exchanged books.

I am sure much of this is no different than what is going on in other states in the United States and in other areas around the world. Still, being based near Boston is particularly convenient for using libraries for research projects.

{62} For bookstores, today Barnes and Noble is the only big chain still in Boston, with a store at the Prudential Center in Back Bay and operating some of the university and college bookstores. It is a little hard to find a general-purpose independent store focusing on new books in Boston. Commonwealth Books and Brattle Book Shop primarily sell used books, and are within easy walking distance of the conference hotel.

In Cambridge, the MIT Press bookstore and the MIT Coop bookstore, selling new books, are across the street from each other. Moving on to Cambridge's Central Square and then Harvard Square, there are still a few used bookstores (e.g., Rodney's). Selling new books, in Harvard Square there are still the Harvard Book Store (founded in 1932 and never a part of Harvard), Grolier Poetry Bookshop (1927), and Schoenhof's Foreign Books (1856). Harvard Book Store and Grolier are still independently owned, in each case by a relatively new owner concerned that an institution not go out of business. Schoenhof's is no longer independently owned.

{63} A particular favorite bookstore for me is the Harvard Book Store. It has no relationship to Harvard University except to have Harvard buildings on three sides of it. They have a large selection of new books and used books, and they offer bicycle delivery in Cambridge and nearby. {64} They also have an Espresso Book Machine (EBM), which they have named Paige M. Gutenborg, for in-store on-demand printing of a customer's self-published book or millions of legally printable books from Google books, publishers, and other on-list books archives. (The EBM is another publishing activity where Jason Epstein has been a prime mover. Epstein was previously editorial director at Random House for 40 years and personally edited many famous authors; he co-founded the *New York Review of Books*, founded the Library of America line of books, and published the *Reader's Catalog* in the mid-1980s to make 40,000 books available through a phone-call purchase.)

{65} A visit to the Harvard Book Store and their in-store book-printing machine brings us geographically full circle. {66} The Harvard Book Store is a three minute walk from the location of Stephen Daye's original 1639 print shop. {67} And by doing in-house printing, the bookstore has in some sense come full circle in the history of American printing and bookselling — back to its roots where booksellers often printed and published books and where printers sometimes had retail sales of books they printed.

Recently, I took the opportunity of visiting the Harvard Book Store and Paige M. Gutenborg to have a facsimile copy of the *Bay Psalms* printed for me. And that brings this talk full circle.

Acknowledgments

{68} Steve Peter gave me pointers to useful books. Jeff Mayersohn of Harvard Book Store pointed me to people and places, loaned me books, and gave me a print-on-demand copy of the *Bay Psalms*. The librarians at the Boston Athenæum and Boston Public Library helped me find books. Karl Berry caught many typographical errors and made other editorial suggestions. Barbara Beeton suggested some content additions and editorial changes. Jeffrey Stanett and Ryan Shea Paré of First Printer restaurant answered questions and allowed me to take photographs of their printing-history artifacts.

Bibliography and references

AAS12. American Antiquarian Society website, 2012: `http://www.americanantiquarian.org`.

Barry12. John M. Barry, *Roger Williams and the Creation of The American Soul: Church, State, and the Birth of Liberty*, Viking Press, New York, 2012.

Beacon12. Beacon Press website, 2012: `http://www.beacon.org`.

Blumenthal89. Joseph Blumenthal, *The Printed Book in America*, published for the Dartmouth College Library by University Press of New England, Hanover, NH, trade paperback edition, 1989 (reprint of the 1977 edition published by David R. Godine, Boston).

Drake70. Sanuel Adams Drake, *Old Landmarks and Historic Personages of Boston*, Singing Tree Press, 1970 (facsimile reprint of the edition published in 1900 by Little, Brown, and Company, Boston, which was the "new and revised" version of the 1974 original published by James R. Osgood and Company, Boston).

Devroye02. Luc Devroye website, 2002 (it's wonderful; look at it): `http://luc.devroye.org/hansen`.

Franklin40. *The autobiography of Benjamin Franklin*, with postscript by Richard B. Morris, Washington Square Press, New York, 1940.

Hall10. David D. Hall, general editor, *A History of the Book in America* (five volumes), The University of North Carolina Press, Chapel Hill, NC, 2010: Volume 1, *The Colonial Book in the Atlantic World*, edited by Hugh Amory and David D. Hall; Volume 2, *An Extensive Republic: Print, Culture, and Society in the New Nation, 1790–1840*, edited by Robert A. Gross and Mary Kelley; Volume 3, *The Industrial Book, 1840–1880*, edited by Scott E. Casper, Jeffrey D. Groves, Stephen W. Nissenbaum, and Michael Winship; Volume 4, *Print in Motion: The Expansion of Publishing and Reading in the United States, 1880–1940*, edited by Carl F. Kaestle and Janice A. Radway; Volume 5, *The Enduring Book: Print Culture in Postwar America*, edited by David Paul Nord, Joan Shelley Rubin, and Michael Schudson.

Isaacson04. Walter Isaacson, *Benjamin Franklin: An American Life*, Simon & Schuster Paperbacks, New York, 2004.

Kelly11. Jerry Kelly, *The Art of the Book in the Twentieth Century*, RIT Cary Graphic Arts Press, Rochester, NY, 2011.

Krieger99. Alex Krieger and David Cobb, editors, *Mapping Boston*, MIT Press, Cambridge, MA, 1999.

Lehmann-Haupt52. Hellmut Lehmann-Haupt (in collaboration with Lawrence C. Wroth and Rollo G. Silver), *The Book in America: A History of the Making and Selling of Books in the United States*, second edition, R. R. Bowker Company, New York, 1952.

Myerson80. Joel Myerson, *The New England Transcendentalists and the* Dial*: A History of the Magazine and Its Contributors*, Fairleigh Dickinson University Press, Rutherford, NJ.

Phillips06. Jerry Phillips and Andrew Ladd, *Romanticism and Transcendentalism (1800–1860)*, Facts on File, Inc., New York, 2006.

Reese89. William S. Reese, The First Hundred Years of Printing in British North America: Printers and Collectors, `http://www.reeseco.com/papers/first100.htm`.

Rose81. Anne C. Rose, *Transcendentalism as a Social Movement, 1830–1850*, Yale University Press, New Haven, CT, 1981.

Tebbel72, 75, 78, and 81. John Tebbel, *A History of Book Publishing in the United States*, R. R. Bowker, New York: Volume 1, *The Creation of an Industry*, 1972; Volume 2, *The Expansion of an Industry, 1865–1919*, 1975; Volume 3, *The Golden Age between Two Wars, 1920–1940*, 1978; Volume 4, *The Great Change, 1940–1980*, 1981.

Thomas70. Isaiah Thomas, *The History of Printing in America with a Biography of Printers & an Account of Newspapers*, M. M. McCorison, editor, Imprint Society, Barre, MA, 1970 (new edition based on the two volume 1874 edition published by the American Antiquarian Society, Worcester, MA, which was derived from the original 1810 book by Thomas).

Updike66. Daniel Berkeley Updike, *Printing Types, Their History, Forms, and Use: A Study in Survivals*, Volume II, The Belknap Press of Harvard University Press, Cambridge, MA, 1966.

Whitehill56. Walter Muir Whitehill, *Boston Public Library: A Centennial History*, Harvard University Press, Cambridge, MA, 1956.

Whitehill68. Walter Muir Whitehill, *Boston: A Topographical History*, second edition, enlarged, Belknap Press of the Harvard University Press, Cambridge, MA 1968.

Wilson00. Susan Wilson, *Literary Trail of Greater Boston: A tour of sites in Boston, Cambridge, and Concord*, Houghton Mifflin, Boston, 2000.

Wilson05. Leslie Perrin Wilson, "No Worthless Books": Elizabeth Peabody's Foreign Library, *The Papers of the Bibliographical Society of America*, Volume 99, Number 1, 2005, pp. 113–152.

◇ David Walden
`http://walden-family.com/texland`

Towards evidence-based typography: First results

Boris Veytsman and Leyla Akhmadeeva

At the previous TEX Users Group meeting we described a program to revisit the long-standing dicta of typography from the point of view of experimental science [9]. Indeed, any practitioner of the art "knows" that serif fonts are better for continuous body copy, while sans serif fonts are better for the texts intended to be read in chunks, like advertisements, that optimal line sizes are based on the physiology of eye movement, etc. However, many traditional views about human organisms are not confirmed by rigorous experiment: after all, a couple hundred years ago people also "knew" that profuse blood letting helps to cure a number of diseases including bubonic plague and the common cold [5]. Thus we started a long term program to study whether the traditional ideas of typography are confirmed by experiment.

Recently Legge and Bigelow studied readability and legibility of text with different font sizes [7] which led them to the so-called *ecological hypothesis*: the print sizes actually used over the centuries in book making are in the "comfort zone" for a normal vision reader, and variations in the size are of low importance. A natural generalization of this hypothesis is that other typographic devices like serifs or their absence do not matter much if they have been used in typography for a long time. We decided to test this generalized hypothesis.

In our experiments we asked the subject to read texts typeset differently and answer multiple choice questions about them. TEX allowed us to easily obtain high quality texts with controlled typographic features.

In the first series of tests we studied the influence of serifs on reading and comprehension. The experiments with serif and sans serif fonts described in the literature [2–4, 8] give controversial results: some studies show the advantage of serifs, while some indicate sans serif fonts are more legible. One should note that in these experiments (with the notable exception of [8]) the texts were typeset in different "superfamilies" (mostly Times and Helvetica), which may be a confounding factor: there are many differences between Times and Helvetica besides one being a serif font and another being a sans serif one. Therefore we chose for our experiments a pair of fonts from the same "superfamily": PT Serif and PT Sans by Paratype [6] (see Figure 1; note the difference in the alphabet width of the fonts due to the serifs). Their shapes are very close, and thus we can assume that the presence or absence of serifs is the main difference between the fonts. The experiments with Cyrillic readers ($n = 238$) showed (see Figures 2 and 3) that there was no statistically significant difference between serif and sans serif fonts — neither in the speed of reading nor in the number of correct answers about the text [1].[1] These results corroborate the generalized ecological hypothesis.

The quick brown fox jumps over the lazy dog.
01234567890 fi æ œ

The quick brown fox jumps over the lazy dog.
01234567890 fi æ œ

Figure 1: PT Serif and PT Sans fonts

Currently we are studying the influence of the line length (is it really necessary to do two-column typesetting in landscape orientation) and justification on reading speed and comprehension. Our preliminary results indicate that these factors also do not significantly influence the outcome.

We are glad to acknowledge the support of TUG and the Federal Program of Russian Federation "Scientists and Science Teachers for Innovation in Russia, 2009–2013". We are grateful to Karl Berry, Charles Bigelow and the participants of TUG 2011 & TUG 2012 for valuable suggestions. Last, but not least, we would like to acknowledge our wonderful students Ilnar Tukhvatulin, Kamilla Mufteeva, Alla Borisova and Aliya Habiryanova.

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] A. Arditi and J. Cho. Do serifs enhance or diminish text legibility? *Invest. Ophthalmol. Vis. Sci.*, 41(4, Suppl. S):S437, March 2000.

[3] A. Arditi and J. Cho. Serifs and font legibility. *Vision Res.*, 45(23):2926–2933, 2005.

[4] M. L. Bernard, B. S. Chaparro, M. M. Mills, and C.G. Halcomb. Comparing the effects of text size and format on the readibility of computer-displayed Times New Roman and Arial text. *Int. J. Hum.-Comput. Stud.*, 59(6):823–835, December 2003.

[5] Thomas Dover. *The Ancient Physician's Legacy to his Country: Being what he has collected*

[1] A preprint of the paper is available at `http://borisv.lk.net/publications/serif1`

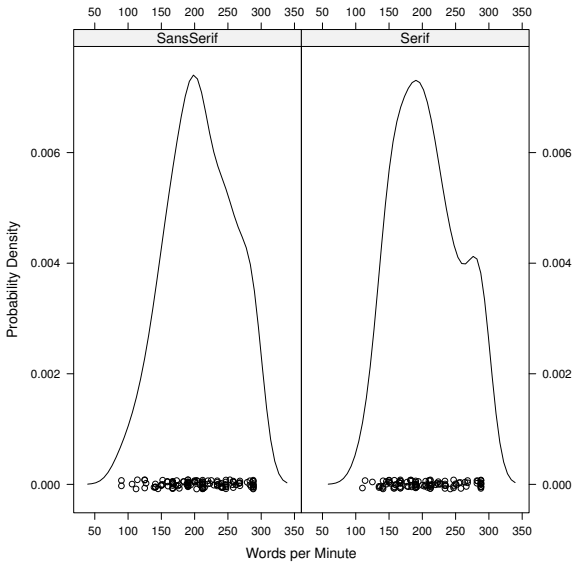


Figure 2: Reading speed for sans serif and serif fonts

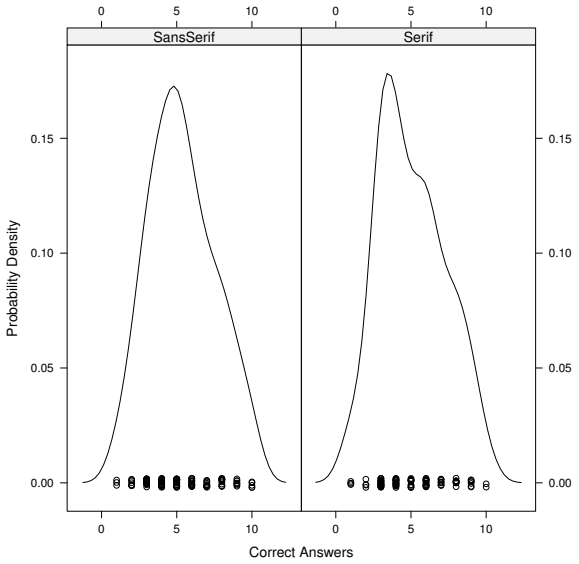


Figure 3: Reading comprehension for sans serif and serif fonts

himself, in Fifty-eight Years Practice: Or, an Account of the several Diseases incident to Mankind, Described in so plain a Manner, That any Person may know the Nature of his own Disease, Together with several Remedies for each Distemper, faithfully let down, Designed for the Use of all Private Families. Henry Kent, London, 1742.

[6] Pavel Farář. *Support Package for Free Fonts by ParaType*, May 2011. `http://mirrors.ctan.org/fonts/paratype`.

[7] 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.

[8] R. A. Morris, K. Aquilante, D. Yager, and C. Bigelow. Serifs slow RSVP reading at very small sizes but don't matter at larger sizes. In *SID 2002, San Jose, CA: Digest of Technical Papers*, pages 244–247. The Society for Information Display, 2002.

[9] Boris Veytsman and Leyla Akhmadeeva. Towards evidence-based typography: Literature review and experiment design. *TUGboat*, 32(3):285–288, 2011. `http://tug.org/TUGboat/tb32-3/tb102veytsman-typo.pdf`.

◇ Boris Veytsman
School of Systems Biology &
Computational Materials
Science Center, MS 6A2
George Mason University
Fairfax, VA 22030
`borisv (at) lk dot net`

◇ Leyla Akhmadeeva
Bashkir State Medical University
3 Lenina Str. Ufa 450000, Russia
`la (at) ufaneuro dot org`

TEX and music: An update on TEX*muse*

Federico Garcia

1 Introduction

For some years I've been working on a professional music typesetting system using TEX and METAFONT, called TEX*muse*. This article has some background and examples of the main idea of the TEX*muse* system which, as will be seen, is entirely inspired by the TEX 'spirit'. This touches on the potentially disastrous problems of WYSIWYM for music typesetting, and on what similarly oriented systems have done (and not). Some achievements of TEX*muse* could potentially change the life of any composer.

2 A sample of the current prototype

2.1 The user's input and the first scan

In the main input file of a TEX*muse* document the user defines a series of instruments (more accurately 'staves'), and then requests TEX*muse* to assemble a score out of a subset of them (possibly all of them):

```
\newstaff[\trebleclef\AMajor]
    {righthnd}{MztAMj.txm}
\newstaff[\bassclef\AMajor]
    {lefthnd}{MztAMj.txm}

\score: {righthnd,lefthnd}.
```

This convention makes it possible for the same document to produce both a general score (which includes all the instruments) and the individual instrumental parts; it also allows changing the order of the staves, replacing one by another, reusing the same music in several different documents, etc. — while keeping the actual musical content independent.

The `\score` command is (planned to be) flexible enough to allow several kinds of groupings between the staves. Piano music is typically typeset as two staves joined by a brace at the left of each musical line, and that is what the braces in `{righthnd,lefthnd}` mean. But a construction like `[oboe,clarinet]` would instead join the staves by a bracket, and the simple `flute, violin` would simply include the staves in the score without any particular grouping or joining shape.

The `\newstaff` command defines the staves, and instructs TEX*muse* where to look for the actual musical content of each staff. The actual musical content is typically collected in `txm` files, in the form of a series of staff environments, each named as the staves themselves. The file may contain staff environments even if they are not used in the document, in which case the program would simply ignore them.

In this case — an excerpt from Mozart's Sonata in A — the music for both staves is collected in the auxiliary file `MztAMj.txm`. The following is the environment for the staff '`righthnd`' as it appears in that file (the output appears in section 2.5):

```
\begin{righthnd}    \rangefrom{g4}\meter68
4c.3d4c 5e4e  4b.3c4b 5d4d  5a4a 5b4b
5c\split{3ed5c4b\\4b 5a4g}  4c.3d4c 5e4e
4b.3c4b 5d4d
5a4b 5c\split{4<bd>5<ac>4<bg>5<ae,>\\
      \,4f5e4d5c}
     4r
\end{righthnd}
```

This kind of input is not too different from the model used in commercial music software: numbers tell the program what rhythmic value to expect for the coming note(s): quarter notes are indicated by '`5`', eighth notes (smaller by one than the quarter note) by '`4`', sixteenths by '`3`', and so on.

Rests are issued with an '`r`', the characters `'` and `,` are used to shift between octaves, and `.` adds a dot to the previous note. Multi-note entries (i.e., individual notes that have several noteheads on the same stem — pitches sounding at the same time) are indicated by `<` and `>`, although there are *many* utilities and shorthands for different multi-note situations.[1]

The `\split` operation tells TEX*muse* that the music in the staff is to be split into two voices, one of which will be typeset above the other (but on the same staff). The user does not need to indicate which note on the upper voice corresponds to (i.e., goes on the same vertical axis as) which in the lower: TEX*muse* scans both voices and pairs them up as expected, according to their rhythmic durations.

In fact, this ability of TEX*muse* to align voices by rhythm is true not only of `\split`, but is indeed the mechanism that allows the user to input staff by staff. Apart from the convenience of keeping each staff independent of the others, horizontal input (i.e., staff by staff) is the natural way the musician thinks of the music. The opposite — requiring the user to figure out in advance the rhythmic/vertical correspondences — is as unnatural as it would be to typeset a paragraph word by word, starting with the first word on the top line, continuing with the word that would be below it on the

[1] For example, after `\dyads` it won't be necessary to wrap every set of two notes with `<` and `>`; `\coll` will make TEX add a note at the specified interval (like in `\coll8b`, which adds the lower octave); and so on.

second line, then the first word on the third line, and so on. It is this unfriendly requirement that makes MusiXTEX (the remarkably complete system developed in the 1990s under the leadership of the late Daniel Taupin) hopelessly useless for the typesetting of music beyond small snippets; it has in fact prompted the implementation over the years of several 'pre-processors', non-TEX utilities that translate horizontal input into the vertical arrays required by MusiXTEX.

TEX*muse* achieves this because its code, so to speak, teaches music to TEX, so that it is able to interpret the music of each staff and deduce their mutual rhythmic relationship (and therefore their vertical alignment). It will be seen in the coming sections where exactly this deduction happens. But this discussion introduces a more general and relevant issue concerning the use of TEX for typesetting music: beyond the question of pure ability to produce the score itself, what concessions would any such program require from the user? How far can we go in preventing the author/composer from having to stop thinking of music to help TEX with typographical decisions? This is an important question, because by its very nature TEX can very well fail at this; but it will be shown that it is here too that it can succeed and surpass the alternatives. This discussion is the main topic of section 3 below.

2.2 From `txm` to `tm`

After scanning the user's input in the `txm` file(s), TEX*muse*'s first step is to translate it into internal functions that govern the appearance of each note.

In the previous section we saw that the user typed '`4c.`' for the very first note of the `righthnd` staff. After TEX*muse*'s internal translation, this first note looks (to TEX, privately) much more involved:

```
\@nntr\@nhd1{c5}\@invbl2{c5}\@agdt\@stem
    \add@bm1\end@{432}\relax
```

To paraphrase:

`\@nntr`	For this coming *new entry*,
`\@nhd1`	a *notehead* of kind 1 (the familiar filled-in ellipse)
`{c5}`	on the fifth c on the keyboard,
`\@invbl2`	with a kind-2 *invisible accidental* (a ♯)
`{c5}`	on that same c,
`\@agdt`	with an *augmentation dot*,
`\@stem`	a *stem*,
`\add@bm1`	added to a *beam* of index 1,
`\end@`	and *ending* at position 432.

The idea of 'a dotted eighth note on c5' (which is what the user requested with '`4c.`') has been made fully explicit into discrete commands that order the actual notehead, the stem, etc. But not only that, in the process TEX has done a couple of other things as well:

- After `\end@432`, TEX knows that the current staff won't have a note until position 432 is reached by all other staves. If, say, the second staff has a note that ends at 288, then that staff will have its second note starting at 288, but the first staff (which already spans up to 432) will have nothing on that note. The process is a 'quantization' of the rhythmic space, in which the 256^{th}-note (a note with six flags) contains 9 units. The first note in our sample is a dotted eighth-note: 288 (32 times a 256^{th}-note) +144 (a dotted note is increased by half) = 432.
- TEX*muse* also deduced that this note should be part of a beam.[2] It has looked ahead for the coming notes and has realized that since those notes are also beamable rhythmic values and they are part of the same beat in the current meter, they should be beamed together.
- The note also has an invisible sharp sign. By 'invisible' TEX*muse* means that it doesn't have to be actually drawn, and the question might arise then why to include it at all. This is part of TEX*muse*'s 'spelling' mechanism, a component of its ability to interpret the musician's intentions and to render it correctly onto the typeset score. This mechanism and its importance will be addressed in more detail later (section 3.3).

  For this note, TEX*muse* has deduced that the user meant c♯ (since we're in A major), but it also realizes that there is no need to explicitly give the sharp in the actual score. In addition to 'invisible', accidentals can be 'rigid' (when they must technically be added to mean the correct note) and 'courtesy' (sometimes also called 'cautionary', when the context makes it desirable to add the clarification).

In this way, notes in the user's input have been converted into sequences of internal commands like the line listed above. These 'entries' are written, staff by staff, on auxiliary files named ⟨*staffname*⟩`.tm` (in this case, the entry listed above is the first entry in `righthnd.tm`). Those files will be read by TEX*muse* in the next step of the run, to gather the notes and write METAFONT programs to draw them.

[2] The *beam* is the familiar thick line that joins the stems of several notes when they are small rhythmic values (an eighth-note or smaller).

2.3 From TEX to METAFONT

With the information in the `tm` files, TEX can now proceed to write programs in METAFONT that will draw the notes into a font. The following is the program that TEX writes for the first note in our sample:

```
newchar(1);
    staff_1;
        notehead(1,C5);
        openbeam(1);
        augm_dot;
        stem;
    staff_2;
        upper;
        notehead(1,E4);
        stem;
        lower;
        notehead(1,H3);
        openbeam(1);
        augm_dot;
        stem;
    endcharat(432-0);
```

The first thing to note here is that this listing contains information about *both* staves. In fact, it is the `tm`→`mf` stage that gathers the notes, from all the staves, to make the composite 'characters' that include any note that belongs in each vertical axis across the score.

In this first note, the right hand (`staff_1` for METAFONT) is simple enough; but the left hand contains two noteheads, which are in addition separated as '`upper`' and '`lower`' sub-notes. This comes from a `\split` in the original user's input for the left hand (not shown above), and illustrates what a split ends up looking like in the METAFONT program.

The `\add@bm` ('add to a beam') command has generated an `openbeam` in the METAFONT program. Since this is the first note, there are no beams currently open, so `\add@bm` is interpreted as opening one. Future notes with `\add@bm` will be given an `addtobeam` instead, until a `\close@bm` is found and the beam closed.[3] On the other hand, the '432' is retained and it still signals the end of the entry in `endcharat(432-0)` — METAFONT will do the math and find 432. TEX had used the ending positions of the notes to figure out the relationships between staves; but now the same information (which ultimately encodes the rhythmic profile of the piece) will be used by METAFONT for the spacing of the music. (In music, the longer the rhythmic value the more space there is after a note; the space after each note is later stretched proportionally, if necessary, to reach the right margin.)

TEX*muse* now enters the drawing stage, reading the character programs with the music-drawing functions defined in its code. The user (or a batch file) runs `mf` on these METAFONT programs generated by TEX, and out of this a new font is created (or many, if there are more than 256 characters in the piece). TEX will then use the font to typeset the music — which, for TEX, is just regular text.

2.4 From METAFONT to TEX

But the new font does not contain every single symbol in the music. METAFONT does not really draw some pre-formed symbols that are part of a musical score; for example, any lettering or any numbers will be inserted by TEX (retaining user control on their font and appearance), as well as some musical symbols like the clefs that don't really change from instance to instance and would be inefficient to draw on the fly. TEX will insert those symbols at the appropriate positions in the score, but for that it needs METAFONT to tell it where those positions are.

This information is passed to TEX in the log file of the METAFONT run. This is an excerpt from it:

```
Line at measure 3
Next break after measure 6
\fi\leavevmode\iffalse
\fi\rlap{\hbox{\raise 7.0pt\hbox
    {\mae ?}\kern2.2pt
\raise-3.5pt\hbox
{\mse \char 146}}}\iffalse
\fi\rlap{\raise38.5pt\hbox{\mae \&\kern2.2pt
{\mse \char 146}}}\iffalse
[1]
\fi{\tmfont\char1}\iffalse 3.9 [2]
\fi{\tmfont\char2}\iffalse 1.5 [3]
```

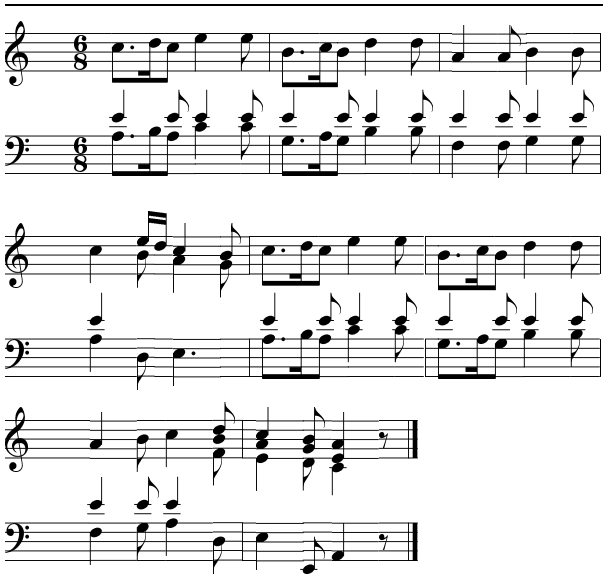
This is pretty low-level plain TEX, but you can still spot the `\char` commands that request particular symbols. `\mse` is TEX*muse*'s base musical symbols font ('muse'), in which character 146 is the key signature of A major. The clefs, at the time of writing, come from the TrueType font 'Maestro' (the default font for Finale, a mainstream commercial music program): `?` in that font is the bass clef, and `&` is the treble clef.[4]

[3] The reason for this two-step conversion is that the rests, that can also be included under a beam, behave differently. In other words, `\add@bm` is not *exactly* equivalent to `openbeam` or `addtobeam`, and `\close@bm` is not directly `closebeam`; in this particular case the difference is innocuous because no rests are involved.

[4] Eventually, Maestro will be entirely replaced by TEX*muse*'s own fonts. It has so far been used to permit implementation of the prototype even in the absence of a complete native font, and also to test TEX*muse*'s ability to use and interact with user-selected fonts.

`\tmfont` is the command that stands for the actual font that METAFONT has just drawn, and so `{\tmfont\char1}` is the exact point where METAFONT tells TEX to print the very first note of the piece. The log file continues requesting what goes in the score — the characters that it just drew or insertions from other fonts — and simply by reading the file TEX gets to typeset the music.

2.5 Sample output



TEX*muse* is still very incomplete. Apart from the fact that it doesn't yet connect the barlines across the two staves of the piano, much less join the staves with a brace on the left sides, the '$\frac{6}{8}$' (which as of now can't be scaled down...) appears slightly misplaced to the left, and the ' ' is misdrawn (it should give c♯, not b♯!).

But the skeleton is in place, and many of the missing features are more or less simply analogous to features already working in the current prototype. TEX can indeed interpret intuitive user input, and it can program METAFONT virtually without help from the user. The system is promising in terms of flexibility and programmability.

3 TEX and music: Why and why not

There are legitimate reasons, however, to be skeptical about music typesetting in TEX — or in fact in any system like TEX, with plain-text input, a compile phase, and fixed output.

3.1 Music, tables, and TEX

As mentioned, TEX*muse* (like the pre-processors for MusiXTEX) goes to a lot of trouble to make out the vertical correspondences implied by the user's horizontal output. The analogy above (typing a paragraph in vertical fashion) can be complemented by the realization that music, from the typographical point of view, is more like a table than it is like running text: vertical alignment by rhythm is as important, detailed, and meaningful as the horizontal dimension where a melody is presented.

In a way what the system does is *hide* the table from the user, in order to spare him the need to think of it explicitly. But this doesn't change the fact that the music is still a table... and tables are one context where the shortcomings of a TEX-like system are acute.[5] As any LATEX user knows, maintaining a table (editing it, updating it, correcting it) is not the most straight-forward intuitive process for us. In Excel, or even in a table in Word, you can simply click on a cell and start typing or deleting; in LATEX you have to count `\hline`'s and `&`'s even to find the cell. Searching is impeded, selecting a portion of the table for copying or pasting is impossible, ... TEX is simply not a natural environment for tables the way Excel is.[6]

There is therefore a big built-in inconvenience in the use of TEX, or any such system, for music typesetting. For me, a conclusion has emerged, as work on TEX*muse* has progressed, that the advantages that do come naturally with TEX — programmability, for example, or *precisely* targeted control where user decisions are documented, visible, and accessible as part of the input — are, by themselves, not nearly enough to compensate for that inconvenience. In other words, that TEX*muse*, if it was only a matter of those advantages, would unfortunately *not* be the best choice for a musician's music typesetting needs. Much more than that is needed for a system based on plain-text input to compete with WYSIWYGs, no matter how deficient the latter are... (and they are).

3.2 WYSIWYM in music?

The reference in the previous paragraph is to the needs *of a musician*, as distinct from the needs or concerns of an engraver proper. A musician does not need to know, for example, the rules for positioning beams, any more than a mathematician needs

[5] As Frank Mittelbach pointed out in another talk at this conference, (LA)TEX has severe limitations in this area (for example, there's no way to make a cell span several cells *both* horizontally and vertically at the same time). The problem referred to here is even more immediate and pressing.

[6] Of course the problem is not only TEX's: the same is true of HTML, for example, or any system that takes what is inevitably one-dimensional input to manipulate what is inevitably two-dimensional output.

to know about the rules governing the positioning of superscripts. The real test for TEX*muse*, or any system that aspires to being a good choice for musicians (composers, analysts, etc.), is whether or not it assists the *musical* mind without imposing concerns that are more typographical, or, even worse, syntactical, than musical.

For example: when we are in, say, c minor, every e is flat by default — that's part of what 'being in c minor' means. So, in c minor, the musician will write a scale simply as c-d-e-f-g, even though the e is actually e♭. Requiring the musician to explicitly request e♭ is forcing the mind to step outside a purely musical train of thought: think of a mathematician having to request explicitly the italics for the letters in the equations. From the typographical point of view, this is correct: the copyist or engraver, in music or in math, does indeed go through the extra step. But from the 'semantic' point of view, so to speak, of someone who is writing down a thought, this gets in the way. In a sense, those kinds of extra steps, foreign to the train of thought, are the whole point of using a computer — rule-based, mechanical, in principle automatic: isn't that what the computer exists for?

It is a priority that the input the user is required to learn and apply when typing music be as intuitive as possible: as close as possible to the actual steps and motions of writing the music by hand. In the c minor example, e♭ should be simply typed as e (just as you would write it on paper), the flat being silently deduced by the engine. It's not a matter of '*what you see is what you mean*', because we do mean e♭ although we don't write it out (or see it) in full. Rather than WYSIWYM, the paradigm is more one of '*what you see is what you'd write*'.

Or even more to the point: it's a '*dear computer, you know what I mean!*' kind of thing. Say there's a passage where every entry will consist of two notes — well, the user should be able to say `\dyads` and not worry about grouping every pair of notes with whatever syntax; if an entry will be repeated the same way (which happens quite often in music), shouldn't we have a shortcut, like using %,[7] so that we can type '`<ceg>%%%`' and obtain the same result as if typing '`<ceg><ceg><ceg><ceg>`'? Not that we want the %-like symbol to be output — we want to do less typing, and our meaning should be clear to the computer.

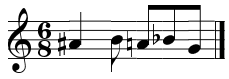
TEX*muse*'s command `\coll`, similarly, permits constructions like `\coll8b{cdef}`, meaning to add a lower octave ('*8va bassa*') to each note — a trick composers have used in their manuscripts for the copyists to add the extra note.[8]

And so on. These are very simple tools to program, and they follow naturally from admitting that horizontal input is, well, a necessary evil, that needs to be alleviated as much as possible.

The point, however, is that even with a successful syntax — one that helps rather than infuriates the user — even then it is legitimate to fear that TEX*muse*, or any system like it, is not a real alternative for use by musicians... the successful syntax just means that entering the music won't be more annoying than entering it in a WYSIWYG; but there's still a long way to go in order to overcome the structural inconvenience pointed out above, of not having direct, mouse-click access to each note in the score.

3.3 Tipping the balance

Consider the following snippet:



The three 'accidentals' in this example (the sharp ♯, the natural ♮, and the flat ♭) are all necessary parts of the pitch specifications; in particular, the second one cancels the effect of the first one (since in principle accidentals carry through full measures and therefore the original sharp on the a would apply to the second a as well).

Next, let's observe that the note with the ♯ and the one with the ♭ are in fact the same pitch (i.e., the same key on the piano). The musical spelling in the example (the particular choice of a♯ for the first one and b♭ for the second) is the correct one musically: since sharps are generally 'felt' to point up while flats tend to point down, the chosen spelling makes the notes actually reflect graphically the contour of the melody — a fact that, apart from looking better, is important semantically as well. Context matters, however: if the note following the b♭ was another b (natural), then the b♭ should be spelled as a♯ (a better way of leading to b); but if the last note was c, then b♭ would again be the preferred spelling (even though that flat would lead upwards in that particular case).

Most music typesetting programs today allow for 'MIDI input', meaning that they can take pitches in as keystrokes on a musical keyboard plugged into

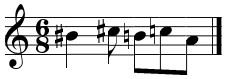
[7] A sign resembling the % is widely used informally for musical repetitions.

[8] Italian *coll* is an archaic contraction of *con il*, which was used since early times in instructions like "*coll violino I*" ("with the first violin"). The engraver would then copy the same music that was in the first violin. "*Coll*" was later generalized to other shorthand uses.

the computer. But they don't have a dynamic interpreter that would spell the notes correctly. You can (with more or less work) set each particular key on the piano to be spelled one way or another, but that isn't good enough: in our example, the two occurrences of a♯/b♭ would be spelled *both* as either b♭ or as a♯, and then you'd have to go back and correct them.

This is actually only a minor inconvenience, in part because it's only one small side of a much bigger annoyance. The 'dynamic pitch interpreter' that is lacking here would not only be able to make musical sense of pitch keystrokes, but, much more importantly, it would free the user from making spelling decisions for each context. It's not simply that the second-to-last note in the example should be spelled differently according to what note follows — the passage can change context in innumerable ways that would affect spelling:

- Say a different instrument will join the melody only starting in the middle: then it doesn't need the ♮, since there's no ♯ to cancel. A copy-paste operation in many a program today would incorrectly include the ♮.
- Say the composer decided to split the one measure into two (changing the time signature): then the ♮ is not technically needed, since a new measure cancels any previous accidentals. But in most cases it is a good idea, for clarity's sake, to put in the accidental anyway (the so-called 'courtesy' accidental). Many programs today would simply remove the accidental altogether, following the technical rule, and once the composer requests that the accidental be shown anyway, that decision will apply in any further use of the music, regardless of whether the new context requires it or not; and ten the courtesy accidental will create more, not less confusion...
- Say this line is for clarinet, and since the clarinet is a transposing instrument, you'll have to transpose it up a whole-tone for the clarinetist to read. In programs today, you'll get:



which is completely unacceptable: b♯–c♯ should have been spelled c–d♭.[9]

Because of all these potential problems, experienced music typesetters know that the process of setting a score includes a special step dedicated solely to proof-read the spelling. Even so, this is the area of music typesetting that is *by far* responsible for most typographical errors found in scores, new or old.[10]

This is a *huge* issue indeed — the kind of issue that, if solved, would make a program so useful above and beyond WYSIWYGs that it may well start to compensate for the inconveniences of plain-text input. TEX*muse* has a prototype of a spelling mechanism that does just this. (And has prompted several colleagues of mine to ask me regularly, but especially after struggling to meet deadlines, how close I am to completing the program so that they can use it!)

Less ground-breaking but similarly welcome is TEX*muse*'s system of beaming automation that also takes context in mind. This is not as significant in terms of a score's correctness, but it does separate amateur from professional musical engraving.[11]

And another important advantage of TEX*muse*: by deciding on line-breaking and page-layout — this latter not fully implemented yet — once again according to context and without user intervention, TEX*muse* will simplify the handling of the multiple looks a score takes. Any piece for more than one instrument consists not only of the general score (with every instrument in it), but also of the individual parts (the flutist gets the flute music only, the oboist the oboe music only, and so on). If, as in commercial software today, user intervention is needed to lay the pages out, then that means that the user will be helping out not with one document, but with a multitude of them. Spelling creates the most typos in scores, but it is this process, 'part extraction', that consumes most of a composer's time these days.

You could say that WYSIWYGs could just add these functionalities to their engines, and that is true to a certain extent. But the fact is that these utilities are actually more at home in an input-compile-output model than in a GUI. They all depend heavily on

[9] The problem arises from the fact that the correct b♭ transposition is not necessarily always a 'major second,' but may actually have to be a 'diminished third' in some portions; the two things are equivalent in terms of pitch — key on the piano — but not in terms of note on the staff. The program's downfall is again the lack of a dynamic, context-sensitive pitch-to-note interpreter.

[10] I saw an exercise in a recent conducting class where students were given one page from a score by Debussy where they had to find five typos (five!). *All* the errors (in a score that is close to 100 years old) were spelling mistakes.

[11] The paradigmatic case is a 3/4 time signature, where beams span across the three beats if there are only eighth-notes, but should break into beats in the presence of sixteenth-notes or smaller rhythms.

context, and on the context of the whole piece at that. With a GUI, automation would mean that editing (by the user) would create on-the-fly changes (by the program) that could easily get the user lost. A spelling mechanism would make notes dance from spelling to spelling after each new note was input; every copy-paste would necessitate the computer going over the full score; every new bar would potentially make you jump to a new line or even a new page (have you seen Word trying to deal with orphans and widows?). TEX, on the other hand, is a parsing, transformation language that operates on an input stream: exactly the right environment to implement the spelling of a melody and the beaming of a rhythm.

To sum up: by (a) keeping the inconveniences of plain-text input as minor as possible, and (b) implementing automation and interpretation mechanisms far beyond present applications, TEX*muse* is indeed promising as a real alternative for high-quality, sophisticated, and musical music typesetting.

4 Future directions

The presentation of the current state of TEX*muse* and of music typesetting in general at the 2012 TUG meeting generated excellent comments. Two major areas of feedback are particularly significant, and I would like to mention them as future lines of work.

4.1 TEX*muse* and LilyPond

An existing program has been conspicuously absent from the foregoing — the music typesetting system LilyPond. Also plain-text-compile-fixed-output, it stemmed from work on a preprocessor for MusiXTEX, MPP, whose author (Jan Nieuwenhuizen) was concerned not only about providing horizontal input capabilities, but also about the relatively poor quality of MusiXTEX's output. In the latter part of the 1990s, Jan and Han-Wen Nienhuys abandoned MPP and started work on LilyPond — now a non-TEX application, but based on TEX (and LATEX) for inspiration and approach.

LilyPond today is a complete working system, well known, with excellent output and an active community of both users and developers. With a much more reasonable input system than MusiXTEX's, it still fails in meeting the need for input that is intuitive to a musician—section 3.2 above makes in fact constant if veiled reference to LilyPond's input model, attempting an explanation to the observed fact that most musicians who try it find it unsatisfactory. (Mentioning LilyPond directly would have been unfair, since on the one hand the points are of general scope, and on the other these 'complaints' are not all there is to LilyPond or all I think of it.[12])

The fact is that LilyPond has essentially solved the problem of graphically realizing a musical score that has been encoded in some kind of plain-text syntax. If, on the other hand, TEX*muse* is able to interpret intuitive input — with a syntax tailored to composers and allegedly amenable to them — and from it write METAFONT programs to draw the music, could it not write LilyPond programs instead? That way we don't have to build METAFONT's musical engine from the ground up.

This is an excellent point and working on this is the most reasonable path to getting TEX*muse* to any degree of completeness in a reasonably short time. Even if the METAFONT macros will eventually be fully developed (so we keep the whole process within the TEX installation), a 'lilytex' package where TEX*muse* uses LilyPond (instead of METAFONT) is certainly high on the agenda of TEX*muse*'s development.

4.2 LATEX3

Some of the modules of TEX*muse*, including the signature automation capabilities, are general-purpose computer programming, and in developing them in TEX some of the well-known 'features' of TEX have made their presence felt. `\expandafter` must be among the top five command names in occurrence in TEX*muse*'s code. With the 'coming of age' of LATEX3 and the update about it at TUG 2012 (by Frank Mittelbach and Will Robertson), a new line for future work on TEX*muse* suggests itself: the conversion to LATEX3's programming environment expl3. Individual, self-contained modules (the spelling mechanism, for example) will be translated into the new language, as a way to test the algorithms, have a taste of expl3, and make TEX*muse* an active part of the most current developments in the TEX world.

◇ Federico Garcia
Artistic Director
Alia Musica Pittsburgh
federook (at) gmail dot com
http://www.fedegarcia.net

[12] In particular, LilyPond has full beaming automation (see note 11).

LaTeX3 News

Issue 8, July 2012

Extended floating point support

Bruno Le Floch has been re-writing the floating point module to function in an 'expandable' manner. This allows floating point calculations to be computed far more flexibly and efficiently than before. The expandable nature of the new code allows its use inside operations such as writing to external files, for which previously any such calculations would have to be pre-calculated before any of the writing operations began. Bruno's work on the floating point module has been officially released into the main SVN repository for `l3kernel`; TeX Live 2012 will contain the 'old' code for stability while this year is spent testing the new code in production environments and ironing out any wrinkles. Here's a neat example as suggested in the documentation, which produces '6.2784×10^2':

```
\usepackage{xparse, siunitx}
\ExplSyntaxOn
\NewDocumentCommand { \calcnum } { m }
  { \num { \fp_to_scientific:n {#1} } }
\ExplSyntaxOff

\calcnum {
  round ( 200 pi * sin ( 2.3 ^ 5 ) , 2 )
}
```

This feature is invaluable for simple (and not-so-simple) calculations in document and package authoring, and has been lacking a robust solution for many years. While LuaLaTeX can perform similar tasks within its Lua environment, the floating point support is written using the `expl3` programming language only, and is thus available in pdfLaTeX and XeLaTeX as well.

Regular expressions in TeX

As if expandable floating point support wasn't enough, Bruno has also written a complete regular expression engine in `expl3` as well. Many reading this will be familiar with the quote attributed to Jamie Zawinski:

> *Some people, when confronted with a problem, think "I know, I'll use regular expressions." Now they have two problems.*

And as humorous as the saying is, it's still fair to say that regular expressions are a great tool for manipulating streams of text. We desperately hope that people will *not* start using the regex code to do things like parse XML documents; however, for general search–replace duties, there's never been anything like `l3regex` for the LaTeX world. As a trivial example, there are 23 instances of the word 'We' or 'we' in this document (including those two). This value is counted automatically in two lines of code.

And again, it is available for pdfLaTeX and XeLaTeX users as well as LuaLaTeX ones; it also bears noting that this provides an easy solution for applying regular expression processing to LaTeX documents and text data even on the Windows operating system that does not have native support for such things.

Separating internal and external code

LaTeX packages are written by a wide range of package authors and consist of code and commands for various purposes. Some of these commands will be intended for use by document authors (such as `\pbox` from the `pbox` package); others are intended for use by other package writers (such as `\@ifmtarg` from the `ifmtarg` package). However, a large portion of them are internal, i.e., are intended to be used only within the package or within the LaTeX kernel and should not be used elsewhere. For example, `\@float` is the LaTeX kernel interface for floats to be used in class files, but the actual work is done by a command called `\@xfloat` which should not be used directly. Unfortunately the LaTeX 2_ε language makes no clear distinction between the two, so it is tempting for programmers to directly call the latter to save some "unnecessary" parsing activity.

The downside of this is that the "internal" commands suddenly act as interfaces and a reimplementation or fix in any of them would then break add-on packages as they rely on internal behavior. Over the course of the last twenty years probably 80% of such "internal" commands have found their way into one or another package. The consequences of this is that nowadays it is next to impossible to change anything in the LaTeX 2_ε kernel (even if it is clearly just an internal command) without breaking something.

In LaTeX3 we hope to improve this situation drastically by clearly separating public interfaces (that extension packages can use and rely on) and private functions and variables (that should not appear outside of their module). There is (nearly) no way to enforce this without severe computing overhead, so we implement it only through a naming convention, and some support

mechanisms. However, we think that this naming convention is easy to understand and to follow, so that we are confident that this will be adopted and provides the desired results.

Naming convention for internals

We've been throwing around some ideas for this for a number of years but nothing quite fit; the issue comes down to the fact that TEX does not have a 'namespacing' mechanism so any internal command needs to have a specific prefix to avoid clashing with other packages' commands. The prefix we have finally decided on for expl3 code is a double underscore, such that functions like `\seq_count:N` are intended for external use and `\__seq_item:n` is an internal command that should not be used or relied upon by others.
All this is well and good, but it can be inconvenient to type long prefixes such as `\__seq_` before all command names, especially in a package for which nearly *all* package functions are internal.
We therefore also extended DocStrip slightly by adding a 'shorthand' for internal package prefixes. Commands and variables in `.dtx` code may now contain `@@` which is expanded to the function prefix when the `.sty` file is extracted. As an example, writing

```
%<@@=seq>
\cs_new:Npn \@@_item:n
  ...
```

is equivalent to

```
\cs_new:Npn \__seq_item:n
  ...
```

There are clear advantages to this syntax. Function names are shorter and therefore easier to type, and code can still be prototyped using the `@@` syntax (e.g., pasting code between a `.dtx` file and a regular `.tex` document). Most importantly, it is explicitly clear from the code source which commands are intended to be used externally and which should be avoided.
We hope that this syntax will prove popular; in our initial experiments we think it works very well. In fact we found a good number of smaller errors when being forced to think about what is internal and what is an external function.

Continual revolution — the 'small bang'

In addition to the major additions introduced above, Frank Mittelbach has been examining `expl3` with a fresh eye to resolve any outstanding issues in the consistency or logic of the names of functions.
We are very mindful of the fact that for people to find `expl3` a useful tool, it must have a stable interface. This said, there are still some musty corners that we can show where people simply haven't been using certain functions. In select cases, we're re-assessing whether all of the (sometimes esoteric) odds and ends that have been added to `expl3` really belong; in other cases, it's now clear that some naming or behaviour choices weren't correct the first time around.
To address this tarnish, we're carefully making some minor changes to parts of the `expl3` interface and we'd like to allay any fears that `expl3` isn't stable. The `expl3` language now offers a wide range of functions plus their variants, and we're talking about changing but a very small percentage of these, and not common ones at that. We don't want it to become a mess, so we think it's better to tidy things up as we go. Follow the `LaTeX-L` mailing list for such details as they arise.

Preparing your thesis in LaTeX

David S. Latchman

Abstract

The submission of a thesis or dissertation is the culmination of many a graduate student's career. Given the time and effort toward research and attaining their degrees, this can often be a stressful time for many students. LaTeX offers the advantage of separating form from content and as the typical university thesis class can take care of a university's formatting requirements, this makes a student's life easier — well, at least it is supposed to.

Unfortunately, some formatting 'blends' into content thereby adding to the stress of an already unpleasant task. But there is some light at the end of the tunnel. With some preparation, typesetting a thesis in LaTeX can be relatively pain-free. But it's not just a matter of knowing what packages but *how* to use them and what is needed to use them effectively. Topics covered will include the typesetting of equations — both mathematical and chemical — as well as the proper formatting of tables and bibliographies.

1 The graduate student and LaTeX

As a TeX consultant, I often work with graduate students to help them get their theses and dissertations ready for final submission. For many graduate students, this marks their first foray into LaTeX and can be, for some, a most stressful experience; the emotional roller coaster ride of dealing with a review board notwithstanding. In most cases, the writing and editing process begins in Microsoft Word and continues there to the final editing and proofreading stages, where they are now getting ready to convert their thesis to LaTeX and eventual submission. There are many reasons why LaTeX isn't used at the start of the writing process. Some academic advisers and students are simply unfamiliar with LaTeX. It can also be difficult to annotate and track changes in LaTeX.

While many universities have their own thesis and dissertation class files to automate formatting of the document itself, class files intentionally fall short in many ways. Such class files are also designed to meet the needs of as many students as possible. Students can then add packages in the preamble depending on their personal needs and requirements. Applied correctly, the right package can help a student stay focused on writing and explaining themselves and not on how to format something.

Typically, a student turns to a consultant when they are unable to compile their document and help is not available. There might be that lone person in the lab or research group who knows LaTeX but who might not have the time or may be unwilling to put the effort into correcting errors. Moreover, consulting with students isn't just about fixing errors. After a project is completed, students are going to continue to make edits. Thus fixing any problems also means educating clients on how they can do things better and make their lives easier. Some of the packages and techniques discussed in this article are based on past projects I have worked on.

2 A theology dissertation

2.1 Language-specific packages

As a science graduate, I often find humanities projects interesting as I infrequently have the opportunity to work in the area. One recent project was a theology thesis that needed to be converted from Word to LaTeX. By itself, that would have been an interesting thesis to read — but since it was in German, and I don't speak a word of the language I could focus only on the technical aspects of the project.

LaTeX is structured for academic writing. While authors generally attempt to divide their writing into chapters and sections in Word, they don't always format their document properly by using the appropriate headings. This isn't necessary only for Word to create a table of contents but for any program to convert a Word document to LaTeX. Once the appropriate headings are set, conversion can take place.

While most universities have very strict formatting guidelines, in the case of this project there were none. Some students think of that as a positive, as it allows the freedom to design and format a thesis to look as they would like. While that may be true to some extent, we need to follow some typographic rules and formatting and not ignore them completely. I decided on the use of the KOMA-Script package as it follows European rules for typography and defaults to ISO A4 sized paper [1]. After an appropriate cover was designed, I focused on typography.

The KOMA-script package has book, article and report classes. The `DIVcalc` option for the `scrbook` class allows for the automatic calculation of both the page margins and text area based on font size. While it may seem lazy to make the package do these calculations, it removes the temptation for users to interfere with the settings of their `.tex` file. This can be done by specifying the options in the `\documentclass`:

```
\documentclass[DIVcalc, 11pt]{scrbook}
```

As the thesis is in German, the `babel` package was also needed [2]:

```
\usepackage[german]{babel}
```

This allows LaTeX to follow German-specific typesetting rules and ensure proper hyphenation.

The next step is to allow my client to type their thesis as easily as possible using the text editor of their choice. Generally, native English writers don't have to contend with accented characters and may forget that they even exist. If we do need them LaTeX has various commands to produce them. This is fine for the occasional accented character but it would be tedious if one is writing an entire report in a non-English language. As most text editors and word processors encode text in UTF-8, which includes characters with accents, we can use the `inputenc` package in the document's preamble to allow an author to write and edit their thesis as naturally as possible [3]:

```
\usepackage[utf8]{inputenc}
```

We also need to load fonts that will support accented characters found in European languages:

```
\usepackage[T1]{fontenc}
```

With the inclusion of the above packages, a student can type and edit the thesis with their favorite text editor.

2.2 Bibliographies

One of the strengths of LaTeX is its ability to handle bibliographic data using BibTeX [4]. English users generally experience no problems compiling their documents to produce a bibliography, but non-English language users are not always so fortunate. BibTeX was first released in 1988 and predates the advent of Unicode. Thus, it does not support files encoded in UTF-8, the default file encoding on most modern OSes. While it is possible to recode a `.bib` file into something that BibTeX can understand, this may be a complex task for a non-computer-savvy first-time LaTeX user. A solution should always allow someone to compile their thesis with little to no effort.

LaTeX does offer some solutions to this problem; the one that stands out for me is the BibLaTeX package. BibLaTeX has several advantages over BibTeX. For one, it can handle files that are encoded in UTF-8. While BibLaTeX has suffered from a lack of bibliographic styles in the past, this has changed in recent times. For German humanities students, we now have the `biblatex-fiwi` package [5]:

```
\usepackage[style=fiwi]{biblatex}
```

Using the BibLaTeX package is slightly different from using BibTeX. We still need the `.bib` file where the bibliographic data is stored:

```
@BOOK{KR,
   AUTHOR = "Kernighan, Brian W. and Ritchie, Dennis
        M.",
   TITLE  = "{The C Programming Language Second
        Edition}",
   PUBLISHER = "Prentice-Hall, Inc.",
   YEAR = 1988
 }
```

To incorporate a bibliography with BibLaTeX:

```
\documentclass[DIVcalc, 11pt]{scrbook}
\usepackage[german]{babel}
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
\usepackage[style=fiwi]{biblatex}
\bibliography{refs}
\begin{document}
All works should be properly cited~\cite{KR}.

\printbibliography
\end{document}
```

In BibTeX, the `\bibliography` command serves two purposes: it controls where the bibliography appears in the text and specifies the file containing the bibliography information. With BibLaTeX, however, the `\bibliography` command just reads the file and appears before the `\begin{document}` environment while the position of the `\printbibliography` command controls where the bibliography appears.

BibLaTeX also makes compiling the thesis easier. With BibTeX, LaTeX (here, `pdflatex`) has to be run twice after `bibtex`:

```
pdflatex mythesis.tex
bibtex mythesis.aux
pdflatex mythesis.tex
pdflatex mythesis.tex
```

With BibLaTeX, it only needs to be run once:

```
pdflatex mythesis.tex
bibtex mythesis.aux
pdflatex mythesis.tex
```

3 Publication-quality tables

Proper formatting of tables is another task I look at. Tables feature mainly in the sciences and engineering and may be infrequently seen in the humanities. Nonetheless, proper formatting is important. A properly formatted table organizes and displays data in rows and columns and should make it easy to look up values.

For many people, the standard practice is to design tables in Excel or some other spreadsheet program and translate this into LaTeX. Unfortunately, this often leads to poor table design. It may come as a surprise to some, but there are rules for proper table design which have been established for centuries [6]. Proper table design can be an art in itself, but the fundamentals are easy to follow. They are:

Parameter (Units)	Estimate (% RSE)[1]
V_1 (mL/kg)	48.6 (18.0)
K_{10} (1/day)	232 (5.51)
K_{12} (1/day)	3.76 (15.3)
K_{21} (1/day)	10.8 (28.4)
F_1	0.242 (4710)
F_{Total}	0.339 (15.0)
K_a (1/day)	18.9 (11800)
K_{aa} (1/day)	19.6 (164)

[1] Relative standard error of estimate (standard error divided by estimate), presented as a percentage

(a) Standard LaTeX table

Parameter (Units)	Estimate (% RSE)[1]
V_1 (mL/kg)	48.6 (18.0)
K_{10} (1/day)	232 (5.51)
K_{12} (1/day)	3.76 (15.3)
K_{21} (1/day)	10.8 (28.4)
F_1	0.242 (4710)
F_{Total}	0.339 (15.0)
K_a (1/day)	18.9 (11800)
K_{aa} (1/day)	19.6 (164)

[1] Relative standard error of estimate (standard error divided by estimate), presented as a percentage

(b) Table using the `booktabs` package

Figure 1: A table using LaTeX vs. `booktabs`.

1. Avoid use of vertical lines,
2. Avoid double horizontal lines,
3. Always place the units in the heading of the table instead of the body, and
4. Do not use quotation marks to repeat the content of cells.

Once a table has been edited to conform to the above rules, I include the `booktabs` package [7].

```
\usepackage{booktabs}
```

This package increases the vertical spacing between lines, increasing the white space in the table which allows data to be found and read easily.

We can see the differences between the two tables. The bottom table has more space between lines. This improves readability as the eye can follow a line more easily. This is an easy fix that makes a thesis look noticeably better.

4 Breaking equations

Sometimes students in the sciences and engineering are faced with the challenge of dealing with equations that span more than one line. The collection of $\mathcal{AMS}$-LaTeX packages allows for the typesetting of multi-line equations with the `align`, `split` and `gather` environments. While these packages can and do work, they all require equations to be broken manually. This poses a problem as breaks are sensitive to changes in fonts, column width and various other edits in the equation itself. Ideally a solution should exist that allows LaTeX to automatically break equations and allow students to focus on the process of writing their thesis.

The equation shown (fig. 2) appears in a biophysics thesis on the effects of drugs on cancer growth. We can see in fig. 2(a) that there are several problems with using the `align` environment to manually break equations as the `\left` and `\right` delimiters for parentheses do not span lines. This has the effect of parentheses with different sizes on different lines. While it is possible to manually adjust the size of the parentheses, this means that considerable effort and attention must be given to equation editing.

One solution to this problem is the `breqn` package, which supports automatic line breaking [8], as seen in fig. 2(b). Usually, when edits have been made to equations, attention must be paid to carefully check that equations have been typeset properly. By allowing the package to handle equation breaking and formatting, students can focus on editing and writing their thesis. (The `dmath` environment is just like the `equation` environment except that it supports line breaking.)

5 Nitrogen isotopes and technical editing

Another project I worked on was a thesis that focused on the nitrogen isotope cycle in the Arctic. Nitrogen and its compounds cycle through the planet and the rates at which the various isotopes do so differ from each other. By tracking the differences in cycle rates, scientists can better understand the implications for climate change. This was a technical editing project and I needed to ensure that physical quantities and chemical equations were entered correctly. As these featured heavily in the thesis, an easier solution for both me and my client was needed.

Physical quantities comprise a number and an associated SI unit (*Système International d'Unitès*) and chemical equations are symbolic representations of chemical reactions from beginning to end. Both can be entered in LaTeX by entering math mode but this poses problems — starting with editing. While entering equations in TeX is intuitive, this is only applicable to mathematical equations and one has to literally enter an SI unit or chemical equation as a mathematical equation which brings us to the next

```
\begin{align*}
\frac{d}{dt}\ApopSig(t) &= K_{\inApop,\UpSig} \cdot \UpSig(t) \cdot \\
 &\left( 1 - \right.\\
 & \left.\frac{\ProSrvlSig(t)}{IC_{50,\ProSrvlSig} + \ProSrvlSig(t)} \right) \\
 & - K_{\outApop} \cdot \ApopSig(t)
\end{align*}
```

$$\begin{aligned}\frac{d}{dt}ApopSig(t) = K_{inApop,Upsig} \cdot Upsig(t) \cdot \\ (1- \\ \frac{ProSrvlSig(t)}{IC_{50,ProSrvlSig} + ProSrvlSig(t)}\Big) \\ - K_{outApop} \cdot ApopSig(t)\end{aligned}$$

(a) Less than ideal line breaking using the `align` environment

```
\begin{dmath*}
\frac{d}{dt}\ApopSig(t) = K_{\inApop,\UpSig} \cdot \UpSig(t) \cdot \left( 1 -
  \frac{\ProSrvlSig(t)}{IC_{50,\ProSrvlSig} + \ProSrvlSig(t)} \right)
  - K_{\outApop} \cdot \ApopSig(t)
\end{dmath*}
```

$$\frac{d}{dt}ApopSig(t) = K_{inApop,Upsig} \cdot Upsig(t) \cdot \left(1 - \frac{ProSrvlSig(t)}{IC_{50,ProSrvlSig} + ProSrvlSig(t)}\right) - K_{outApop} \cdot ApopSig(t)$$

(b) Much better line breaking using the `breqn` package

Figure 2: Equation breaking using `align` vs. `breqn`. (`\ApopSig`, etc., just typesets `\mathit{ApopSig}`).

problem, that of consistency. Typesetting equations has its own set of rules from the spacing between numbers and variables to how superscripts and subscripts are placed. A writer or editor will always have to keep these typesetting rules in mind. A solution to these problems is to utilize the `siunitx` and `mhchem` packages [9, 10].

5.1 SI units

Entering physical quantities in LaTeX can be something of a challenge as the subscript and superscript operators are only available in math mode. Thus,

```
This is my 1^{st} article in a TUG journal.
```

will result in an error and not compile. Of course, one can simply enter math mode to write in superscript but we are then faced with the problem of the text being in (math) italics.

```
This is my 1$^{st}$ article in a TUG journal.
```

becomes

This is my 1^{st} article in a TUG journal.

This particular case doesn't look *too* bad but the italicization can be a problem when typesetting physical units. To solve this, we can enter math mode, print text and then leave math mode.

```
This is my 1$^{\textrm{st}}$ article in a TUG
    journal
```

becomes

This is my 1^{st} article in a TUG journal.

This looks better and works when typesetting physical quantities. While this may be easy for one or two cases when one needs to typeset physical quantities, it can become difficult to maintain consistency during the writing and editing process for many units and troubleshoot problems when they do arise. Consider:

```
The acceleration due to Earth's gravity, $g$,
is $9.81\, \textrm{m}\,\textrm{s}^{-2}$.
```

becomes

The acceleration due to Earth's gravity, g, is $9.81\,\mathrm{m\,s^{-2}}$.

The `siunitx` package provides users with the tools to typeset numbers and units in a way that is both consistent and easy. Each unit is defined by its own macro. When typing a physical quantity, one enters the information syntactically and this makes for much easier editing without the need to constantly compile to see and verify output.

The `siunitx` package can be loaded with the `\usepackage` command as usual:

```
\usepackage{siunitx}
```

Typesetting SI units then becomes simple. The `\SI` macro allows a writer to enter a number with its physical quantity. Thus:

```
The acceleration due to Earth's gravity, $g$, is
\SI{9.81}{\meter\per\second\squared}.
```

As we can see, this package allows for physical quantities to be written as they are said in day-to-day language. This makes for much easier writing as well as proofreading and editing.

5.2 Chemical equations

Many of the same problems that exist with typesetting and editing physical quantities in LaTeX also exist with chemical equations. One can manually enter math mode, typeset a chemical equation using TeX's mathematical notation and then leave math mode. To solve some of these problems and make editing and proofreading easier, I turned to the `mhchem` package. This allows for chemical equations and symbols to be typeset easily.

The biggest plus of the `mhchem` package is that the notation is both easy to read and write. It is much easier to type the chemical notation for phosphoric acid as `H3PO4` than `H_3PO_4`. The former is typically used in email communication when discussing chemical equations and reactions and is thus a more natural form of expression. The `mhchem` package uses this form of notation. Instead of using mathematical notation to write phosphoric acid, we can write the chemical as:

```
$\ce{H3PO4}$
```

which gives H_3PO_4. As the package does all of the heavy work, this means neither the author nor editor has to think about formatting. This makes the writing and editing process much easier.

The true strength of the `mhchem` package comes from entering and typesetting chemical equations. In math mode, the chemical equation for photosynthesis would be written as

```
$6\,\mathrm{CO}_2 + 6\,\mathrm{H}_2\mathrm{O} \
    xrightarrow{\textrm{light}} \mathrm{C}_6\mathrm{
    H}_{12}\mathrm{O}_6 + 6\,\mathrm{O}_2$
```

As we can see, this is not only difficult to write but read as well. Instead, we can write

```
\ce{6CO2 + 6H2O ->[\text{light}] C6H12O6 + 6O2}
```

which is a much more natural representation of

$$6\,\mathrm{CO_2} + 6\,\mathrm{H_2O} \xrightarrow{\text{light}} \mathrm{C_6H_{12}O_6} + 6\,\mathrm{O_2}$$

This is much easier to read, edit and check for problems and can be entered in either math mode or text mode.

6 Conclusion

The packages mentioned are by no means a comprehensive list of the packages that a student should use but rather highlight how appropriate choices can make writing and editing a thesis easier and allow for consistent typesetting. This means that a thesis doesn't just read well but also looks good. University thesis and dissertation class files are designed to fit the needs of as many students as possible and it would be impossible to include all the packages that every student could conceivably need. To do so would make a class file unreasonably large and make it more difficult to maintain.

The advantage of a minimalist design or including the packages that most students need is that packages can be added by students depending on their needs. It would be best for students to take the time to acquaint themselves with the various LaTeX packages available and how they can best use and take advantage of them to make the eventual writing and editing process easier. But for first-time LaTeX users, this may be an intimidating task; not only must they learn how to use LaTeX but they must also select the "right" packages they need. If possible, it may be best to seek the advice of a more experienced LaTeX user who can offer advice on how best to prepare the packages to be used in the document's preamble.

In the absence of such an expert, academic departments can help their students by compiling how-to lists of the most commonly used packages to be published on the department website or be given along with the university's formatting requirements. This should help students be better prepared to focus on the task of writing and publishing their research and hopefully, eventually graduating.

Bibliography

[1] Markus Kohn, Jens-Uwe Morawski, Frank Neukam, and Axel Kielhorn. *The Guide: KOMA-Script*, 2012.

[2] Johannes Braams. *Babel, a multilingual package for use with LaTeX's standard document classes*, 2011.

[3] Alan Jeffrey and Frank Mittelbach. *inputenc.sty*, 2008.

[4] Philipp Lehman, Audrey Boruvka, Philip Kime, and Joseph Wright. *The BibLaTeX Package*, 2012.

[5] Simon Spiegel. *Der biblatex-fiwi-Stil*, 2011.

[6] Stephen Few. *Show Me the Numbers: Designing Tables and Graphs to Enlighten.* Analytics Press, Oakland, Calif., 2004.

[7] Simon Fear. *Publication quality tables in LaTeX*, 2005.

[8] The breqn maintainers. *The breqn package*, 2012.

[9] Joseph Wright. *siunitx — A comprehensive (SI) units package*, 2011.

[10] Martin Hensel. *The mhchem bundle*, 2011.

◇ David S. Latchman
texnical dot designs (at) gmail dot com

A university thesis class: Automation and its pitfalls

Peter Flynn

Abstract

Despite the number of thesis classes available, there are always features that can better be met by writing Yet Another Thesis Class. There are also variations in the documentation, assumptions about preloaded packages, and the extent to which the author can modify the layout.

In the case of UCC, the official requirements were very simple, avoiding the tendency to overspecify detail. The class was required to be usable in any discipline, so preloaded packages were kept to a minimum.

The class attempted to automate as much of the front matter as possible, based on the class options, to avoid unwanted variations in the metadata; and to ensure that the required components appeared in the right place without the author having to do anything.

The result has been piloted with 20–30 PhD candidates for a year, and now needs only a few final changes before release. Two other institutions in the state have already expressed an interest in basing their own thesis classes on this one.

A summary of some of the points covered here was published in a recent *TUGboat* [3]. Recommendations on how to actually *write* a thesis are covered in the companion paper to this [5] and in an earlier summary [2].

1 Yet Another Thesis Class

A recent retrieval from CTAN[1] shows 42 thesis or thesis-related packages currently available for LaTeX (see Figure 1). These are almost all institution-specific, and implement a wide variety of rules and restrictions which fall into five broad groups:

- title page metadata;
- sequencing of preliminary pages;
- wording of formal declarations;
- formatting and layout;
- markup abbreviation and shortcuts.

The level of detail required by each institution varies so widely that using a thesis class from elsewhere usually means some re-configuration and reprogramming, which may be beyond the skills of the author. In some cases there is extensive documentation and an example thesis document; in others

[1] Scripted with the assistance of *dog*, *tidy*, and *lxprintf*, thanks to the robustness of the directory and link structure implemented by the CTAN team.

adfathesis Australian Defence Force Academy thesis format.
afthesis Air Force Institute of Technology thesis class.
beamer2thesis Thesis presentations using beamer.
classicthesis A 'classically styled' thesis package.
ebsthesis Typesetting theses for economics.
elteikthesis Thesis class for ELTE University Informatics wing.
fbithesis Computer Science thesis class for University of Dortmund.
gatech-thesis Georgia Institute of Technology thesis class.
hepthesis A class for academic reports, especially PhD theses.
jasthesis A 'standard' thesis class.
jkthesis Document class for formatting a thesis.
msu-thesis Class for Michigan State University Master's and PhD theses.
muthesis Classes for University of Manchester Dept of Computer Science.
pitthesis Document class for University of Pittsburgh theses.
pittetd* Electronic Theses and Dissertations at Pitt
psu-thesis Package for writing a thesis at Penn State University.
ryethesis Class for Ryerson Unversity Graduate School requirements.
sapthesis Typeset theses for Sapienza University, Rome.
seuthesis LaTeX template for theses at Southeastern University.
suthesis Typeset a Stanford University thesis.
thesis Typeset thesis.
thesis-titlepage-fhac Little style to create a standard titlepage for diploma thesis.
thuthesis Thesis template for Tsinghua University.
uafthesis Document class for theses at University of Alaska Fairbanks.
ucdavisthesis A thesis/dissertation class for University of California Davis.
ucthesis University of California thesis format.
ucthesis209 LaTeX 2.09 document style for UC theses.
uhthesis University of Houston thesis document style.
uiucthesis UIUC thesis class.
umich-thesis University of Michigan Thesis LaTeX class.
umthesis Dissertations at the University of Michigan.
unamthesis Style for Universidad Nacional Autónoma de México theses.
unswthesis UNSW theses.
uothesis Class for dissertations and theses at the University of Oregon.
uowthesis Document class for dissertations at the University of Wollongong.
uscthesis USC thesis style for LaTeX 2.09.
utorontothesis A thesis class definition for University of Toronto.
utthesis Thesis package for the University of Texas at Austin.
ut-thesis University of Toronto thesis style.
uwthesis University of Washington thesis class.
uwthesis209 LaTeX 2.09 style for University of Washington theses.
withesis University of Wisconsin-Madison Thesis LaTeX Class.
york-thesis A thesis class file for York University, Toronto.

* There are potentially many other non-CTAN classes in Electronic Thesis and Dissertation (ETD) sites worldwide.

Figure 1: Thesis and thesis-related packages available from CTAN as of May 2012

there is very light formatting and specification, and authors are left to modify the document as they see fit.

In the present case, there were other requirements which led to the decision to write a local thesis class rather than modify an existing one:

- the class had to be usable by all disciplines, not just those in which LaTeX has traditionally been used the most;
- it had to automate (where possible) those areas where the author would not in any case have a choice;
- it had to allow for the writing of a thesis in the Irish language.

Some guidance was available in the form of the thesis requirements published by the university. I am indebted to the staff of the Registrar's Office, the Graduate Studies Office, and the Boole Library; to the individual colleges, departments, faculties, and schools; and to students and other users for all their comments and suggestions.

2 Building the thesis class

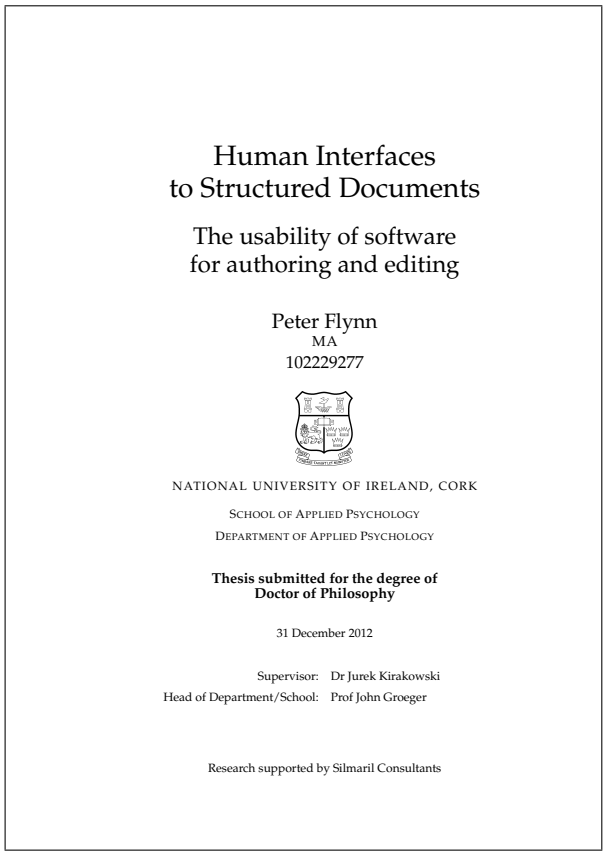
We took an early decision to base the class on the standard report class because it appeared to be the one most familiar to existing users for writing theses.[2] This also meant we could adopt (or in some cases, prohibit) existing class options.

Among the suggestions we received in feedback from users (when we discussed developing a thesis class), was to keep it simple and make it obvious. We interpreted this to mean that we should as far as possible keep the existing meanings for existing commands and environments, and not introduce new ones which were not easily memorable.

Other suggestions included adding optional arguments to certain commands for features which were very frequently used, as repeatedly having to specify something manually was seen as part of the tedium of using LaTeX by users who lacked the programming skills needed to write their own macros.

2.1 Title page metadata

This is often technically one of the simplest parts of a document class design, but it has an impact out of proportion to its position. It is, after all, just one page in a document that will run to hundreds of pages, but it is the first page people see, and the first place that critical eyes will look for errors. It is also hard to convince new postgraduate students


Human Interfaces
to Structured Documents

The usability of software
for authoring and editing

Peter Flynn
MA
102229277

NATIONAL UNIVERSITY OF IRELAND, CORK

SCHOOL OF APPLIED PSYCHOLOGY
DEPARTMENT OF APPLIED PSYCHOLOGY

Thesis submitted for the degree of Doctor of Philosophy

31 December 2012

Supervisor: Dr Jurek Kirakowski
Head of Department/School: Prof John Groeger

Research supported by Silmaril Consultants

Figure 2: Title page layout for the UCC thesis

that they do not have much choice in how this page is laid out.

The current layout (see Figure 2) is vertically- and horizontally-centred, and contains conventional metadata:

- title and subtitle;
- author and [optional] registration number and qualifications;
- institutional identity (crest[3] and name);
- divisional identity (college, school, etc);
- class of degree;
- date;
- names of supervisors and head of discipline.

The only less conventional addition was the acknowledgement of a sponsor — this is commonplace in the Acknowledgements, but its appearance on the title page is becoming a condition of some funding agencies. There is provision for the numbering of multi-volume theses with a `\volume` command, but

[2] Users later reported the most compelling reason was that the report class supports the `abstract` environment, which the book class does not.

[3] The crest is a character in a METAFONT font which was drawn for us by Jeremy Gibbons in 1989, and is only accessible for download locally.

```
\documentclass[apsych,phd,12pt]{uccthesis}
\usepackage{palatino}
\begin{document}
\title{Human Interfaces
     \\to Structured Documents}
\subtitle{The usability of software
        \\for authoring and editing}
\author[102229277]{Peter Flynn}
\qualifications{MA}
\department{Applied Psychology}
\professor{Prof John Groeger}
\supervisor{Dr Jurek Kirakowski}
\sponsor{Silmaril Consultants}
\date{31 December 2012}
\maketitle
...
\end{document}
```

Figure 3: Commands for the title page metadata

I am informed by the Graduate Studies Office that multi-volume theses are a great rarity nowadays.

Because of complexities in the university structure, we had decided to enforce the discipline and degree options in the `\documentclass` command, rather than allow authors to name their affiliation in an uncoordinated manner. This enabled us to use the information before anything else, to set the string names for the faculty (college) and department (school) and the bibliographic reference format prevalent in the discipline. The remaining metadata, therefore, is given with conventional commands as shown in Figure 3.

The `\title`, `\author`, and `\date` are standard; the remainder were added with the commonly-used method of defining an internal command default which is then redefined when the author uses the equivalent external command, for example:

```
\def\@subtitle{\relax}
\newcommand{\subtitle}[1]{%
    \gdef\@subtitle{#1}}
```

We can then test the internal commands for equality to `\relax` during the processing of `\maketitle` to see whether or not the metadata commands were used, so that their absence can be accommodated in the spacing or replaced by warning messages, for example:

```
\if\relax\@subtitle
   \else{\large\@subtitle\par}\fi
\if\relax\@professor You need to give the
   name of your head of discipline
   \else\@professor\fi
```

All the additional commands are given defaults or warnings in this way, so that a beginner accustomed only to `\title`, `\author`, and `\date` (or a user who has not Read The Fine Manual) will not be faced with LaTeX errors.

2.2 Sequencing of preliminary pages

The university rules require only that the Table of Contents comes immediately after the title page. Because the underlying `report` is invoked with the `oneside` option, there are no blank pages. The `\tableofcontents` command is therefore contained in the `\maketitle` command, so that it cannot be omitted or moved.

The List of Figures and List of Tables are also in the `\maketitle` command, but as the class must also be usable in disciplines where there may be no tables or figures at all, the need for these lists is determined by two Boolean switches:

```
\iflof\listoffigures\fi
\iflot\listoftables\fi
```

These are set to false at the end of the document if there were no figures or tables, using global counters defined in the table and figure code, and written to the `.aux` file, where they will take effect on the subsequent run:

```
\ifnum\c@totfigure=0 \write\@mainaux{%
   \string\global\string\loffalse}\fi
\ifnum\c@tottable=0 \write\@mainaux{%
   \string\global\string\lotfalse}\fi
```

However, there are also class options `nolot` and `nolof` which will prevent the LoT or LoF being used even when tables and figures are present — when there are only one or two tables or figures in use, a formal list may not be wanted.

The compulsory formal Declaration that this work is the student's own is also produced automatically as part of the `\maketitle` command, after the ToC (and LoF and LoT, if present), set centred on a page to itself.

No other declaration is required by the university, and it is not usual for copies of any of the forms signed by the supervisor or Extern to be included, but these could clearly be implemented at the same point by the same method if needed.

A considerable number of students require preliminary (unnumbered) sections, before the thesis proper starts with the first part or chapter. These are needed to hold explanatory material such as an Introduction, a list of materials, or tutorial matter on a special topic. While this could be done with a `\section*` command, a new `\prelim` command was created to ensure a page-break beforehand, and to

create an entry in the ToC which would otherwise be absent.

The `abstract` environment was also changed to use this `\prelim` so that it too would occupy a page to itself (the Abstract is limited by the rules to 300 words).

In addition, two new environments were created, `dedication` and `acknowledgements`; the first sets the content centred on a page by itself; the second just uses the `\prelim` command to title the Acknowledgements. The rules do not specify an order for the Abstract, Dedication, Acknowledgements, or any other prelims.

No decision has been taken about the position of glossaries: these are not mentioned in the rules, and while the glossaries package is recommended, there is no compelling evidence one way or another for its placement either here or at the end of the document.

2.3 Formatting and layout

The current rules [7] are very undemanding in this regard:

> The text must be either printed, typewritten or otherwise reproduced on good quality size A4 paper, with a left-hand margin 4 cm. Double or one and a half spacing is recommended. Copies must be bound or otherwise securely fastened and numbered consecutively, page numbers to be located centrally at the bottom of the page.

No mention is made of the other margins, or of the typeface or size, or of the format of bibliographic references, so some unilateral decisions were made (the handling of bibliographic formats is dealt with in § 3).

- The top and bottom margins are set to 3cm and the right-hand (foredge) to 2.5cm;
- The typeface defaults to Computer Modern;
- The body size defaults to 11pt;
- The setspace package is used to set the default to $1^1/_2$ line-spacing — some leading would in any case have been needed for 11pt type on lines this length;
- Page numbering uses roman numerals from the title page to the beginning of the first part or chapter, at which point it restarts in arabic numbers.

After some discussion locally and on `comp.text.tex`, we decided to make the default document setting `\raggedright`. This was partly because it helps avoid H&J problems, especially in the natural sciences where very long words are more frequent; partly because a thesis is not a professionally-typeset publication like a book, and does not appear to benefit greatly from justification; and partly because ragged-right setting improves readability on a page width with relatively long lines.

However, as `\raggedright` also turns off paragraph indentation, the parskip package is used to add space between paragraphs. This layout is in fact expected by students whose experience to date has been restricted to wordprocessors, where it is conventional to use an empty paragraph between paragraphs to simulate paragraph-spacing. Two extra options, `justified` and `indented` can be used to restore the state to book-style.

Running headers and footers were implemented with the fancyhdr package, to provide navigational detail from `\leftmark` and `\rightmark` *without* the capitalization used in the default classes (this proved remarkably difficult to defeat), and *with* the use of an `\hbox` in the definitions to allow long titles to be line-wrapped. In draft mode, the footer also provides the `\jobname` and a timestamp.

Some minor changes included more space above and below captions; the enforcement of the rule for caption positions (Tables: above, Figures: below) by restyling floats with the float package;[4] the allocation of more space to the page number in the ToC, LoF, and LoT; and the reassignment of wider values to the page-fractions for floats, along the lines suggested in the TEX FAQ [1].

Two minor changes are made to the default layout of block quotations and description lists (below), but all the remaining parts, chapters, sectioning, lists, and other structural elements behave and appear as usual in a LATEX document. The `\usepackage` command can be used in the normal way: a list of the packages already in use is contained in the class documentation.

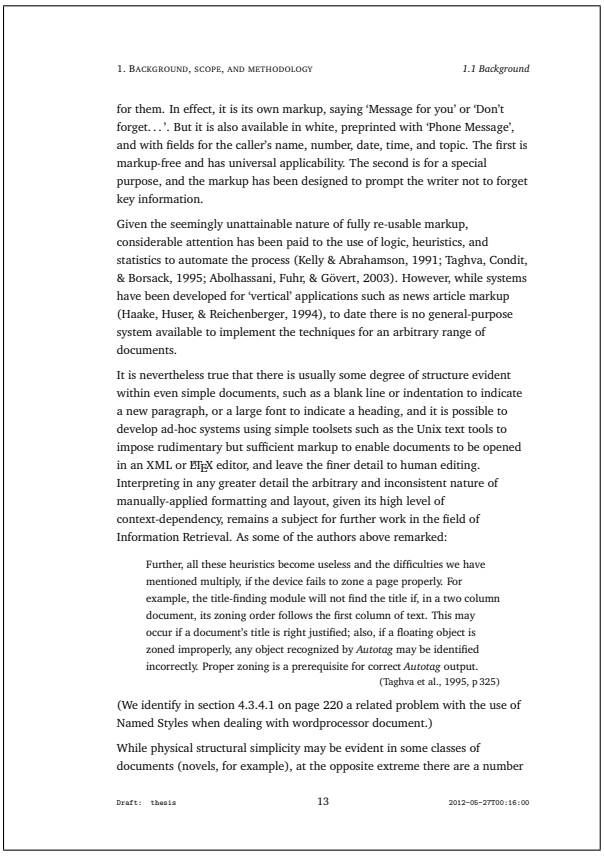
2.4 Markup abbreviation

No attempt was made to abbreviate any of the commands: although students do this frequently for their own pattern of usage, I have not been able to see any common methods. There are a few conventions in some disciplines for short-name macros for commonly-occurring constructs, and authors are free to use them. However, as mentioned earlier, some frequently-occurring constructs were found to benefit from a small amount of automation, the most important of which is the block quotation.

2.5 Quotations

The LATEX default for quotations is unusual, in that it does not make the type size smaller, nor does it

[4] This has the (possibly) useful side-effect of allowing the `[H]` positional specifier.


1. Background, scope, and methodology 1.1 Background

for them. In effect, it is its own markup, saying 'Message for you' or 'Don't forget…'. But it is also available in white, preprinted with 'Phone Message', and with fields for the caller's name, number, date, time, and topic. The first is markup-free and has universal applicability. The second is for a special purpose, and the markup has been designed to prompt the writer not to forget key information.

Given the seemingly unattainable nature of fully re-usable markup, considerable attention has been paid to the use of logic, heuristics, and statistics to automate the process (Kelly & Abrahamson, 1991; Taghva, Condit, & Borsack, 1995; Abolhassani, Fuhr, & Gövert, 2003). However, while systems have been developed for 'vertical' applications such as news article markup (Haake, Huser, & Reichenberger, 1994), to date there is no general-purpose system available to implement the techniques for an arbitrary range of documents.

It is nevertheless true that there is usually some degree of structure evident within even simple documents, such as a blank line or indentation to indicate a new paragraph, or a large font to indicate a heading, and it is possible to develop ad-hoc systems using simple toolsets such as the Unix text tools to impose rudimentary but sufficient markup to enable documents to be opened in an XML or LaTeX editor, and leave the finer detail to human editing. Interpreting in any greater detail the arbitrary and inconsistent nature of manually-applied formatting and layout, given its high level of context-dependency, remains a subject for further work in the field of Information Retrieval. As some of the authors above remarked:

> Further, all these heuristics become useless and the difficulties we have mentioned multiply, if the device fails to zone a page properly. For example, the title-finding module will not find the title if, in a two column document, its zoning order follows the first column of text. This may occur if a document's title is right justified; also, if a floating object is zoned improperly, any object recognized by *Autotag* may be identified incorrectly. Proper zoning is a prerequisite for correct *Autotag* output.
> (Taghva et al., 1995, p 325)

(We identify in section 4.3.4.1 on page 220 a related problem with the use of Named Styles when dealing with wordprocessor document.)

While physical structural simplicity may be evident in some classes of documents (novels, for example), at the opposite extreme there are a number

Draft: thesis 13 2012-05-27T00:16:00

Figure 4: Default page layout for the UCC thesis

defeat indentation on the first line, both of which are established conventions.[5] In academic work, a block quotation also usually requires a citation.

To implement this, the `quotation` environment was modified to change the size and start with a `\noindent`; and to take an argument, the BibTeX key of the cited passage. This enables the quotation to be set with a right-aligned citation immediately below. Although such a citation would be compulsory, it is currently defined as optional in order for its presence to be tested, and to allow for uncited use where the context already makes the origin obvious. However, if an optional argument such as this is to contain page or chapter references, which are themselves optional to the `\cite` command, additional armour is required:

```
\begin{quotation}[{[p.36]{smith92}}]
```

In response to user requests for an Epigraph at the start of chapters, an `epigraph` environment was created, in the same way as the modified `quotation` above. This has two arguments, however: a compulsory one for the BibTeX key, but if the citation is to be informal and non-rigorous, that argument can be empty, and the optional argument can be used, for example:

```
\begin{epigraph}[Popular saying]{}
```

In both cases this is cumbersome and needs regularising, so a future version will probably use the `xargs` package to handle the additional metadata.

2.6 Description lists

A frequent annoyance for users is the inability of the label argument to an `\item` in the `description` list environment to be broken at line-end when it is very long. The default formatting is also unusual compared with modern practice of setting the label value on a line by itself (as with the default formatting of HTML's `<dt>` element type). A number of alternative formats from the LaTeX Companion [6] have been tried, but no final decision has been taken on this yet.

3 Adherence to university structure

The university is currently in transition from the traditional hierarchical Faculties and Departments to a more fluid structure of Colleges and Schools. As a result we have a complex and overlapping transitional organisation in which disciplines are being merged, split, and renamed. In order to ensure that the correct names are used, the author's affiliation and class of degree must be given as class options, rather than as free text in the argument of a command. This also avoids misspellings, and the unfortunate tendency of some students to represent the discipline in terms other than the official ones. With the imminent arrival of electronic submission, where the name of the discipline or school will be part of the PDF metadata, regularity is becoming more important.

In creating these options, it became clear that other data could also be keyed to it, in particular the bibliographic reference format required for each discipline. As a result, selecting an affiliation option now both sets the correct naming *and* presets the `.bst` file (and any associated `.sty` file) for the discipline.

As there are currently 87 options for affiliation and 92 for class of degree, this method would have been unworkable in terms of manual maintenance. Fortunately, the class was developed using an XML-based methodology which generates the `.dtx` and `.ins` files, so the relevant string names and tokens could simply be transformed from an annual XML extract from the databases maintained by the university administration. In any case, if and when the transitional phase of restructuring is completed, the

[5] I have never been able to find a use for the `quote` environment, so the use of `quotation` is recommended to authors.

class interface will be updated to use a key/value syntax rather than 179 separate options!

4 Testing, feedback, and adoption

Informal testing was initially carried out by use in my own thesis, but was extended to drafts provided by students who came looking for help with LaTeX formatting. Discussions were held with the Registrar's Office and the Graduate Studies Office to ensure that the layout implemented conformed to the rules.

In January 2010, an early version of the XML-generated class was made available locally for download [4], and over the course of 18 months about 45 students used it for their theses, reporting bugs as they were discovered.

The feedback was largely positive, and the automation of the title page and prelims was seen as a major benefit. The most useful feedback came as bug reports, and led to a spate of updates over the next six months as various solutions to errors were tested and implemented.

Some unresolved issues remain:

- separate options for the disciplines and classes of degree need to be replaced by key/value pairs;
- glossaries appear to be much more common than was previously envisaged;
- the formatting problems of description lists remain unresolved;
- the debate continues over the default unindented, paragraph-spaced, ragged-right setting *vs* justified and indented setting;
- the identification of the 'right' bibliographic reference format for each discipline is problematic. An enquiry among colleagues representing each discipline provided only a 20% response, so there is a lot more data to collect (the default has been set to Harvard). A few disciplines have two common formats — physics, for example, allows either IEEETR or AIP — so a mechanism is required to allow that to be specified.

At the moment the assumption is that students will be using BibTeX, not *biblatex*, as not all the formats required are yet available in the latter.

The class has by no means been tested to destruction: it appears to work with all the common packages, including hyperref, but the more use it gets and the more bugs are reported, the more likely it is to work.

A number of academics who use LaTeX themselves have started recommending the class to their students, and some informal changes have been made to create variant formats within the class for essays, term papers, and minor dissertations. An approach has also been received from a group of students in another institution in the state (which has no resident LaTeX expertise or support), asking for help in writing a thesis class, and I am aware of at least three students in a third institution who have adopted the class and simply changed the identity by editing the `.cls` file.

Writing a class file is a non-trivial activity, and I was fortunate to have access to a development and maintenance methodology which made editing and creation very much easier than writing a `.cls` file by hand. Having a clearly-defined goal makes development easier, as does having a set of patterns to work to, and I am indebted to the many authors of the classes listed in Figure 1 for their work. The intention is that from version 1.00 the package will be available on CTAN (minus the crest, which is reserved to the institution).

References

[1] Robin Fairbairns, editor. *TEX Frequently Asked Questions*. UK TEX Users Group, Cambridge, UK, Mar 2012.

[2] Peter Flynn. Sorry, Professor, the dog ate my thesis: How to expect the unexpected when using LaTeX. In *Living and Working with LATEX*. London Mathematical Society, Oct 2006.

[3] Peter Flynn. Typographers' Inn. *TUGboat*, 33(1):8–10, 2012.

[4] Peter Flynn. UCC Thesis Document Class. `http://research.ucc.ie/latex/#uccthesis`, Apr 2012.

[5] David Latchman. Preparing your thesis in LaTeX. In *TUG 2012*, Boston, MA, Jul 2012. TEX Users Group.

[6] Frank Mittelbach, Michel Goossens, Johannes Braams, David Carlisle, and Chris Rowley. *The LATEX Companion*. Addison-Wesley/Pearson Education, Boston, MA, 2nd edition, 2004.

[7] University College Cork. Thesis Submission: Layout of Thesis. `http://www.ucc.ie/en/graduatestudies/current/thesis/Layout/`, Nov 2009.

◇ Peter Flynn
Electronic Publishing Unit,
University College Cork
Cork, Ireland
Phone: +353 21 490 2609
`pflynn (at) ucc dot ie`
`http://research.ucc.ie/profiles/H505/pflynn`

LaTeX source from word processors

Bart Childs

Abstract

Hennings' CTAN survey is a good starting point when considering projects implied by the title of this article. I found it a fair view of most related packages. He suggests having one of two goals: converting the *document structure* or converting the *appearance*. My goal is neither of these. I want to produce LaTeX source that is accurate in content, clean, and therefore maintainable.

This is in keeping with Knuth's original goals in producing TeX: graphic excellence and a document convenient for archiving. Structure and appearance are important. I believe clean LaTeX is more likely to have this intrinsic result (not use of word processing systems). My current conversion system is a hybrid based on the use of the Open Office `Writer` package, its `Writer2LaTeX` application, and macros for the Emacs editor written in Elisp. The test cases for this system are books: 1) on rotordynamics, 2) a C++ programming text, 3) a memoir on a friend's life including significant text fragments in the Czech language, and 4) a novel that includes three love triangles. Even the worst case with significant mathematics formatting done in WordPerfect is tractable; *I did not say easy*. The lack of (intelligent?) use of word processors causes many of the problems. I estimate that a 300-page novel written in a reasonable dialect of Word, WordPerfect, or `Writer` could be converted to LaTeX in an hour or two.

1 Genesis

My primary formatting system has been TeX-based for more than thirty years. Throughout this time I have had occasional need to import small parts of documents done in word processors into my TeX-based documents. I have accomplished that in a number of ways: from keyboarding small projects to somewhat automatic conversion depending upon what was available. I used some of the earlier systems discussed by Hennings [2].

Several years ago, two colleagues were writing a text on "programming" and became aware that they would have significant advantages if they could convert the half of the book that was completed to LaTeX and take some instruction on how to complete the rest in LaTeX.

I sketched the process and created a small set of Emacs Elisp macros to do that conversion. We agreed to the generalities with plans to make a formal agreement upon the return of the senior author from a summer-long trip. Much of the LaTeX work was to be done by the junior author, naturally. The health of the junior author suddenly deteriorated and my conversion project was cancelled.

I continued to be intrigued by the concept. I learned more Elisp, added macros, and a number of *open* packages that seemed to offer promise as a means of getting much of the conversion done in an automatic manner. I never felt that a mostly automatic conversion was realistic for projects involving significant mathematics content. I expected to pursue a "PhD with a screwdriver" approach. I was willing to do this based on working from the Word `.rtf` (Rich Text File format), total extraction of text without formatting, *and/or* a mostly automatic conversion that needed tweaking — my pipe dream.

A few years after retirement, a friend and colleague in the college of engineering asked me for help finding someone to keyboard a new text he was writing based on a few dozen of his research papers — and related studies. The topic of the text is rotordynamics — from small pumps and turbines to large ones as in the main engine of the space shuttle. I resurrected my plan and we agreed on the plan of work.

The draft source of this rotordynamics text is being done in WordPerfect, the formatter the author has used for many years. Most of the text is being adapted from the author's contributions in the subject. The current version is approximately 400 pages in length with another 25% to be added. The lists of contents, figures and tables will likely occupy 18 pages. There are hundreds of equations with one of them being a full page.

2 The process evolves

I started this conversion using the process I had prototyped for the programming text. The rotordynamics text was quite a different document because of the large fraction of displayed equations. The displayed equations and figures in the rotordynamics text require approximately the same fraction of space required by figures, programs, and code fragments in the programming text. Most (maybe all) of the code fragments, programs, and figures in the programming text were restricted from floating. There had to be some "manual floats".

I did some small portions of the rotordynamics book as manual conversions for test cases. Some of the equations were manually entered because conversion of mathematics among word processing systems was generally accepted to be non-existent (I think that is improving). The manual process was based on: a) having a `.pdf` of the document, b) editing the `.rtf` file, c) editing a text file exported from a word

processor (with some encoding), and/or d) a form of LaTeX exported from one of several systems. I was delivering LaTeX source faster than I could have keyboarded it from good copy. Still, it was unsatisfactory because it was mostly a manual process.

The source documents were done in WordPerfect on a PC and I was doing LaTeX on a Mac. There are good TeX and Emacs systems for the Mac using Mac OS X. Some Emacs systems were not acceptable to me because my system uses function keys.

I continued to strive for big improvements because keyboarding mathematics would be slow. A significant improvement came by changing the format in which sources were delivered to me. The source was 1) edited to remove the graphics from the WordPerfect source, and 2) exported in `.rtf` form, with 3) the graphics elements put in a `.zip` file. The version of WordPerfect being used would create `.rtf` files hundreds of times bigger than needed if the graphics was included in the export to the `.rtf`. Removing the graphics was no loss because it — like the mathematics — was not being exported.

I would take the `.rtf` from WordPerfect, import it to OOo `Writer`, and save it! This apparently lost nothing but gave a smaller file and therefore my system was faster in using it. I also noticed that `Writer`'s export of *text with encoding* was different from the other systems I had used. Further, the export could be done in Unicode which was compatible with Emacs.

Apparently there was significant appreciation of Unicode in the WordPerfect export process. The export of the mathematics from WordPerfect was not converted but many symbols, Greek letters, etc. were now viewable on the screen. Most (La)TeX users should be able to glean the proper content from a printed `.pdf` of the WordPerfect. Now, the Emacs macros could do much more. At this time, my benefactor had other obligations and so I had time to work on the macros and test the system using the modified process.

I continued to learn more Elisp.

3 Keeping the mind busy

My benefactor's diversions lasted a bit longer than planned. I read more about Unicode and realized how provincial some of us are here in the *English-only USA*.

A college buddy of mine is a Czech immigrant and was corresponding with a publisher in the Czech Republic about his memoir. When he wrote to the publishers and sent it by email, the formatting was lost. I suggested learning a bit of LaTeX, converting it to `.pdf`, and emailing that. He had sent me a draft of the book so I could create some examples. The published version [1] was done while I was creating this system. Of course I was naïve and would still have been so had I not read Horak's note [3].

But while waiting, I thought I could polish my Emacs macros to handle his Czech problems. It was fairly easy and with the improvements in the `Writer` export process, it was really easy. I mention this project because it shows evidence of *real problems* with similar projects. That will be discussed later.

In the abstract I mentioned a novel about three love triangles. That project was technically trivial but also contains the same *real problems* with conversion of word processor sources.

4 Real problems

There are several sources of problems that impeded progress in these projects. Some of these sources could be avoided by "user learning" while others resulted from differences in the design and implementation of the systems they used. The authors had several kinds of problems that automatic conversion did not handle:

1. Inconsistent use of functionality.
2. Wrong use of functionality.
3. Not using available functionality.
4. Oops. Operator, operand placement. Misunderstandings. Mysticism about style files.

This quote is in section 1.2 of the *`Writer2LaTeX` User's Manual* [4]:

> You can use LaTeX as a typesetting engine for your OOo documents: `Writer2LaTeX` can be configured to create a LaTeX document with as much formatting as possible preserved. Note that the resulting LaTeX source will be readable, but not very clean. ... You will find that `Writer2LaTeX` uses the principle *garbage in — garbage out*!

Each of the above examples of *garbage in — garbage out* was present in at least two of the test cases cited. *Garbage in — garbage out* may be a bit strong of a description for these but the message is clear. For example, in the Czech memoir it was certainly appropriate to attempt to show correct accents — Horak [3] would be proud. It overwhelmed the author's limits of skill with the systems he was using.

Each of the authors has a doctorate and has taught at major universities. They are consistent users of computers but obviously are not the most persistent readers of the formatter manuals. Maybe the manuals are poor, non-existent, or not convenient? Maybe the easy-to-use graphics interfaces overwhelmed the authors? Maybe these interfaces

do not encourage users like these to seek the information they need? Maybe they just do not care?

4.1 Inconsistent use of functionality

The author of the memoir that used many Czech words, phrases, and sentences is to be saluted for attempting to make that text look proper to a Czech reader. There are five special items in this sentence

> On my next visit to Prague, he joined Vlád'a and me, along with our wives, for lunch at a French restaurant in *Obecní dům* (Municipal House).

The nickname Vlád'a has an accent over the letter "a" and an accent often called a *caron* modifying the letter "d". The accented "i" in the first italicized word is a dotless "i". Finally, the second italicized word has an accent that almost appears to be the degrees (as in temperature) symbol. Although it was not the author's intention, the distances these accents were raised or kerned differed in most cases. (I do not claim my caron here is perfect.)

4.2 Misuse of functionality

In the rotordynamics book there were many instances of using different Greek characters as the same: the phi and varphi, ϕ and φ, as well as others. Since this document was constructed using papers written years ago, this is easily understood.

The author of the novel containing three love triangles suffered a similar problem. The author did not like the double prime (") for the opening and closing quotes. When he wrote the first part he selected special graphics characters for the quotes. When he wrote the other two parts, the smart quotes were automatic for him. He did not recall why; it may have been a new revision of his formatter.

4.3 Not using available functionality

In two of the test cases the authors used itemized lists. The exported form yielded consecutive lists of one item. This did not bother the bulleted lists but would have been an error with enumerated and description lists.

In many cases the authors did not use styles, so chapter and section beginnings show the formatting but no LaTeX commands. This is not a total loss, because I convert the section numbers into labels that would aid if we were trying to resolve differences in my output with the older version.

4.4 Oops?

These examples can be difficult. A glaring example is that WordPerfect's mathematics operators may follow the operand in some cases. In LaTeX the operator is always first! I did not find a general rule as to when to expect this. My Emacs macros for adjusting this are interactive to enable the user (me) to minimize such problems.

A really big *oops* worth repeating is the lack of using styles, which caused inconsistencies. I had to handle some of these manually.

5 Typical Emacs macros

The first versions of these macros were developed when I was using an export that was usually designated *text with encoding*. This export would discard all (or nearly all) formatting, such as emphases. The improvements in `Writer2LaTeX` have led to a reduced need of this kind of detailed editing. Still, the concepts in the design of these macros are applicable in the current system of conversion as well as keyboarding original documents.

This list contains three cases where it is more efficient to use *text with encoding* exports than the converted exports, assuming the goal of clean LaTeX. These came from the rotordynamics text, the programming text, and the User's Manual. These are:

Tables Tables are exported with all formatting on every cell. The usual (LaTeX) procedure is to give default formatting in a template and exceptional formatting when needed in a cell.

Mathematics Text is often used for explanatory purposes in equations.

Programs as well as verbatim text need special handling.

Portions of some documents are easier to convert by exporting as *text with encoding* and then inserting the formatting by editing. Two examples are mathematics that does not convert and formatted code fragments in a processor where font changes are done manually rather than using a package like `listings`.

The macros were implemented using the mouse (or similarly functioning device) to point or highlight in conjunction with function keys. In Emacs one can also highlight a region of text by setting the mark and moving the point. The function keys can also be modified by use of `shift`, `control`, and `alt`.

5.1 Applying fonts to text

In this paragraph there are single words and a three-word sequence that are emphasized by changing fonts. The default font is changed to *italics* or `typewriter`. Source exported as *text with encoding* will have formatting removed. A similar situation occurs when text is inserted into mathematics code.

The user can highlight a phrase or click within the single word. Then the user presses the appropriate function key for the formatting command to be inserted with grouping of the appropriate text. If the user has clicked within a word, then the extent of the word is determined by whitespace delimiters. Clicking on whitespace is a special form of this — the commands are inserted and the cursor placed on the right brace for user input.

Instead of highlighting a region, the user can use the Emacs form of setting the mark and moving the cursor to the other end of the region. I implemented these functions for **bold**, *italic*, sans serif, and `typewriter` fonts. I did not insert the italic correction but easily could have paying attention to the following character. I did not because in many cases it is just not needed, and besides, the user should have some responsibilities. The same functions are reused for simple grouping and the `\text{}` commands which were used mostly in math modes.

5.2 Inline mathematics

Inline mathematics is common in the rotordynamics text. Most of the resulting mathematics is usually a fraction of a line in length.

The implementation is like the font changes in the previous subsection. A significant difference is that the export processes handling WordPerfect mathematics yields significant artifacts of excessive white space and formatting trash. This almost always includes many of the grave characters — this must be an *escape character* for the internal form of WordPerfect mathematics.

I have not had a reasonable test case with Word mathematics, yet. There are small examples of mathematics in the programming text.

5.3 Display mathematics

The concepts in the previous subsection are applicable. However, there are several forms of display mathematics. These forms were used in the rotordynamics text:

1. `\[...\]`, the standard for display equations without numbers.
2. `\begin{equation*}...\end{equation*}`, just an alias for the former, or *vice versa*.
3. `\begin{equation}...\end{equation}`, which numbers the equations and should have an accompanying `\label`.
4. `\begin{equation}\begin{split}... \end{split}\end{equation}`, which numbers a collection of equations and should have an accompanying `\label`.

Chapter 8 of Frank Mittelbach et al.'s *LaTeX Companion* has some seventy pages of excellent details of advanced mathematics formatting.

I implemented these four display math choices using one function key and prompting the user for which of the above forms was desired. I developed similar choice macros for presenting fractions and matrices which made conversions faster and most importantly more consistent. The most important facet of this conversion is that with a little care the totality of the mathematics was converted correctly and hours of detailed, laborious proofreading was avoided.

5.4 Programs, code fragments, verbatim text

Programs should be formatted by language sensitive packages like `listings`. The package `fancyvrb` requires some study but gives great results. Both packages come with inline commands whose use is aided by adaptation of the above font changing and inline mathematics concepts.

5.5 Other macros — fix-up

There were several other macros that aided the conversion. I consider these to be "fix-up" in nature. These include:

- `\caption`s in the rotordynamics text often contain inline mathematics. The use of the LaTeX delimiters (`\( \)`) is not allowed; they must be converted to the TeX toggle (`$`).
- Interactive aid to standardizing presentation of fixed-point and floating-point numbers.
- Locating multicharacter super/subscripts that were likely exported incorrectly (needing grouping).
- Locating likely problems due to insertion of inadvertent whitespace.
- Locating unescaped TeX control characters.
- Macros to aid the insertion of labels and their references.

6 Current system

The current system has been improved greatly with the release of OOo `Writer2LaTeX` version beta 1.2. Despite its being labeled a beta release, I have not found any problems to date.

I find these observations about this new release interesting: 1) the user's guide is 10% shorter and 2) the output files are 3–5% shorter than with version 1.0. The LaTeX output is cleaner, as most of the reduction in the size is the elimination of needless formatting like: 1) most paragraphs were inside

grouping braces and a declaration that I used English and 2) `{\textquotedblright}` for a simple (”). A cursory look at the user's guide indicates some removal of redundancy. There is a lack of the completeness that is characteristic of the documentation of releases from the TEX communities.

I plan to work with OOo and continue to make this product better. I believe it to be the best hope I know of, especially in the *open* domain.

The following quote is from one of the pages on its web site (`http://writer2latex.sourceforge.net/index3.html`):

> You will never get a result that looks *identical* to the original, in fact that's the whole point: LaTeX is in general a superior typesetting engine compared to Writer. For example LaTeX produces much better results for formulas, it has an excellent paragraph and page breaking mechanism, it uses ligatures etc. On the other hand Writer has a few features that LaTeX does not support well. If the layout of your document depends on text flowing around pictures or linked text boxes, you will never get good results with Writer2LaTeX.
>
> According to TeX's author Donald E. Knuth, TeX is a *typesetting system intended for the creation of beautiful books - and especially for books that contain a lot of mathematics* (quoted from "The TeX book"). **Writer2LaTeX** will aim to produce excellent result for this kind of documents; including of course shorter texts with a book-like layout.

This quotation is fair but I think it makes my point “go ahead and inhale”. Show the logos (TEX and LATEX) correctly, use the correct dashes and spacing, use the proper quotes, ...

6.1 Examples of other problems

I present an annotated list of a few other problems I addressed in the macros. These are based on two of the test cases: the rotordynamics text and the programming text. I think it is fair to classify most of these as “not very clean LATEX”.

Export of spacing. The export of chapter 5 of the rotordynamics text has 47 occurrences of (}), a space preceding a right brace. The majority of those are in constructs like `\textit{word }` while most of the rest are weird constructs like `\textit{ }` and `\textbf{\textit{\ \ }}`.

The first may be sloppy keyboarding by the author. The second seems to be intentional spacing, why not (\ )? The last is likely a hacked indentation kludge?

Inline mathematics. Some inline mathematics is converted to italics. That is troublesome to me because it should really remain as unconverted mathematics. Then too, that may be the fault of the author.

Export of structure. The structure of the chapter and lists range from inconsistent to missing. This is likely the authors' fault as the use of styles seems to be the cause.

7 `Writer` and friends

In spite of these remarks I salute OOo. I believe that the `Writer` package and `Writer2LaTeX` application have made a great contribution to the goal of converting many documents into a form for better presentation and archival, namely (LA)TEX. That may not have been the intent. The intent may have been to enable a good `Writer` user to simply use LATEX as an output device?

The LATEX code output in version beta 1.2 is improved, but not clean. The *`Writer2LaTeX` User's Manual* is 45 pages in length. The exported LATEX (with the `clean` option) source averages about fourteen occurrences of `\mdseries` and twelve occurrences of `\textstyleSourceText` per page. Each paragraph is grouped with `\mdseries` as the start. The latter is effectively an alias for `\texttt` and used in tables.

8 Conclusions

Reasonable document interchange and archival quality is now possible for a wide range of systems. I believe that (LA)TEX is the most reasonable basis for many archival systems.

The advances by OOo and its `Writer` system are impressive and appreciated. I hope that its *open* status and development will continue. Note: I have addressed only a small part of a large project — OOo.

A point made in a number of venues is the problem of TEX systems not having a native graphical input process. Lyx and OOo are touted as solutions — along with several others. The authors of the three test cases I have used show that the graphical interfaces are not a solution to the problems — in my humble opinion. All the authors are highly educated and familiar with the problems of getting people to learn at the college level. Still, each has shown the results from casual learning about their systems. The effective use of styles, consistent use of symbols and special functions, document structure, etc., were lacking in each of their documents.

The first line of a LaTeX document requires a statement of the class of the document. There is a finite number of them. It does not seem to enter the stream of consciousness for many that if they learned how to type "Mary had a little lamb" on a machine that there should be at least a small change in the start of a letter to a sweetheart, a grocery list, or any other class of documents.

In a moment of frustration I lamented "Users avoid using LaTeX because you have to learn how to do some things while users of Word believe if it takes any non-obvious effort to do something, *it should not be done!*"

I raised the question earlier about why educated users of computers seem to get so little from user's guides and manuals. Maybe the manuals are poor, non-existent, or not convenient? Maybe the easy-to-use graphics interfaces overwhelmed the authors? Maybe these interfaces do not encourage users like these to seek the information they need? Maybe they just do not care?

Was the intent in creating `Writer2LaTeX` to give the user "LaTeX as an improved output device"? I think that poses a bigger challenge, "How do you teach a `Writer` user to write for LaTeX?"

9 Questions

I did not intend this as a FAQ but thought it might be a good way to end the present paper.

LL LaTeX Do any of the test cases use LaTeX beyond Leslie Lamport's book?

Answer *No* for memoir and book on the three love triangles. *Yes* for the science and engineering texts. Packages used: `float`, `lscape`, `makeidx`, `fancyvrb`, `graphicx`, `array`, `amsmath`, `amssymb`, `sidecap`, `wrapfig`, and `caption`. These were probably not all necessary, but useful.

Word test case? What do you want for a Word test?

Answer A one-pager, like Norman Naugle's *An Elementary Sum*. Then, many others would help. I hope it would also convert to `Writer` and back too.

How long? How long did it take you to type Norman's note?

Answer An hour or so. The next question might be, why didn't you just do it in Word? Well, probably that would have taken seven or eight hours — and fortunately I do not have Word in my house.

References

[1] Charles Ota Heller. *Prague, My Long Journey Home*. Abbott Press, December 2011.

[2] Wilfried Hennings. Converters from PC Textprocessors to LaTeX – Overview, June 2012. `http://tug.org/utilities/texconv/pctotex.html`.

[3] Karel Horák. Those obscure accents.... *TUGboat*, 29(1):42–44, 2007.

[4] Henrik Just. User's manual for `Writer2LaTeX`, March 2012. `http://sourceforge.net/projects/writer2latex`.

◇ Bart Childs
Texas A&M University
College Station, TeXas 77843-3112
USA
`bart (at) tamu dot edu`
`http://faculty.cse.tamu.edu/bart/`

The MacTEX install package for OS X

Richard Koch

Abstract

MacTEX installs everything needed to run TEX on a Macintosh, with a single click of the mouse. I'll discuss the history of this package — Wendy's conspiratorial lunch and Jonathan Kew's all-night coding session — modifications over the years, and important changes in the 2012 release.

1 A demo

MacTEX is a flat file available as a free download, linked from `http://tug.org/mactex`. It is a very large download, about 2.16 GB, but smaller versions are available for users with slow download links.

On the desktop, the file inherits an icon from Apple's Installer program.

Figure 1: MacTEX-2012

Double clicking this icon starts the installation process and the window shown at the top of the next page appears (fig. 2).

This window is familiar to Mac users because the same window appears when they install other packages, and (until recently) when they installed system updates. The installation process is summarized by the list of items on the left; this list is fixed by Apple and cannot be changed. Notice that the dialog background is a merging of the Mac OS X logo with a Duane Bibby drawing of the TEX lion and Donald Knuth. The idea of a TEX-related illustration goes back to Jonathan Kew's initial version of the package; this particular form was provided by Bob Kerstetter.

At the extreme top right of the window you'll notice a small padlock. It brings up a window listing the Developer ID Certificate issued by Apple to validate this package. Apple's Mountain Lion system refuses to install packages which lack an official Developer signature (although workarounds exist). An Apple engineer contacted me in May to make sure MacTEX would have this signature, so Apple knows about TEX.

When the user pushes Continue, a more detailed document is displayed; we don't show it here. Pushing Continue again leads to a pane listing licenses governing the various pieces of MacTEX. This dialog is also shown on the next page (fig. 3).

(The abstract to this paper says that MacTEX installs TEX with a single button click. Are you counting clicks? By my count, we are up to four, not counting the double click which started the process.)

The next dialog (fig. 4) appears during what Apple calls the "Installation Type" portion of the process. This allows users to select the disk where the installation will appear. MacTEX always installs on the system disk in standard spots, so this dialog merely shows the available space required on the hard disk. A button at extreme left leads to a Custom installation panel.

That custom dialog is shown below the Installation Type dialog (fig. 5). One can see that MacTEX is separated into pieces: TEX Live, GUI applications, Ghostscript, Convert, the Latin Modern fonts, and the TEX Gyre fonts. Most users will install TEX Live, but users with a favorite editor may skip GUI Applications, and users who compile Ghostscript and ImageMagick themselves or obtain them as part of Fink or MacPorts will skip those packages. We provide two optional font packages; these packages install duplicates of certain TEX fonts in Apple's Font Directory, making them available to standard Macintosh programs like Adobe Illustrator.

A final click leads to a standard dialog (not shown) asking the user to supply an Administrator Password. This password is required because TEX Live will be installed in `/usr/local`, which is owned by root. On the Macintosh, standard users are Administrators and the Administrator Password is their own password rather than an actual root password.

Then installation occurs, with a progress dialog as shown (fig. 6), followed by a final success dialog (fig. 7). Not counting custom installation, the process requires six clicks and one password.

2 GUI applications

TEX is installed under `/usr/local/texlive`, a location not shown by Apple's Finder. Consequently, the only files commonly seen by users are those installed in `/Applications/TeX`. We install two editors, TeXShop and TEXworks, and we install LATEXit, a graphical application which allows users to input equations in TEX source format, convert them to PDF, and paste the PDF into standard Macintosh programs using drag-and-drop operations. We install the Excalibur spell checker, but we do not install the more commonly used cocoAspell by Anton Leuski because it has a special installer and extra dictionaries available from Leuski's web site, `http://cocoaspell.leuski.net/`.

Figure 2: Initial Dialog

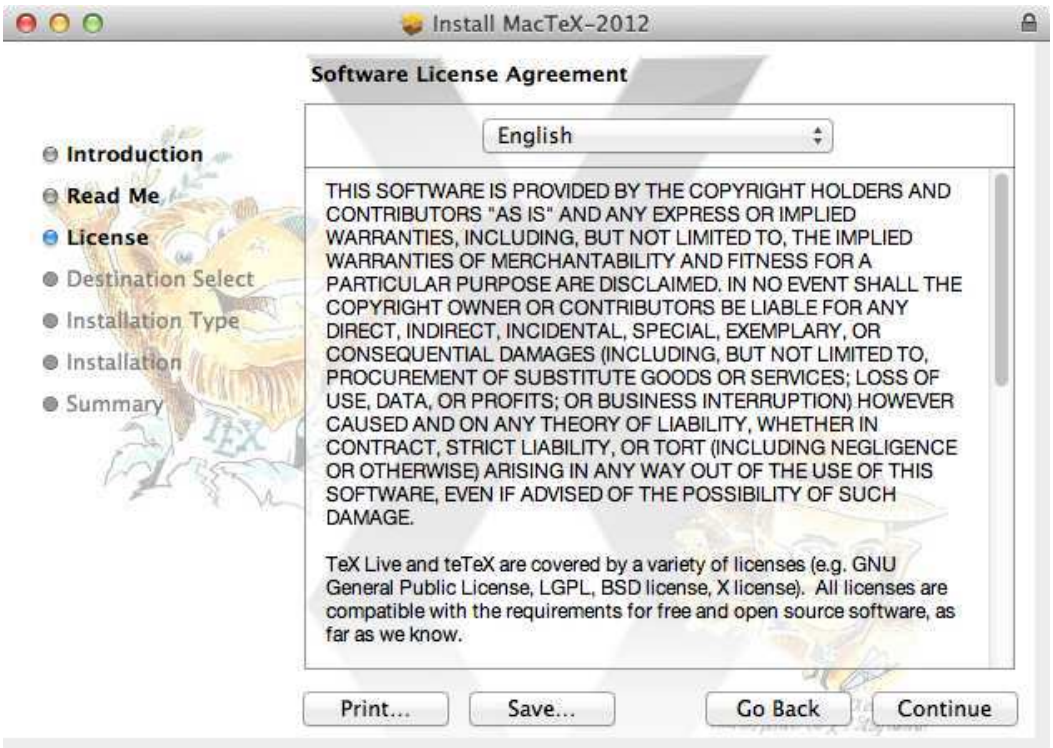


Figure 3: License Dialog

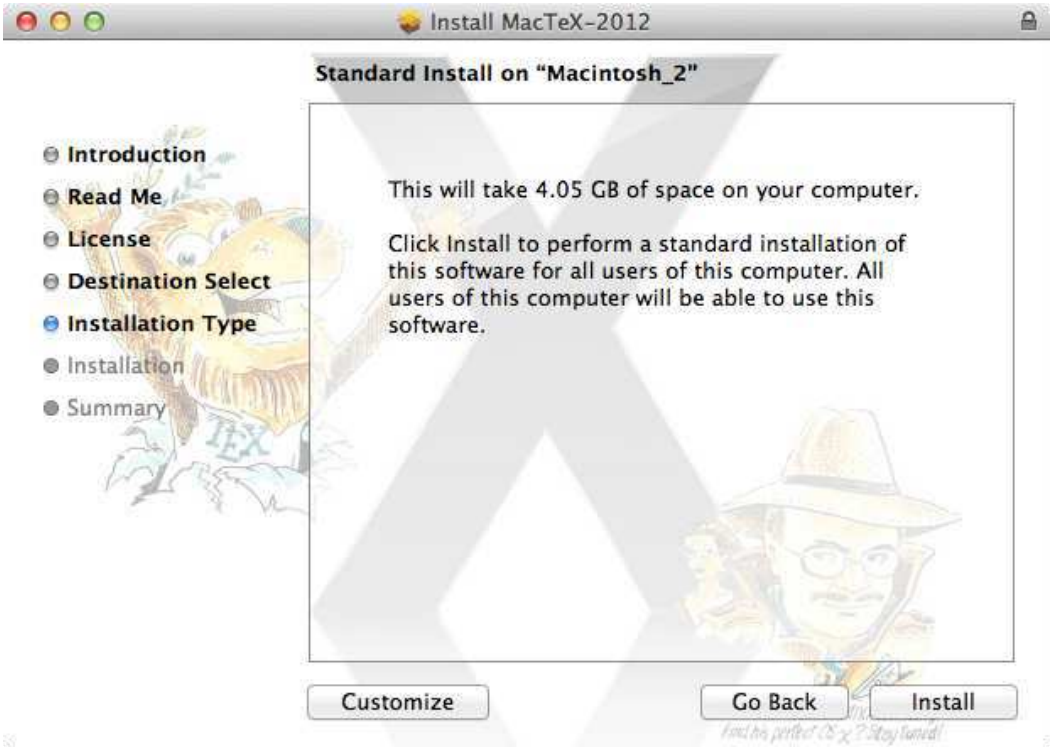


Figure 4: Installation Type Dialog

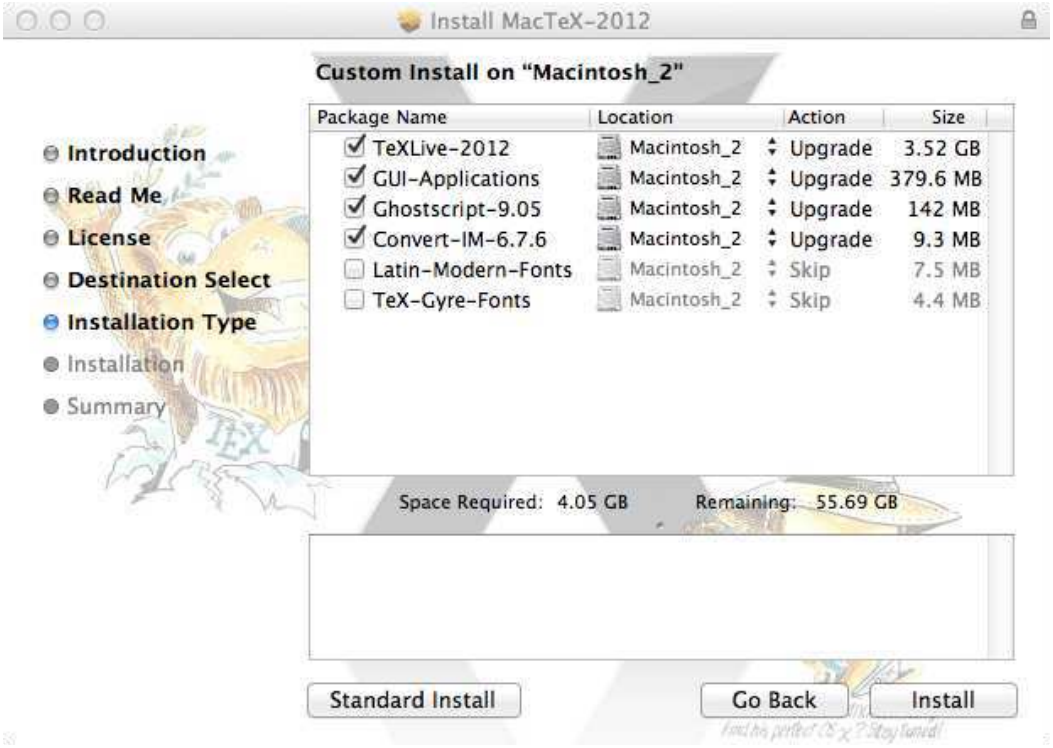


Figure 5: Custom Install Dialog

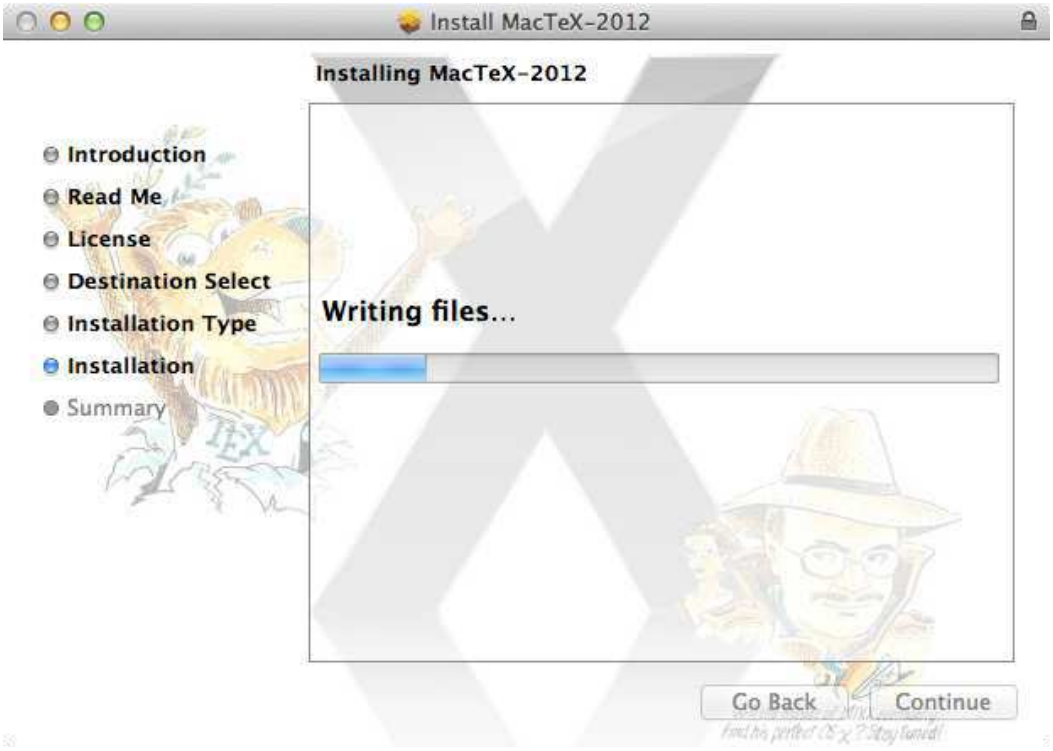


Figure 6: Actual Installation

Figure 7: Success Dialog

Finally, we install Adam Maxwell's wonderful TEX Live Utility, a program which gives a standard Macintosh interface on the TEX Live `tlmgr`, allowing users to keep TEX Live packages up to date, search for new packages, and configure paper size, among other things.

3 Configuring TEX Live and GUI applications

In a word, nothing is needed. The installer guesses the user's desired paper size from Mac printer settings. It adds the TEX binary location to the `PATH` variable, and makes TEX man pages readable. All supplied GUI applications are already configured to find TEX.

4 Getting started with TEX

There is no standard spot for user documentation on the Macintosh, so we install a short `READ ME FIRST` document in `/Applications/TeX`. This document starts with a two-page introduction to TEX for a first time user, leading the user through the process of writing and typesetting a short document with TeXShop. Since both pages contain half-page illustrations, the introduction is very short with only the essential steps. The user is also directed to a movie in the TeXShop help menu illustrating the typesetting job.

After that, the `READ ME FIRST` document lists links to information about other editors and GUI front ends on the Macintosh, to important TEX information at TUG and elsewhere, to online tutorials about TEX and TEX Live with links that immediately bring up the information, and to the TUG web page for books about TEX.

5 TEXDist

MacTEX installs the TEX Distribution data structure and Preference Pane by Gerben Wierda and Jérôme Laurens which makes it easy to use multiple TEX distributions on Mac OS X. This structure is described in detail in my article *Support for multiple TEX distributions in i-Installer and MacTEX*, *TUGboat* 28:3 (2007), `http://tug.org/TUGboat/tb28-3/tb90koch.pdf`.

Installing MacTEX does not erase distributions from past years. The TEXDist Pref Pane is added to the standard Apple System Preferences by MacTEX. It lists all available TEX distributions on the present Mac, and lets the user choose the one to make active with a single button click. This click automatically reconfigures all GUI editors and utilities, and modifies `PATH` and `MANPATH` in shells. It isn't even necessary to restart applications. You can typeset with TEX Live 2011, keep your editor active, switch to TEX Live 2012 with the Pref Pane, and typeset again with TEX Live 2012.

MacTEX installs a link named `/usr/texbin` which points (indirectly) to the active TEX distribution. Any GUI app configured to find TEX at `/usr/texbin` can share in the advantages of the TEXDist structure.

6 The (nonexistent) special version of TEX Live installed by MacTEX

The key message of this section is that *there is* **no** *special version of TEX Live for the Mac!* We have always strictly followed the rule that we install the full TEX Live, completely unmodified.

On Unix machines, TEX Live is generally installed by running the `install-tl` script in a shell. The TEX Live portion of MacTEX is constructed by removing `/usr/local` from the machine creating it, installing TEX Live to `/usr/local` on that machine with the TEX Live install script, pointing Apple's PackageMaker software to the install location, and asking it to construct an install package. Later when MacTEX installs this package on a user machine, it runs a postinstall script to configure paper size and do a few other things, but this script does not modify files in TEX Live.

The `install-tl` script has a menu which allows users to change a few configuration options. We change just three things, namely setting `TEXMFVAR`, `TEXMFCONFIG`, and `TEXMFHOME` to (respectively):

- `~/Library/texlive/2012/texmf-var`
- `~/Library/texlive/2012/texmf-config`
- `~/Library/texmf`

On Unix machines, these variables point to "hidden" locations in the user's home directory. The home directory on the Macintosh has a special folder named `Library` which is the standard place for configuration information, so we use it for TEX.

In Apple's latest operating systems, Lion and Mountain Lion, the Library folder is itself hidden. But by holding down the Option Key while pushing the Finder's Go menu, the user can visit this folder.

7 Smaller install packages

MacTEX is a gigantic download. That is why we supply the much smaller install package BasicTEX, which is about 66 megabytes. This package installs most files needed for ordinary typesetting using TEX, LATEX, or XETEX. It contains the Computer Modern fonts and Latin Modern fonts. Many users report that all their documents typeset fine with this installation.

A new user can easily produce TEX documents using only TeXShop or another GUI front end with BasicTEX.

In 2012, the subset of TEX Live installed by BasicTEX became one of the install schemes for TEX Live. BasicTEX is exactly the result of installing `scheme-small` with `install-tl`.

BasicTEX used to contain ConTEXt, but then Mojca Miklavec introduced a standalone ConTEXt distribution. Since ConTEXt is upgraded more often than once a year, and since this distribution can coexist with TEX Live, it makes sense for ConTeXt users to install it separately. See `http://wiki.contextgarden.net/ConTeXt_Standalone`.

We also distribute MacTEX Additions, an install package containing everything in MacTEX except TEX Live: `http://tug.org/mactex/morepackages.html`.

8 A defective install on the NeXT machine

I am one of the few people who bought a NeXT computer. Software for this machine was not — let us say — abundant. So owners bought more or less everything released for the machine.

Early in the life of the NeXT, I bought software which came on a CD and was installed by the NeXT analogue of Apple's Installer program. As installation proceeded, icons in the dock began changing to question marks, and by the end only a couple of icons remained. Puzzled, I clicked on one of the question marks, but nothing happened. With some concern, I started the Terminal program to run a shell, but Terminal had vanished. Eventually I logged into the machine remotely and discovered that the entire Applications folder had been erased.

A few hours later, the company selling the CD issued a profound apology and explained that the NeXT installer didn't anticipate symbolic links in the Application directory, or maybe didn't anticipate hard links, or maybe it had nothing to do with links, but at any rate it didn't anticipate *something unusual*. I don't know the details, but I learned a lesson: installers are dangerous.

The memory of that event colors my life to this day. Every so often, users ask for a feature in MacTEX which is not provided by Apple's PackageMaker utility. When I explain the problem, users often sketch a way to construct the package directly without using Apple's utility. I will *never* do that. There are a half dozen Apple engineers who know everything about pax files, compression algorithms, soft links, hard links, dangerous links, secret links, and everything else that could go wrong with an install package. The thought that they'd lose their jobs if something went wrong is strangely reassuring.

9 Gerben Wierda's i-Installer

My first TUG conference was in 2001 in Delaware (`http://tug.org/tug2001`), where I met Hàn Thế Thành, the author of pdfTEX. Since pdfTEX outputs PDF files and the graphic system of Mac OS X is based on PDF, his software made creating an interface to TEX a breeze.

TUG 2001 occurred only a few months after the first release of Mac OS X, version 10.0 on March 24, 2001. I talked about TeXShop, which had been running on an early beta version of the system, and about i-Installer, a program by Gerben Wierda which installed TeX, Ghostscript, ImageMagick, various font utilities, and other Unix software. Gerben's software worked over the Internet, downloading packages from servers as needed. The current web page at `http://ii2.sourceforge.net/` doesn't seem to contain an initial release date, but it must have been early in the beta period for Mac OS X. Gerben ceased supporting i-Installer in 2007, but Google searches lead to users who report trying to install TEX with it as late as April of 2011.

During advance preparation for my 2001 talk, I noticed a situation when the Finder could become confused. Sure enough, I ran into that problem during the talk and I had to restart the Finder before proceeding. Afterward, someone came up to me and said "I couldn't care less about TeXShop, but I was very impressed when you restarted the Finder without rebooting the Macintosh".

10 Wendy's lunch

In Delaware, I also met Wendy McKay — a Mac fanatic with twice my enthusiasm and five times my energy. MacTEX is really Wendy's invention.

Gerben's i-Installer was an ambitious project, able to install not just TEX packages, but also more general Unix open source code. It had to deal with network issues like choosing an appropriate server, dealing with timeouts, and security matters. That made for a program with an industrial look which could be intimidating for new users. Wendy began lobbying for a "one-button TEX installer". This lobbying extended over several TUG conferences.

Everything came to a head in North Carolina at the Practical TEX 2005 conference (`http://tug.org/practicaltex2005`). By then, Wendy was lugging suitcases full of electronic equipment to conferences, and had set up a long distance meeting of Mac folks on Thursday afternoon, in which Europeans not at the conference could participate remotely.

To prepare for the meeting, Wendy asked Mac folks to share the same table for Wednesday lunch. She soon began discussing a one button installer, said something like "who's going to volunteer to make one", and suddenly turned to Jonathan Kew and said "it looks like you, Jonathan". Done. The sweetest maneuver I've ever witnessed.

11 Jonathan Kew

Jonathan had to leave the conference early on Friday. We wished him safe travels after the Thursday meeting, expecting an installer in a couple of months.

TUG conferences are fun, but information tends to come so fast that I'm exhausted after a couple of days. In North Carolina, I went to bed as soon as I could Thursday evening. When I got up the next morning, I read an email message from Wendy: "Jonathan just finished the installer."

Jonathan programmed all night. And he didn't have just a rough draft of an installer. He had a package which installed everything: TEX (in those years we used teTEX), Ghostscript, ImageMagick, and font utilities. His installer displayed a custom Mac OS X image. It was constructed with elaborate shell scripts, so the entire process of creating it involved installing teTEX and Ghostscript with Gerben's i-Installer and then running a few scripts. The installer contained postinstall scripts to set the user's `PATH` and `MANPATH` variables, using code which Jonathan found hidden inside i-Installer.

That morning at breakfast, Jonathan willed the project to me. I said "but I don't even understand shell scripts" and he said "read what I have, it is self-explanatory." And it was. When I need a shell script today, seven years later, I look up Jonathan's scripts and carefully copy the syntax.

12 Herbert Schulz and MacTEXtras

The next year, we put the install package on the TEX Collection DVD. The DVD also contains extensive extra material curated by Herbert Schulz. This extra material includes the front ends Aquamacs, iTEXMac, LyX, and TEXMaker. Information is provided about BBEdit, TextMate, and TextWrangler. It contains the Skim previewer, the CocoAspell spell checker, other useful utilities, and documentation and demos. All this extra material is also available on the MacTEX web site in a package named MacTEXtras.

For several years now, Herbert and I have been jointly responsible for the Macintosh portion of the DVD. Herb is an expert on features of TEX Live I ignore: installing extra fonts, running updmap and updmap-sys, issues with restricted shell escape. People who attended the Boston conference learned that Herb is also an expert on several features of TeXShop which I ignore.

13 Gerben's surprise

The MacTEX installer Jonathan wrote depended heavily on Gerben Wierda to do the heavy lifting. In May of 2006, Thomas Esser announced that teTEX would no longer be upgraded, and suggested that users migrate to the TEX Live project. Gerben began issuing warning messages to the TEX on OS X mailing list which most of us ignored; after all, he had provided TEX reliably since the beta days of Mac OS X.

After some grumbling, Gerben indeed developed a new TEX distribution based on TEX Live rather than teTEX, called gwTEX. He told us it would be officially released at TUG 2006 in Marrakesh (`http://tug.org/tug2006`), which started on November 9th, and to expect a surprise announcement there.

For the surprise, see `http://www.tug.org/twg/mactex/award/2007/gerben/aboutgwtex.html`. It shows a picture taken at this event; Gerben is holding a large sign with the text "I Quit". To my knowledge, this is the first time that the announcement of a new software release was accompanied by the announcement that support for it would end in two months.

Gerben's announcement caused some fast footwork on the MacTEX front, and after several months of indecision we switched to providing an unmodified full TEX Live in the package.

14 MacTEX changes over the years

We provide MacTEX to a small group of beta testers before releasing it to the Internet and for the DVD. I need to mention the most important beta tester, Bruno Voisin, who was at the Boston conference. Bruno is an even stronger Mac fanatic than Wendy, and he will complain bitterly if an interface behaves in a non-Mac fashion. The hidden files in TEX Live are visible files in `~/Library` due to Bruno. In MacTEX 2012, the Ghostscript installation is improved over past years due to discoveries made by Bruno this spring. Thanks.

There have been a few significant changes in MacTEX over the years. The first occurred when we added optional install packages, so that, for example, a user could install only TEX Live, skipping other packages. This change was made to accommodate users who obtain Ghostscript and ImageMagick through MacPorts or Fink, or compile them directly from source. But it also made MacTEX easier to maintain, since the various pieces can be created independently.

Originally, we installed a few libraries from ImageMagick and some font utilities to `/usr/local/lib`. A concerted effort has been made to get rid of these; today we install *no libraries*. This is a deliberate choice which will not change; it makes the lives of developers easier because they do not need to contend with foreign libraries on their machines.

When MacTEX was first provided on the DVD, it contained its own separate copy of TEX Live. TEX Live is enormous, so putting two separate copies on the DVD rapidly become untenable. Nowadays, we provide a special version of MacTEX for the DVD; this special version installs TEX Live by calling the `install-tl` script on the DVD. Therefore the DVD contains only one copy of TEX Live, used by MacTEX and by users on other platforms.

The most recent change occurred in 2012. Apple's Mountain Lion system requires that install packages be signed by a registered Apple Developer. Until this year we created MacTEX using the original PackageMaker, which created install packages which were actually folders in disguise. Such packages cannot be signed. So with some pain we switched to Apple's newest PackageMaker, which creates flat files. Since the interface did not change, most Mac users probably don't know that anything is different.

This more recent PackageMaker is poorly documented and contains several unfinished features. This caused two problems with the 2012 version of MacTEX, which I like to call *The Two Fiascos*.

15 The first fiasco

By design, MacTEX doesn't provide choices for the user. That's the whole point of the package: install and run.

But in the final days before release, we discovered that Apple's new PackageMaker constructs packages which allow users to change where software is installed. For example, our package installs TEX Live in the standard location, `/usr/local/texlive/2012`, but users can change this location to their home directory. If they do this, they will have a folder named `2012` in their home directory containing a gigantic number of files owned by root. And TEX won't work. *If you are a Mac user, don't change our default locations!*

Quiz: glance back at the pictures of installation shown at the start of this paper. What unexpected item in these pictures is an active element the user can manipulate to cause this fiasco?

16 The second fiasco

Some users trying to install from the DVD first copy the install package to their hard disk and eject the DVD. Installation won't work if they do this because the Install package reads TEX Live from the DVD. So we check for the problem by making certain that the directory `/Volumes/TEXCOL2012/texlive` exists. If not, we abort installation with an error dialog that the DVD must be mounted.

Apple's new PackageMaker makes this check very easy. It contains canned JavaScript modules which can be dragged into the project to test for various conditions. We dragged, and we dropped. Then we made a test DVD, and we tested by installing on Leopard and on Snow Leopard. Worked like a charm. At that point, TUG manufactured the DVD.

It turns out that the canned JavaScript doesn't work on Lion or Mountain Lion. So without help, the version of MacTEX on the DVD will not install on these systems. Users who want to install from the DVD should go to `http://tug.org/mactex` and download a very small fix for this problem.

17 A final glitch

MacTEX installs two Ghostscript binaries, `gs-noX11` without X11 support and `gs-X11` with X11 support. In a post-install phase, MacTEX determines whether X11 is installed on the Macintosh, and sets the symbolic link gs to point to the appropriate binary. Users who upgrade to Mountain Lion and then install MacTEX will have no problems.

Apple removes X11 during the Mountain Lion upgrade because it now wants users to obtain Xquartz directly from the open source developers. So users who install MacTEX and then upgrade will end up linked to the wrong version of Ghostscript. To fix this, either install Xquartz or use Terminal to run the following commands:

```
cd /usr/local/bin
sudo rm gs
sudo ln -s /usr/local/bin/gs-noX11 gs
```

18 Making MacTEX

The full documentation explaining how MacTEX is constructed is now part of TEX Live. It can be found in the TEX Live source repository available at `tug.org/texlive/svn/` in the file `Master/source/mactexdoc.tar.xz`.

◇ Richard Koch
2740 Washington St.
Eugene, Oregon, USA
koch (at) math dot uoregon dot edu
http://uoregon.edu/~koch/

TEX and friends on a Pad

Boris Veytsman

Abstract

TEX on an Eee Pad is quite workable.

1 Introduction

Some time ago a blog entry [15] made quite a splash in the community. The (semi-anonymous) author stated that LATEX cannot be made on a tablet due to its "speed, bloat, and complexity" and needs a complete rewrite. He also asked for a complete change in the licensing scheme of TEX components in order to make LATEX acceptable for the App Store.

In my opinion, this is a complete misunderstanding of what TEX is and what it is not. The most important thing, TEX is not an "app" in the same sense OpenOffice is. TEX is designed as a compiler which takes a program written in a language understandable by humans, and creates "binary code" in a language understood by machines. The `tex` files we write are not "documents" in the same sense as OpenOffice files. They are *programs* with familiar (to a programmer) constructions like macros, loops, conditionals. The result of compilation is code — a DVI, a PS or a PDF file — which is basically a set of instructions for a machine to produce printed pages or images on a screen. Furthermore, the TEX system does not have just one compiler, but a family of compilers and utilities, like `gcc` and friends. While the article [15] exclusively discusses LATEX, it is nice to have index processors, bibliography formatters, font manipulation utilities and many others, not to mention alternative engines to `pdfetex` and alternative formats to LATEX.

Once we understand that we are talking about a family of compilers with auxiliary programs and libraries, many objections in [15] become irrelevant. The compilation of the engines is a complex process with many helper applications? Anybody who ever tried to bootstrap `gcc` from source would not make this comment. Huge code base? Well, the code base of C/C++ with *all* the free libraries commonly used is not small, either.

The comparison of a TEX distribution to a C/C++ distribution including all possible libraries is not as far fetched as it seems. A modern distribution like TEX Live contains almost all the freely distributable code from CTAN, the Comprehensive TEX Archive Network. The users of Perl and R have created and maintain similar huge collections — CPAN and CRAN. It is commonly considered to be the strong feature of these languages rather than a weakness.

The minimal subset of TEX Live occupies about 40 Mb — by no means large by today's standards. The full distribution, indeed, is 3.5 Gb and growing, because it includes solutions for many different problems: typesetting musical scores and chess games, working with many languages and scripts, drawing geographic maps in any projection, creating circuit diagrams, using medieval fonts, and many, many others. A user can install only the parts which she really needs: for somebody the main reason to work with TEX might be the possibility to typeset in the Klingon language, while another user might work with TEX for years and never find out it speaks Klingon. Fortunately, modern distributions provide easy and powerful tools to select only the parts of TEX and friends one really needs. And modern hard disks are large enough that installing the full distribution is a reasonable default.

Still, the proof of the pudding is in the eating, so we would like to offer the most convincing proof that TEX can be used on a tablet: the experience of compiling and using TEX Live on one. This is the aim of this paper.

2 Device

There are reports of running TEX on Apple IOS devices [3]. However, due to the usability considerations discussed below I chose an Android tablet on `armv7l` architecture. I recently got an ASUS Transformer Eee Pad TF101 [2]. The selling point was the dual nature of the device: it has a detachable keyboard with an extra battery, so it can be used both as a light tablet (when the lower part is detached) and a netbook (when the lower part is attached). It has turned out to be a very useful and surprisingly powerful machine.

Another advantage of this choice is that Android-OS is a derivative of GNU/Linux, so I hoped to use the familiar Linux tool chain for working with it. I found out that one can actually install a full distribution as an application, running in a `chroot` environment, which made my task quite simple.

2.1 Rooting

To really own the device one need to "get a root" on it. This is a dangerous operation *which may break your device and almost certainly voids your warranty.* Please do not do this unless you absolutely understand what you are doing, and in no circumstances blame me if anything goes wrong!

The operation is described in detail in [1]. After rooting the device you need terminal access and (optionally) a convenient shell to work before you

start Linux. Android Terminal Emulator [12] and BusyBox [16] are good choices.

For Emacs and vi users it is useful to map the "back" key on the dock to Escape (see, e.g. [7]) by editing `/system/usr/keylayout/asusec.kl`.

2.2 Linux in `chroot`

The user interface of Android assumes working with one full-screen application at any time. This is a challenge for compilers like TEX and friends: one needs an editor window, a compiler window, a log window, etc. One way to solve this challenge is to use an Integrated Developed Environment like TEXworks [8] or TEXnicCenter [17], where the window management is done by the application. However, I am an incorrigible Emacs user with the fingers used to all those Control- and Meta-sequences, so it would make most sense to recreate the familiar work flow on Android. Emacs is well integrated in Unix-like systems, and I looked for the way to install a Linux environment on the device. There are several ways to do so:

1. Dual boot. The system can be booted to Linux *or* to Android. At any time the device is either Linux or Android, without much integration between these two.

2. Virtual machine. The host operating system (Android) emulates the hardware for the guest operating system (Linux). There is some integration between the OSes, but this requires a lot of processing power. Besides, we are not aware of any VM software with Android as host (while there are many VMs with Android as guest).

3. Chroot. This is a unique possibility due to the fact that Android uses the Linux kernel with the same system calls. Thus, one can just start the standard Linux daemons and programs under Android with a separate directory mimicking the standard Linux layout (the "root directory" for these processes, hence the name).

I chose that last possibility. It should be stressed that the Linux programs with this solution are tightly integrated with the native Android system. In Figure 1, the `top` program shows both Android processes (e.g. `ys.android.jump`) and Linux processes (e.g. `top` itself).

The application "Linux Installer" [13] turned out to be an excellent way to go. I used it to install Debian Squeeze on a 32 Gb removable SD card.

To allow a non-root user to run useful processes, the user must be granted some rights to the corresponding devices on the tablet. This is done by adding the user to the groups listed in Table 1 in the file `/etc/group` on the Linux side.[1]

Group	gid
`AID_NET_BT_ADMIN`	3001
`AID_NET_BT`	3002
`AID_INET`	3003
`AID_NET_RAW`	3004
`AID_NET_ADMIN`	3005
`AID_MISC`	9998
`AID_SDCARD`	1015

Table 1: Some useful Android groups

To get X running on the Android I used the following trick, taken from an Android forum [5]. One can start a "headless" X accessed from a remote computer through the VNC protocol. The interesting thing is that this "remote" computer can actually be the *local* machine with a VNC client talking to the server on the same device at IP address 127.0.0.1. There are VNC clients for Android, normally used to remotely access computers from a mobile device. That one can deploy them to access *the same device* is a good example of the power of unintended use.

On the server side I installed TightVNC [6], included in the Debian distribution. There are a number of free VNC clients for Android. Unfortunately I did not find a single one that provided easy access to Escape and Control keys, which are essential for Emacs users. Therefore I chose Jump VNC [14]; this is the only non-free software used in this project. Jump VNC understands Control and Escape keys on the keyboard dock and provides a convenient on-screen panel with special keys when the dock is disconnected. I hope some free VNC client can implement this convenient interface in the future.

I did not use resource heavy environments like KDE or Gnome; instead, I installed a lightweight window manager, FVWM.

Some additional screenshots of the resulting desktop can be found at `http://android.galoula.com/screenshots/LinuxInstall/Boris_Veytsman_2011_12_31/`.

3 Installing and running TEX

Debian Squeeze has TEX in the distribution, so it runs "out of the box". Unfortunately, it is the very old TEX Live 2009.[2] As a nice exercise, I decided to build TEX Live 2011 binaries from sources — maybe

[1] I am grateful to Gaël Person for this advice.

[2] Debian is famous for its stability, which means, among other things, rather obsolete packages.

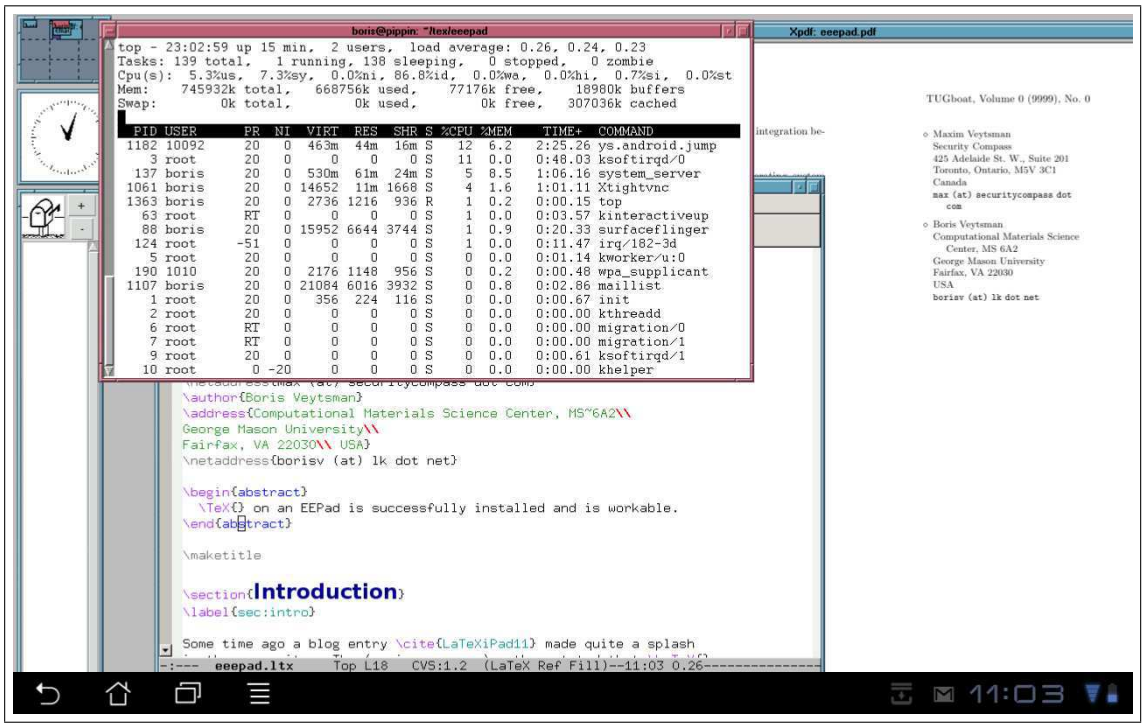


Figure 1: A screenshot with `top` running

the statement in [15] that the author could not do this was an additional incentive.

The build instructions on the TEX Live web page [10] were easy to follow. The full build took about 2.5 hours. I decided to install all TEX Live packages: being a TEX consultant, I prefer the full installation since one never knows what a next customer might need. The installation of the binaries and packages went without a problem.

The subsequent TEX Live 2012 builds on this machine also proceeded without errors, compiling all 350 binaries. This `armel-linux` port became an official part of TEX Live with the 2012 release.

The update cycle of LuaTEX and ConTEXt is typically faster than that of TEX Live. I also maintain a ConTEXt standalone distribution (see `http://wiki.contextgarden.net/ConTeXt_Standalone`).

The resulting environment is quite usable. In fact this paper was partially written on this device.

To check how fast TEX is on the device, I used the following files:

1. `story.tex`: the famous story about Mr. Drofnats by A. U. Thor (see [9]).
2. The source of *The TEXbook*[9].[3]

[3] While TEXing of this book is prohibited, a special dispensation for benchmarking is traditionally recognized by the American Mathematical Society. I am grateful to Barbara Beeton for explaining this.

File	Pages	Engines				
		`tex`	`pdfetex`		`xetex`	`luatex`
			DVI	PDF		
`story.tex`	1	0.77	0.90	1.45	2.97	1.49
The TEXbook	494	5.84	6.94	11.63	12.05	18.50
LATEX 2_ε	492	N/A	23.84	26.28	29.10	32.66

Table 2: Benchmarks; times are in seconds

3. `source2e.tex`: the sources of LATEX 2_ε, as of 2011/06/27 [4].

All benchmarks were done with TEX Live 2011. Since only the `plain` format uses Knuthian TEX in this distribution, I benchmarked this engine only on the first two files. Also, LATEX requires several runs for the references to converge; only one run was measured. All benchmarks were done by calling

```
time TEX_COMMAND FILE
```

and taking the first time ('real time') from the output. Each test was repeated three times, and the average was taken.

The results are in Table 2. As seen from this table, a 500-page document is processed in about 15 seconds in plain and about 30 seconds in LATEX.

Another benchmark is compiling the present paper. One run of `pdflatex` takes 4.1 seconds.

The full compilation from scratch (issuing `make` after `make distclean`) involves a run of `pdflatex`, a run of `bibtex` and two runs of `pdflatex`.[4] This takes 13.3 seconds to complete. For comparison, on my desktop (4-core 2.4 MHz processor) it takes 1.9 seconds, and on my laptop (ASUS Eee PC 900HD, 800 MHz processor) — 11.9 seconds.

While details are beyond the scope of this paper, I would like to mention that such important and useful tools as R, Maxima, Octave, and Gnuplot also run on the device without any problems and with reasonable speed.

4 An alternative approach

This paper describes a creation of a full Linux environment under `chroot` on an Android device. Recently Mǎ Qǐ Yuán started to work on an alternative approach [11]: a compilation of TEX binaries using Android NDK. In this way one can create standalone applications that do not require a Linux installation to run. My tests showed that these applications are not faster than those under Linux; this is not surprising, since Linux applications are not run in a virtual machine, i.e., do not incur any overhead.

5 Conclusions

TEX and friends *can* run on an Android tablet. Moreover, they make it a useful work machine rather than a mere consumer toy.

Acknowledgements

I am grateful to my son Max Veytsman for rooting the device, helping with understanding the Android system, reading the manuscript and many useful comments; to Gaël Perron for help with Linux Installer; to Qǐ Yuán Mǎ for telling me about his approach; to Karl Berry and Barbara Beeton for encouraging this paper, suggesting benchmarking targets and editing the text.

References

[1] Anon. AsusTransformer Root + CWM recovery. `http://androidroot.mobi/technical/asus-eee-pad-transformer-tf101-root-cwm-recovery`, May 2011.

[2] AsusTeK Computer, Inc. Eee Pad Transformer TF101. `http://www.asus.com/Eee/Eee_Pad/Eee_Pad_Transformer_TF101`, 2011.

[3] Kaveh Bazargan. TEX as an eBook reader. *TUGboat*, 30(2):272–273, 2009. `http://www.tug.org/TUGboat/tb30-2/tb95bazargan.pdf`.

[4] Johannes Braams, David Carlisle, Alan Jeffrey, Leslie Lamport, Frank Mittelbach, Chris Rowley, and Rainer Schöpf. *The LATEX 2ε Sources*. LATEX3 Project, June 2011.

[5] Dangermouse. Gnome, KDE, IceWM or LXDE desktop on your Android! `http://www.androidfanatic.com/community-forums.html?func=view&catid=9&id=1615`, March 2009.

[6] GlavSoft LLC. TightVNC software. `http://www.tightvnc.com`, 2012.

[7] Patrick Hof. Installing a Debian chroot on the Asus Eee Pad Transformer. `http://www.offensivethinking.org/thoughts/2011/07/14/debian-chroot-eee-pad-transformer`, July 2011.

[8] Jonathan Kew and Stefan Löffler. TEXworks. Lowering the entry barrier to the TEX world. `http://www.tug.org/texworks`, 2012.

[9] Donald Ervin Knuth. *The TEXbook*. Computers & Typesetting A. Addison-Wesley Publishing Company, Reading, MA, 1994. Illustrations by Duane Bibby.

[10] TEX Live. Build procedure. `http://www.tug.org/texlive/build.html`, 2012.

[11] Qǐ Yuán Mǎ. TEX Live for Android. `http://code.google.com/p/texlive-for-android`, 2012.

[12] Jack Palevich. Android Terminal Emulator. `http://www.appbrain.com/app/android-terminal-emulator/jackpal.androidterm`, 2012.

[13] Gaël Perron. Linux installer. `http://android.galoula.com/en/LinuxInstall`, 2011.

[14] Phase Five Systems LLC. Jump Desktop. `http://www.jumpdesktop.com`, 2011.

[15] Valletta Ventures. The price of a messy codebase: No LATEX for the iPad. `http://vallettaventures.tumblr.com/post/13124883568/the-price-of-a-messy-codebase-no-latex-for-the-ipad`, November 2011.

[16] Denys Vlasenko. BusyBox. `http://busybox.net`, 2012.

[17] Tino Weinkauf and Sven Wiegand. TEXnicCenter — The center of your LATEX universe. `http://www.texniccenter.org`, 2012.

◇ Boris Veytsman
School of Systems Biology &
Computational Materials
Science Center, MS 6A2
George Mason University
Fairfax, VA 22030
USA
`borisv (at) lk dot net`

[4] As it happens, the second run is not necessary and is triggered by the message "Label(s) may have changed" produced by a too-cautious LATEX.

YAWN — A TEX-enabled workflow for project estimation

Pavneet Arora

Abstract

A framework for using TEX and its variants — the term TEX is used generically in this article — in a project estimation workflow is discussed. While the emphasis here is less on the final output and more on issues related to the upstream processing, the framework itself does not place limits on how TEX might be used to create more beautiful output.

1 Introduction

Part of TEX's enduring appeal is its ability to be molded into a workflow. As the available tools have evolved — both in their expression (e.g., languages such as Ruby, Python, Perl, and Lua), and in their representation (e.g., XML) — the decoupling of the typesetting engine from the rest of the toolset has enabled it to adapt and stay current.

Often, these workflows relate to publishing and the emphasis is on the form of the final output given a marked up input. TEX's programming capabilities further foster integrating workflow solutions tightly to the document.

The problems that I deal with in my work, however, typically have less to do with the final output and more to do with upstream processing. Chief among these is the issue of project estimation.

The challenge with project estimation is that its evaluation is not a straightforward derivation. It requires exploring different solutions, adjusting costing parameters iteratively, and, yes, even judgement based on past experience to come up with a reasoned, if not always reasonable, attempt at an estimate. In essence, we seek the capability to run a set of guided "what-if" scenarios.

In the end, though, one does need to represent this estimate in a form that is meaningful to both the supplier whose very viability relies on it, and the customer who will use it to help decide to whom the project will be awarded. So the desired outcome is still a well-presented document. It is the steps leading up to it, though, that are the focus of this article.

2 A specific example to illustrate the generic problem

To help illustrate the nature of the problem, let's begin with a specific estimation problem: develop a cost estimate for a lighting control system: one that covers both hardware and labour, and encompasses design, cabling, control componentry, installation, etc. I will explain this in some detail so that the knottiness of the issues might be exposed. As specific as this example is, however, the estimation problem is a generic one and many aspects are shared across domains.

Everyone is familiar with light switches and dimmers in their own homes, so that is a good place to start. Architectural designs show these on their drawings connected to light fixtures, e.g., pot lights, pendants, surface mount lights, under-cabinet lights, etc. Imagine if to this mix we add:

- scene based keypads
- control processors
- power modules
- contact closure modules
- occupancy and vacancy sensors
- RS-232 interfaces to other equipment

Clearly, the complexity of the design grows immensely. From an estimation perspective, though, even this complexity is tractable, anchored as it is to physical aspects of the design. So far the engineering is contained.

The complexity runs quickly ahead, though, when we have to confront the haze of the gestation period during which projects are formed, and when details are decidedly lacking. During this time, it is easy to become overwhelmed with the fractured shards of the project definition. Amongst these are:

- Varying granularity of the known scope. In the case of our lighting control system, for instance, we may not know what types of light fixtures are to be used, nor a breakdown on a room-by-room basis. But we might have a rough count of the total number of light loads whose detailed decomposition will evolve during the design. To use a software development analogy, we are talking here about stubs for routines which will be fleshed out later.
- Blended designs utilizing existing and new components. We need only to include the new components in the estimate, but need to ensure fit and compatibility with existing components.
- Staged implementations requiring that the design anticipate, at the onset, the expected capacity of the overall design. The design needs to be comprehensive, but the estimate need concern itself with only the most immediate phase of the project.
- Interdependencies between components, i.e., a selected component requires a host of other components in order to function. As an example, a

wireless keypad or switch would require the presences of a wireless network to communicate with the main processor. Otherwise, its inclusion in a materials list is not meaningful.

- Tiered product solutions. Often, component manufacturers will have tiered groups of solutions with overlapping applications to a specific problem. From an estimating perspective it is good to run these alternative solutions against a common problem specification to see how the solution differs in cost and capability.

3 Typical solutions

Keep in mind, though, that the requirement to produce a budget does not wait for a full and final specification. We must produce budget estimates with varying degrees of confidence during the entire design period.

There are two typical approaches to the estimation problem, especially when they relate to a specific vendor's equipment: either use their design software to create an equipment list, or their estimation software to create broad-brushed budgets. Neither approach is satisfactory.

The design software has several shortcomings:

- It is quite laborious. Often, it can take days to create a design.
- With the absence of detailed specifications during the early stages of the project, it is difficult to capture this uncertainty in the design. The design software assumes a final or near final specification.
- The software, oriented as it is towards a design, is specific to a single solution set. So from the same company we might have multiple pieces of software, one per tiered solution, into which the design has to be entered repeatedly.
- There is no way to mark components as existing even when the design is to be blended with existing parts.
- There is no way to capture items outside of the vendor's portfolio, e.g., wire and cable costs cannot be captured inside the design software. This leads to manual tracking of material and labour costs.

The estimation software, which amounts to a simple spreadsheet, also suffers from the following:

- Most importantly, this approach ties pricing, part numbers, and quantities to the design. If the pricing changes, or if parts are added to the vendor's portfolio, one is left to migrate the design from one spreadsheet to the next — an approach that is error-prone at best.
- The template to capture the design is rudimentary and requires specific quantities of components. One is left to aggregate these outside of the estimation software.
- It does not have the logic to detect interdependencies between components.

4 A workflow that works

The estimation problem kept me up for several nights and also led me to the mirthful twin title of my talk at TUG 2012: Sleep De(p)rived Typesetting. Sleep deprivation is always a powerful motivator when seeking out a solution. The secondary title stems from the acronym that I gave to the workflow that I assembled: YAWN. This stands for its constituent components:

- YAML (YAML Ain't Markup Language)
- Algebra
- Words
- Numbers

In struggling with the software solutions offered from vendors, I yearned for greater flexibility. To my mind, if the vendors were simply going to package spreadsheets, why couldn't they do so with the (say) incredible capabilities of Lotus Improv — a superb spreadsheet product developed for NeXTSTEP — now sadly relegated to history.

The Wikipedia entry for Improv only reinforced my unease regarding existing solutions. Pito Salas, the lead developer of Improv, expressed it succinctly:

> ... it became clear that the data, views of the data, and the formulas that acted on that data were separate concepts. Yet in every case, the existing spreadsheet programs required the user to type all of these items into the same (typically single) sheet's cells.

What I was seeking was to decouple the specification from the materials list and the materials list from the budget estimate with its detailed bill of materials and labour costs.

Pito's words pointed me to the natural object-oriented framework: Model-View-Controller, or MVC. That is, if one expressed the specification in a human-readable form, and used a *controller* to analyse this specification *model* along with component and pricing *models* for alternative solutions, one could then produce a *view*, which would satisfy both the supplier and customer in providing a meaningful document expressing the estimate.

I began with XML as a representation of the model, but decided that I didn't need all of its features. Instead, I chose to use the simpler syntax of YAML, a data serialization language which allows for key-value pair hashes as well as arrays, both of which

I did need. The implementation of the controller in Ruby grew directly from the selection of YAML as the model language, since Ruby is able to read and write YAML directly and easily. I should note that there are YAML wrappers for many other languages as well.

Here is a small example of a design specification, to give a flavour of the YAML representation:

```
:specification:
    - :area:
        :name: F0
        :sections:
            - :room:
                :name: Common Areas
                :design:
                    :loads:
                        - :lightload:
                            :type: :mlv
                            :fixturewattage: 50
                            :fixtureqty: 10
                            :qty: 70
                    :controlstations:
                        - :gangbox:
                            - :keypad:
                                :qty: 10
```

Entries preceded with a dash indicate elements of an array, while keys that are surrounded with colons might contain either single or compound elements, and tokens prefixed with colons indicate constants. Because it is a simple text file, one is easily able to track the development of the design by using version control software. Additionally, since the structure of the document is free-form, one is free to interject key–value pairs as comments or secondary information that might be ignored during processing but is still valuable when reviewing the specification.

The controller first takes a specification and apply business logic to create a materials list. This intermediate form was also expressed in YAML, again allowing for visual inspection. In some cases, a request for pricing is given already as a predefined materials list with no design specification required. By having this intermediate form as a text representation, one is able to create it directly as an input to the remainder of the workflow. As a secondary step this materials list was combined with a pricing model also expressed in YAML to create a complete bill of materials. Different controllers and/or pricing models may be used against a common specification to explore their impact on the generated estimate.

Which leads me to the *view*. I believe that if we consider TEX as a toolset that can produce views on demand, we have rounded out the framework into a workable form.

In my estimation workflow, TEX's capabilities are used only lightly — in essence to convert tabular data of the bill of materials into formatted output — but as the document requirements grow, I can easily enhance the output into more pleasing forms. I also envision doing a summary document that creates an estimation `diff` that compiles and compares the costs of alternative product solutions.

5 Conclusions

TEX works very effectively in the standard object-oriented Model-View-Controller framework. By isolating specification and the control logic from final document production in the manner prescribed by the MVC framework and its associated design patterns, TEX can be of tremendous value in enabling workflows outside of the domain of publishing.

References

[1] Oren Ben-Kiki, Clark Evans, and Ingy. YAML Ain't Markup Language (YAML) version 1.2. `http://www.yaml.org/spec`, October 2009.

[2] Trygve Reenskaug. MVC. `http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html`.

[3] Wikipedia. Lotus Improv. `http://en.wikipedia.org/wiki/Lotus_Improv#ATG`.

◇ Pavneet Arora
`pavneet_arora (at)`
`bespokespaces dot com`
`http://blog.bansisworld.org`

Star TEX: The Next Generation

Didier Verna

Abstract

While TEX is unanimously praised for its typesetting capabilities, it is also regularly blamed for its poor programmatic offerings. A macro-expansion system is indeed far from the best choice in terms of general-purpose programming. Several solutions have been proposed to modernize TEX on the programming side. All of them currently involve a heterogeneous approach in which TEX is mixed with a full-blown programming language. This paper advocates another, homogeneous approach in which TEX is first rewritten in a modern language, Common Lisp, which serves both at the core of the program and at the scripting level. All programmatic macros of TEX are hence rendered obsolete, as the underlying language itself can be used for user-level programming.

Prologue

TEX
The `[final]` *frontier.*
These are the voyages,
Of a software enterprise.
Its continuing mission:
To explore new tokens,
To seek out a new life,
New forms of implementation...

1 Introduction

In 2010, I asked Donald Knuth why he chose to design and implement TEX as a macro-expansion system rather than as a full-blown, procedure-based, programming language. His answer was twofold:

1. He wanted something relatively simple for his secretary who was not a computer scientist.
2. The very limited computing resources at that time practically mandated the use of something much lighter than a complete programming language.

The first part of the answer left me with a slight feeling of skepticism. It remains to be seen that TEX is simple to use. Although it probably is for simple things, its programmatic capabilities are notoriously tricky. The second part of the answer, on the other hand, was both very convincing and arguably now obsolete as well. Time has passed and the situation today is very different from what it was 30 years ago. The available computing power has grown exponentially, and so have our overall skills in language design and implementation.

Several ideas on how to modernize TEX already exist. Some of them have actually been implemented. In this paper, we present ours. The possible future that we would like to see happening for TEX is somewhat different from the current direction(s) TEX's evolution has been following. In our view, modernizing TEX can start with grounding it in an old yet very modern programming language: Common Lisp. Section 2 clarifies what our notion of a "better", or "more modern" TEX is. Section 3 on the next page presents several approaches sharing a similar goal. Section 4 on the following page justifies the choice of Common Lisp. Finally, section 5 on page 205 outlines the projected final shape of the project.

2 A better TEX

The TEX community has this particularity of being a mix of two populations: people coming from the world of typography, who love it for its beautiful typesetting, and people coming from the world of computer science, who appreciate it for its automation and similarity to a programming language. It is true that the way TEX works is much closer to that of a compiled language than a WYSIWYG editor.

In both worlds, TEX is unanimously acclaimed for the quality of its typesetting. This shouldn't be surprising, as it has always been TEX's primary objective. The question of its programmatic capabilities, however, is much more arguable. People unfamiliar with programming in general easily acknowledge that TEX's programmatic interface is not trivial to use. For people coming from the world of computer science, it is even more obvious that TEX is no match for a real programming language, partly due to its macro-expansion nature.

Let us recall that TEX was not originally meant to be a programming language. To quote its author, Donald Knuth [8]:

> I'm not going to design a programming language; I want to have just a typesetting language. [...] In some sense I put in many of TEX's programming features only after kicking and screaming.

From this perspective, it seems natural to consider that a "better" TEX, today, would essentially deliver the same typesetting quality from a more modern programmatic interface. More precisely:

- access to the typesetting subset of TEX's primitives should be provided with a consistent syntax (in particular, no more need for `\relax`),
- the programmatic API (`\def`, `\if`, *etc.*) should be dropped in favor of support from a real programming language,

- the system should remain simple to use, at least for simple things, just as TEX is today,
- the system should remain highly extensible and customizable, just as TEX is today,
- and finally, preserving backward compatibility, although not considered mandatory, should also be considered.

3 Alternatives

The idea of grounding TEX into a real programming language is not new. Let us mention some such attempts that we are aware of. `eval4tex`[1] and sTEXme[2] both use Scheme (another dialect of Lisp). PerlTEX [13, 15], as its name suggests, chooses Perl. QaTEX/PyTEX [4] use Python, and finally, also as its name suggests, LuaTEX[3] employs Lua.

These approaches, although motivated more or less by the same general idea, work in very different ways. Some of them wrap TEX in a programming language by giving the language access to TEX's internals. Others wrap a programming language in TEX by allowing TEX to execute code (LuaTEX belongs to this category). Some do both. For instance, sTEXme ships with a extended Scheme engine that can access TEX's internals, as well as a modified TEX engine that can evaluate Scheme code.

Some of them allow authors to write TEX macros in a different language (PerlTEX lets you write TEX macros in Perl). Others, like QaTEX, aim at completely getting rid of TEX macros so that all programmatic functionality is written in another language (Python, with PyTEX in that case). This particular case is closer to what we have in mind.

Finally, some approaches use a synchronous dual-process scheme in which both TEX and the programming language of choice run in parallel, communicating either via standard input/output redirection, or by file or socket I/O. That is the case of sTEXme and PerlTEX. Others, like `eval4tex` use a multi-pass scheme instead. In a first pass, the "foreign" code is extracted and sent to the programming language. The programming language in question executes its code and sends it back to TEX. In the final pass, TEX is left with only regular TEX macros to process.

In spite of all these variations, it is worth stressing that all these approaches have something in common: they are *heterogeneous*. They involve both a programming language engine on one hand, and the original, though possibly modified, TEX engine on the other hand. Even LuaTEX which is somehow more integrated than the other alternatives is built like this: a Lua interpreter is embedded in a more or less regular TEX, written in WEB and C.

The idea that we suggest in this paper is that another, fully integrated approach is also possible. In this approach, another programming language would be used to completely rewrite TEX *and* provide the desired programmatic layer at the same time. We are aware of at least one previous attempt at a fully integrated approach. NTS[4], the "New Typesetting System" was supposed to be a complete re-implementation of TEX in Java, but the project was never widely adopted. We don't think that this project's demise indicates in any way that the approach is doomed in general. On the contrary, the remaining sections explain why we think that Common Lisp is a very good candidate for it.

4 Common Lisp: why?

Lisp is a very old language. It was invented by John McCarthy in the late 1950s [9]. Contrary to what many people seem to think however, being old doesn't imply being obsolete. In this case, it is a synonym for being mature and modern. When Lisp was invented, it was in fact way ahead of its time, to the point that even its inventor hadn't realized the extent of his creation. A sign of this is the recent incorporation of features that Lisp already provided 50 years ago into so-called "modern" programming languages, such as C# with the addition of dynamic types, or C++ and Java with the addition of lambda (anonymous) functions.

4.1 Standardization

Common Lisp, in particular, was standardized in 1994 [1] (it was in fact the first object-oriented language to get an ANSI standard). Remember how stability was important to Donald Knuth for TEX? This means that even across different implementations (there are half a dozen or so), that the core of the language will *never* change, and in fact hasn't for the last 20 years. Compare this to modern scripting languages such as Python or Ruby, for which the "standard" is essentially *the* current implementation of *the* current version of *the* language written by *the* author of *the* language...

The Common Lisp standard is fairly comprehensive: it includes not only the language core but a large library of functions. Of course, because the standard is 20 years old, it lacks several things that are considered important today (such as a multi-threading API). Every Common Lisp implementation provides its own version of non-standard features, but again,

[1] `http://www.ccs.neu.edu/home/dorai/eval4tex/`
[2] `http://stexme.sourceforge.net/`
[3] `http://www.luatex.org`
[4] `http://nts.tug.org`

there are only half a dozen out there, and if one chooses to stick to only one of them, then one gets the same stability as for standardized features, or at least backward-compatibility.

4.2 General purpose *vs.* scripting

Common Lisp also has this particularity of being both a full-blown, general purpose, industrial scale programming language *and* a scripting or extension language at the same time. This is something that cannot be said of most modern languages out there but is nevertheless crucial in the fully integrated approach that we are advocating. This was certainly not the case of Java, in the now dead NTS project.

Common Lisp is indeed a full-blown, general purpose, industrial scale programming language. It is multi-paradigm (functional, object-oriented, imperative, *etc.*), highly extensible (both at the syntactic and semantic levels [12]), highly optimizable (notably with static typing facilities [19, 20]) and has a plethora of libraries (such as Perl-compatible regular expressions, database access, web infrastructure, foreign function interfaces, *etc.*). Today, millions of lines of Common Lisp code are used in industrial applications all over the world.

But Common Lisp is also a scripting language. It is highly interactive (it comes with a REPL: a Read Eval Print Loop), highly dynamic (with features ranging from dynamic type checking to an embedded JIT-compiler and debugger), highly reflexive (something crucial for extensibility and customizability) and at the same time easy to learn (notably out of its minimalist syntax).

This last point constitutes the beginning of our tour of the language. Remember that one of our objectives is to provide a system just as simple as TEX, at least for simple things. Below is a one minute crash course on Lisp syntax.

Literals Common Lisp provides literals such as numbers (`1.254`) or strings (`"foobar"`). Literals evaluate to themselves.

Symbols Common Lisp provides symbols that can be used to name functions or variables (possibly at the same time). `pi` has the expected mathematical value. `identity` is the identity function.

S-Expressions Compound expressions are written inside parentheses and denote function calls in prefix notation. `(+ 1 2)` represents the sum of 1 and 2.

Quotation In Common Lisp, everything has a value. If you want to prevent evaluation, put a quote in front of the expression. For instance, `'identity` is the symbol `identity` itself. `'(+ 1 2)`, instead of being a function call, now represents the list of 3 elements: the symbol `+` and the numbers 1 and 2.

Definitions To define a global variable, we can write something like this:

```
(defvar devil-number 666)
```

To define a function:

```
(defun dbl (x) (* 2 x))
```

And that is the end of our Common Lisp crash course. With that knowledge, you know practically all there is to know about the language in order to use it for simple things. The rest is a matter of knowing the names of standard (built-in) variables and functions. In particular, this is certainly not more complicated than learning the basic and inconsistent TEX syntax (do you provide arguments in braces or inline with a final `\relax` to be on the safe side?), and it would also certainly be enough to write basic LATEX documents like this:

```
(document-class article)
...
(begin document)
(section "Title")
...
(end document)
```

4.3 Built-in paradigms

In a somewhat surprising way, Common Lisp provides programming paradigms, idioms or library-based features that (LA)TEX also provide (or would like to provide). This means that such features are already here for us and would not need to be re-implemented in a Lispy TEX. This section only presents some of them; there are in fact many more.

4.3.1 `key=value` pairs

The success of key/value arguments in LATEX is proportional to their actual need: there are at least a dozen packages providing this functionality, each and every one of them with its own pros and cons. Common Lisp provides a clean and straightforward implementation of this for free in its function call protocol (the so-called "lambda lists"). The following function takes one mandatory (positional) argument and two optional *keyword* arguments, which are in fact named, floating arguments:

```
(defun include-graphics
  (file &key width height)
  ...)
```

It can be called with just the mandatory argument:

```
(include-graphics "image")
```

or with either or both keywords (prefixed with a ':'):

```
(include-graphics "image" :height <value>)
```

Keyword arguments can have default values which themselves may be dynamically computed.

4.3.2 Packages

LaTeX implements the notion of *packages*, essentially a collection of macros stored in a file. The *de-facto* standard for this in Common Lisp is called ASDF[5] (Another System Definition Facility). Common Lisp *systems* resemble LaTeX packages, only much more evolved. For instance, systems are composed of a hierarchy of files with customizable loading order, automatic dependency tracking and recompilation of obsolete object files (compare this to having only interpreted macros) and much more.

4.3.3 Namespaces

A frequent rant about LaTeX is the lack of namespaces. Common Lisp provides a related concept called *packages* (not to be confused with LaTeX packages). A package is a collection of symbols naming functions, variables, or both. Packages have names through which you access their symbols. Packages may declare some symbols as *public* while the others remain *private*. Suppose for instance that there is a package named `ltx`, that implements LaTeX 2_ε, and declares its function `document-class` as public. From the outside, one would canonically refer to this function as `ltx:document-class`. On the other hand, the need to reference all such LaTeX symbols explicitly in a document would probably be cumbersome. In such a situation, one may use `use-package`, in which case all public symbols become directly accessible. Using the `ltx` package makes it possible to call the function `document-class` implicitly, without the package name prefix.

4.3.4 Interactivity

As you likely know, TeX can be used in an interactive fashion. The following example shows a sample conversation between TeX and a user who mistypes a macro name:

```
didier(s003)% tex
This is TeX, Version 3.1415926 [...]
**\relax

*\hule
! Undefined control sequence.
<*> \hule

? H
The control sequence at the end of the top
line of your error message was never \def'ed.
If you have misspelled it (e.g., `\hobx'),
type `I' and the correct spelling
(e.g., `I\hbox'). Otherwise just continue,
and I'll forget about whatever was undefined.

? I\hrule

*\bye
```

This kind of interaction pretty much resembles a REPL which Common Lisp, like every interactive language, provides out of the box. Where Common Lisp specifically comes into play is that every Common Lisp application may embed a interactive debugger for free, precisely useful for this kind of error/recovery interaction. In Common Lisp, the programmer has the ability to implement his own recovery options (known as *restarts* in Lisp jargon) without unwinding the stack. This is different and much more powerful than regular catch/throw facilities. If you are not interested in using the full-blown debugger, you can implement a function for catching errors and listing the available recovery options in a bare 10 lines of code.

4.3.5 Dumping

Out of the historical concern for performance, TeX has the ability to dump and reload so-called "format" files, which saves a lot of parsing and interpretation (*cf.* the `\dump` command). Given the increase in computing power over the last 30 years, the question of performance is admittedly much less critical than it used to. Even today, though, performance is not a concern that should be completely disregarded. For example, compiling a lengthy Beamer presentation with lots of animations can still be annoyingly slow.

Common Lisp happens to provide a dumping feature out of the box. Although not part of the ANSI standard, all Lisp compilers provide it. In SBCL[6] for instance, the function `save-lisp-and-die` dumps the whole global state (stack excepted) of the current Lisp environment into a file that can later be quickly reloaded with the `--core` command-line option. Instead of dumping a core image, it is also possible to dump a fully functional standalone executable.

Because the dumping mechanism is accessible to the end-user, interesting applications could be envisioned with very little programming, such as mid-document dumping. By outputting to in-memory strings instead of files, for example, a document author could be given the possibility to dump in the middle of a large document's processing. The resulting facility would be quite similar to `\includeonly` — except that the whole document would be typeset every time.

[5] `http://www.common-lisp.net/project/asdf`

[6] `http://www.sbcl.org`

4.3.6 Performance

Let us tackle the problem of performance from a more general point of view. One frequent yet misinformed argument against dynamic languages is: "they are slow". From that point of view, it may seem odd to even begin to envision the reimplementation of a program such as TEX in a dynamic language.

One first and frequent misconception about interactive languages is that as soon as they provide a REPL, they must be interpreted. This is in fact not the case. Nowadays, many Common Lisp implementations such as SBCL don't even provide an interpreter. Instead, the REPL has a JIT (Just In Time) compiler which compiles the expressions you type and only then executes them. To put this in perspective, compare the processes of interpreting TEX macros by expansion and executing Lisp functions compiled to machine code...

Yet, starting with the assumption that performance should indeed be a concern (this is not even necessarily the case), the argument of slowness may still make some sense in specific cases. For example, it is obvious that performing type checking at run-time instead of at compile-time will induce a cost in execution speed. In general however, this argument, as-is, is meaningless. For starters, let us not confuse "dynamic language" with "dynamically typed language". A dynamic language gives you a lot of run-time expressive power, but that doesn't necessarily mean that you have to use all of it, or that it is impossible to optimize things away.

Let us continue on type checking in order to illustrate this. Look again at the definition for our "double" function:

```
(defun dbl (x) (* 2 x))
```

This function will no doubt be relatively slow, because Common Lisp has a lot of things to do at run-time. For starters, it needs to check that `x` is indeed a number. Next, the multiplication needs to be polymorphic because you don't double an integer the same way you double a float or a complex. That is in fact not the whole story, but we will stop here.

On the other hand, consider now the following version:

```
(defun dbl (x)
  (declare (optimize (speed 3) (safety 0))
           (type fixnum x)
  (the fixnum (* 2 x)))
```

In this function, we request that the compiler optimizes for speed instead of safety. The result is that the compiler will "trust" us and bypass all dynamic checks. Next, we actually provide static type information. `x` is declared to be a `fixnum` (roughly equivalent to integers in other languages) and so is the result of the multiplication. This is important because there is no guarantee that the double of an integer remains the same-size integer. Consequently, in general, Lisp would need to allocate a bignum to store the result.

As it turns out, compiling this new version of the function leads to only 5 lines of machine code. The compiler is even clever enough to avoid using the integer multiplication operator, but a left shift instruction instead. What we get in this context is in fact the behavior of a statically and weakly typed language such as C. Consequently, it should not be surprising that the level of performance we get out of this is comparable to that of equivalent C code. Recent experimental studies have demonstrated that this is indeed the case [19, 20].

This particular example is also a nice illustration of what we meant earlier by saying that Common Lisp is both a full-blown, industrial scale, general purpose programming language, *and* a scripting language at the same time. When working at the scripting level, the first version of `dbl` is quick and good enough. When working in the core of your application however, you appreciate it when the language provides a lot of tools (optimization ones notably) to adjust your code to your specific needs.

4.4 Extensibility and customizability

Another aspect of the language well worth its own section is its level of extensibility (adding new behavior) and customizability (modifying existing behavior). We know how important this is in the (LA)TEX world, which is a complicated ball of intermixed threads all interacting with each other [21], which wouldn't be possible without the level of intercession that TEX macros offer. Similarly, at least part of the success of LuaTEX is due to its ability to provide access to TEX's internals, so it seems that there is also a lot of interest in this area.

4.4.1 Homoiconicity and reflection

Once again, Common Lisp is here to help. We mentioned earlier how the root of extensibility and customizability in Common Lisp is its highly reflexive nature. Reflection is usually decoupled into *introspection* (the ability to examine yourself) and *intercession* (the ability to modify yourself).

In Lisp, reflection is supported in the most direct and simple way one could think of. Remember the expression `(+ 1 2)`, with or without evaluation? As we said before, this expression can either represent a call to the function "sum" with the arguments 1 and 2, or the list of three elements: the symbol `+` and

the numbers 1 and 2. What this really means is that every piece of code, if not evaluated, can be seen as a piece of data, and hence can be manipulated at will. In fact, every piece of Lisp code is represented as a list, which happens to be a user-level data structure. This property of a programming language is known as *homoiconicity* [5, 11].

Another important distinction in this notion is structural *vs.* behavioral reflection [10, 17]. While structural reflection deals with providing a way to reify a program, behavioral reflection deals with accessing the language itself. Lisp is one of the very few languages to provide both kinds of reflection to some extent, as we'll now discuss.

4.4.2 Structural reflexivity

This section gives only a couple of examples of structural reflexivity, again, to demonstrate how some well-known TEX idioms map to Common Lisp in a straightforward way.

The functional nature of Lisp implies that functions are first-class citizens in the language [3, 18]. In general, this means that functions can be used like any other object in the language. In particular, this means that functions can be modified, stored in variables, *etc.*

Storing a functional value in a variable will be useful in order to implement a variant of the function which needs to call the original one at some point. This is equivalent to the following common TEX idiom:

```
\let\oldfoo\foo
\def\foo{... \oldfoo ...}
```

Defining a function several times is simply equivalent to overriding the previous definition(s). This behavior matches that of `\def` or (more or less) `\renewcommand`. It is in fact more powerful for at least two reasons:

1. Since Common Lisp has a proper notion of scope (and in fact provides both dynamic and lexical scoping; something that very few other languages, can do), function or variable redefinition can be performed at different scoping levels, not only local or global.

2. Because of its interactive nature, there is no real distinction between functions defined in a core image or executable and those defined in the REPL and that consequently, they can be redefined in the exact same way. Consider what this really means for a minute: one could redefine *any* function in the TEX program just as easily as any TEX macro...

Finally, reflexivity in Common Lisp goes as far as allowing both introspection and intercession at the level of package internals. In most other languages, it is simply not possible to access the so-called "private" parts of a namespace, class, or whatever encapsulation scheme is supported. In Common Lisp, remember that a public symbol is accessed by prefixing its name with the name of the package and a colon separator, for example: `ltx:document-class`.

It turns out that it is just as easy to both introspect and intercede a package's internals. One just needs to use a double colon instead of a single one. This is not unlike the `@` character convention used by LATEX, along with the macros `\makeatletter` and `\makeatother`. The double colon really is a warning that you are prying on private property, but nothing technically prevents you from doing so.

4.4.3 Behavioral reflexivity

Lisp goes even further by providing some level of behavioral reflection as well.

Lisp macros (functions executed at compile-time) provide a form of intercession at the compiler level, allowing one to program language modifications in the language itself (what [16] calls a "homogeneous meta-programming system", as opposed to C++ templates for instance, which are heterogeneous: a different language).

CLOS [2, 6], the Common Lisp Object System is written *in itself*, on top of a so-called Meta-Object Protocol, known as the MOP [14, 7]. Using the CLOS MOP permits intercession at the object system level, allowing the programmer to modify the semantics of the object-oriented layer.

Finally, it is also possible to extend the Lisp syntax, which is a form of intercession at the parser level, allowing to modify the language's syntax directly. This is the only concrete example that we will provide in this section, although a striking one. We assume that the reader is familiar with TEX's double superscript syntax, allowing to denote characters that are not normally printable.

Suppose that we are given a function called `^^-reader` which performs TEX's double-superscript syntax to character conversion (this function is 10 lines long). The following code effectively installs the corresponding syntax in the Common Lisp reader:

```
(make-dispatch-macro-character #\^)
(set-dispatch-macro-character #\^ #\^
  #'|^^-reader|)
```

The first line informs the Lisp reader that the `^` character is to be treated in a special way (do you see a relationship to active characters and catcodes?). The second line informs the Lisp reader that if two

such characters are encountered in a row, then, the regular parser should stop and pass control to our user-provided function. We can now verify that this extended syntax works:

```
CL-USER> ^^M
#\Return
CL-USER> ^^00
#\Nul
```

This particular example is a bit simplified, but it conveys the idea. By modifying the way the Lisp parser behaves, we are able to modify the language itself, not only the program we are executing. And again, we are not very far from TEX's notion of active characters.

5 Common Lisp: how?

Section 2 on page 199 listed five objectives for a modern reimplementation of TEX. Section 4 on page 200 demonstrated how Common Lisp can help fulfill three of these goals: providing real programming capabilities, extensibility and customizability, all of this while maintaining a relative ease of use.

This section is devoted to the last two objectives, namely providing a more modern and consistent API while at the same time (although not mandatory) maintaining backward compatibility. In actuality, this section provides a more concrete view of the project itself. Although very little has been implemented already, the project *does* have a name: TiCL (the acronym for "TEX in Common Lisp").

5.1 API

Mapping the typesetting TEX primitives (that is, the non-programmatic ones) to Common Lisp would be rather straightforward.

- TEX parameters become Lisp variables. For instance, `\badness` is represented by a Lisp (global, dynamically scoped) variable `badness`.
- TEX quantities become Lisp objects. The term "object" is to be taken in a broad sense, that is, depending on the exact requirements, either an object of some class from the object system, of some structure, or anything else. In Lisp, it is customary to provide abstract constructor functions following a specific naming scheme. For instance, creating a TEX glue item could be done with a call to a function such as `make-glue`:

  ```
  (defun make-glue (b &key plus minus)
    ...)
  ```

  Since functions like this are bound to be used quite often, a syntactic shortcut may come in handy, such as the rather idiomatic one following, which also demonstrates the Lisp way to set some variable to a specific value (but see section 5.1.1):

  ```
  (setf baselineskip
        #g(b :plus x :minus y))
  ```

- Obviously, every TEX primitive command becomes a Lisp function. Again, the point here is to both simplify the syntax and make it more consistent at the same time (no more `\relax`!). Here are a couple of examples:

  ```
  (input file)
  (hbox material)
  (hbox material :to dim)
  (hbox material :spread dim)
  ```

  The reader familiar with TEX will notice immediately that the arguments in the last two examples are in reverse order, compared to the regular TEX versions. If this is really too much to get accustomed to, variants are easy to implement:

  ```
  (hbox-to dim material)
  (hbox-spread dim material)
  ```

  Such syntactic variants are usually implemented with Lisp macros, evaluated at compile-time, so that there is no additional run-time cost.

5.1.1 Lisp-2

Let us go back to the `baselineskip` assignment example for a minute:

```
(setf baselineskip <glue>)
```

This assignment may seem odd to a TEXnician, who is more accustomed to direct assignments such as

```
\baselineskip 10pt
```

In fact, TEX has this way of using the same macro for both denoting its value and setting it.

We intentionally omitted one point in section 4.3 on page 201 in order to put it here: the fact that Common Lisp is a "Lisp-2" (as opposed to Scheme for instance, which is a Lisp-1). What this means is that Common Lisp has 2 different namespaces for functions and variables. In other words, the same symbol can be used to both refer to a function and a variable *at the same time*.

An interesting consequence of this is that it is possible to define a *function* `baselineskip` the purpose of which is to assign a value to the eponymous *variable* `baselineskip`. Assuming this function exists, the above Lisp expression can hence be simplified as follows:

```
(baselineskip <glue>) ; set it!
```

Again, this aspect of Common Lisp brings us even closer to one of TEX's idioms: that of quantity assignment.

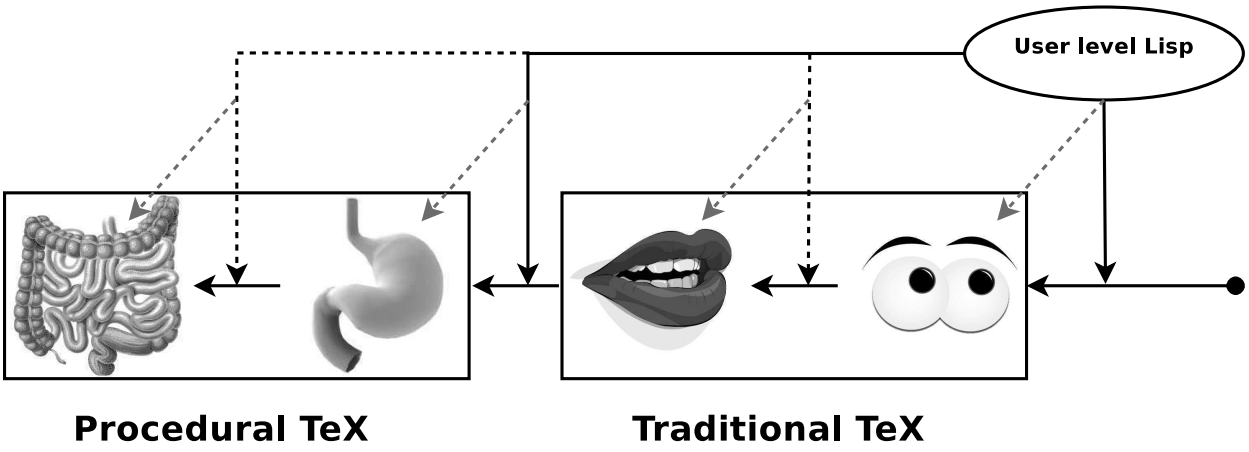


Figure 1: TiCL architecture

5.2 Backward compatibility

The programmatic interface to the typesetting part of TEX described in the previous section is what we call "procedural TEX". Along with the actual typesetting engine, this mostly corresponds to TEX's stomach and bowels. On top of that, it is in fact possible to design either completely new typesetting systems or programmatic versions of Plain TEX, LATEX, *etc.* entirely in Lisp.

In order to maintain backward compatibility, we also need to implement "traditional" TEX (still in Lisp), that is, the surface layer consisting of both the macro versions of Lispified TEX primitives, and the rest of TEX's programming API. This is mostly located in TEX's mouth and eyes. Figure 1 depicts this architecture (disclaimer: this is an overly simplified, extremely naive view; see section 5.3 for details). As mentioned earlier, the whole point of this architecture being implemented in Lisp, in terms of extensibility and customizability, is that the user of TiCL can basically interfere at every level of the system, whether by adding personal functions, rewriting built-in functions or even internal typesetting algorithms.

Another advantage of this fully integrated approach is that it is actually quite simple to provide "mixed" functionality, that is, using Common Lisp code directly in an otherwise regular TEX source file. The only requirement is an escape syntax allowing Common Lisp to take over interpretation of Lisp code, and insert the result back into the regular TEX character stream. A prototype for this has already been implemented. It simply consists in a Common Lisp implementation of TEX's eyes with an additional bit of syntax: the appearance of two consecutive and equal subscript characters in a regular TEX source file will trigger the evaluation of the subsequent Lisp expression. Since this syntax is invalid in TEX, it cannot break any existing document.

Below is an example illustrating this idea (taken from [15]). The Lisp function `ast` is used to define a TEX macro `\asts` outputting a specified number of asterisks.

```
\documentclass{article}

\newcommand\asts{}
__(defun ast (n)
    (format nil "\\renewcommand\\asts{~A}"
      (make-string n :initial-element #\*)))

\begin{document}
__(ast 10)
\asts
\end{document}
```

For the curious, our current implementation of TEX's eyes in Common Lisp is roughly 200 lines. The support for the double subscript syntax (including parsing it, reading the Lisp code, evaluating it and inserting the result back into the regular TEX stream) amounts to only 16 lines, that is, around 8% of the total.

5.3 Expected problems

After all those "would" and "could", let us get to a more pessimistic (realistic?) view of the project. This section sheds some darkness on the too-bright picture we have drawn of TiCL until now. As mentioned earlier, TiCL is in fact pretty much only an idea at present. If this project ever comes to fruition, a number of problems are expected.

5.3.1 A huge task

Completely reimplementing TEX is a huge task. This is probably one of the reasons all alternative projects

(NTS excepted) chose the hybrid approach instead of the fully integrated one. One thing that may help is the existence of foreign function interfaces, notably for the C language. The project could be developed gradually by linking to the WEB/C implementation of the not-yet-reimplemented parts.

5.3.2 Compatibility

Another question to consider is whether the TEX-incompatible part would eventually be accepted by *the* (or *a new*) community. This question is in fact pertinent for LATEX3 as well and we don't have an answer. The existence of a compatibility mode with pluggable Lisp, as described in section 5.2 on the facing page, would probably help getting people accustomed to the benefits of using Lisp, while remaining in a reassuring context of traditional TEX.

5.3.3 TEX's organs

Figure 1 on the preceding page was already pointed out to be overly simplified. In reality, we know that the TEX organs don't constitute a pipeline. Some levels from "down below" do affect the upper stages which makes things much more complicated. In the long run, this may imply that it would be impossible to have new programmatic typesetting systems (accessing "procedural TEX" directly) work in conjunction with traditional TEX. Currently, we don't know for sure, although we are aware of the fact that this was one of the reasons the NTS project failed.

5.3.4 Mixed mode

Along those same lines, "mixed" mode, that is, the ability to mix regular TEX macros with Lisp code is expected to be tricky. If we want to provide more interaction than just the double subscript syntax, for example, the ability to define TEX macros in Lisp with Lisp access to the macro's arguments, we are bound to encounter issues related to the time of expansion.

This problem is in fact not related to Lisp at all, but rather to *any* approach aiming to provide the ability to define TEX macros in another language. Section 4 of [15] describes these kinds of problems in more detail.

5.3.5 Sandboxing

As with any scriptable application, the security problem is an important one. How much programmatic access to the application itself, or its surrounding environment, are we willing to give the end-user? Some versions of TEX are equipped with means to address this issue (*cf.* the `-shell-escape` command-line option and the `\write18` command). Using a true, comprehensive programming language with scripting capabilities makes this problem even worse because we have more than a couple of "macros" to control access to. We have a whole stack of language features to control, such as file I/O, operating system interfaces, *etc.* Currently, we are not aware of any sandboxing library already available for Lisp.

5.3.6 Intercession

This is in a similar vein as the sandboxing problem. In [21], we were complaining about (rejoicing in?) the huge intercession mess that the LATEX world is. Well, now you should *really* be afraid because we are going to make things worse! Remember that any Lisp executable granting scripting privileges to the end-user effectively provides write-access to the whole executable. This surely makes it trivial to extend or customize the system. Whether this is a good thing for the system in the long run, there is no way to tell. After all, LATEX *is* alive and kicking, in spite of (because of?) its high intercession capabilities...

6 Conclusion

In this paper, we advocated a "homogeneous" approach to TEX modernization in which TEX itself is first rewritten in a programming language serving both at the core of the program and at the scripting level. All programmatic macros of TEX are hence rendered obsolete, as the underlying language itself can be used for user-level programming. In a rather puzzling way, our notion of "modernity" consists of reimplementing a 30-year-old language using a 50-year-old one!

We demonstrated why we think that Common Lisp is a very good choice for doing this. Common Lisp is both a scripting language and a full-blown, industrial scale, general-purpose programming language at the same time. Common Lisp also provides several paradigms or idioms that are actually quite close to features that TEX either provides as well, or would like to provide.

Will the TiCL project ever come to birth? That is not sure at that point, as it will require a tremendous effort. We would like to see it happening, of course. Amongst all the current alternatives, LuaTEX seems to be the only one vigorously alive and gaining momentum.

Of course, the other project that needs to be mentioned is LATEX3. We only do so in an anecdotal fashion because it does not make use of a real programming language, but continues in the tradition of building directly on top of TEX. Still, LATEX3 is not experimental anymore and is light years ahead of what LATEX 2_ε is. In particular, it is much better

than its ancestor in terms of syntax consistency and programmatic capabilities. Nevertheless, since it is still restricted to TEX's macro-expansion world, every time a new programming paradigm is needed, it will have to be implemented manually.

To sum up, the "big TEX modernization plan" currently seems to follow two different paths: staying strictly in the TEX area or creating hybrids of TEX and another real programming language. The approach we would like to suggest with TiCL is a third one: a homogeneous approach in which the implementation language of TEX is also the one which serves at the scripting level. Will this project see the light of day? Can these three approaches co-exist in the long term? Only the future will tell...

Epilogue

These were the voyages,
Of a software enterprise.
Its continuing mission:
To explore new tokens,
To seek out a new life,
New forms of implementation.
To `\textbf{go}`*,*
Where no TEX has gone before!

References

[1] Common Lisp. American National Standard: Programming Language. ANSI X3.226:1994 (R1999), 1994.

[2] Daniel G. Bobrow, Linda G. DeMichiel, Richard P. Gabriel, Sonya E. Keene, Gregor Kiczales, and David A. Moon. Common Lisp Object System specification. *ACM SIGPLAN Notices*, 23(SI):1–142, 1988.

[3] Rod Burstall. Christopher Strachey — Understanding programming languages. *Higher Order Symbolic Computation*, 13(1-2):51–55, 2000.

[4] Jonathan Fine. TEX forever! In *Proceedings EuroTEX*, pages 140–149, Pont-à-Mousson, France, 2005. DANTE e.V.

[5] Alan C. Kay. *The Reactive Engine*. PhD thesis, University of Hamburg, 1969.

[6] Sonya E. Keene. *Object-Oriented Programming in Common Lisp: A Programmer's Guide to CLOS*. Addison-Wesley, 1989.

[7] Gregor J. Kiczales, Jim des Rivières, and Daniel G. Bobrow. *The Art of the Metaobject Protocol*. MIT Press, Cambridge, MA, 1991.

[8] Donald E. Knuth. *Digital Typography*. CSLI Lecture Notes. CSLI, September 1998.

[9] John MacCarthy. Recursive functions of symbolic expressions and their computation by machine, part I. *Communications of the ACM*, 3:184–195, 1960. Online version at `http://www-formal.stanford.edu/jmc/recursive.html`.

[10] Patty Maes. Concepts and experiments in computational reflection. In *OOPSLA*. ACM, December 1987.

[11] M. Douglas McIlroy. Macro instruction extensions of compiler languages. *Commun. ACM*, 3:214–220, April 1960.

[12] Marjan Mernik, editor. *Formal and Practical Aspects of Domain Specific Languages: Recent Developments*, chapter 1. IGI Global, 2012.

[13] Andrew Mertz and William Slough. Programming with PerlTEX. *TUGboat*, 28(3):354–362, 2007.

[14] Andreas Paepcke. User-level language crafting — Introducing the CLOS metaobject protocol. In Andreas Paepcke, editor, *Object-Oriented Programming: The CLOS Perspective*, chapter 3, pages 65–99. MIT Press, 1993. Downloadable version at `http://infolab.stanford.edu/~paepcke/shared-documents/mopintro.ps`.

[15] Scott Pakin. PerlTEX: Defining LATEX macros using Perl. *TUGboat*, 25(2):150–159, 2004.

[16] Tim Sheard. Accomplishments and research challenges in meta-programming. In Walid Taha, editor, *Semantics, Applications, and Implementation of Program Generation*, volume 2196 of *Lecture Notes in Computer Science*, pages 2–44. Springer, 2001.

[17] Brian C. Smith. Reflection and semantics in Lisp. In *Symposium on Principles of Programming Languages*, pages 23–35. ACM, 1984.

[18] J.E. Stoy and Christopher Strachey. OS6 — An experimental operating system for a small computer. Part 2: Input/output and filing system. *The Computer Journal*, 15(3):195–203, 1972.

[19] Didier Verna. Beating C in scientific computing applications. In *Third European Lisp Workshop at* ECOOP, Nantes, France, July 2006.

[20] Didier Verna. CLOS efficiency:instantiation. In *International Lisp Conference*, pages 76–90, MIT, Cambridge, Massachusetts, USA, March 2009. Association of Lisp Users.

[21] Didier Verna. Classes, styles, conflicts: The biological realm of LATEX. *TUGboat*, 31(2):162–172, 2010. `http://tug.org/TUGboat/tb31-2/tb98verna.pdf`.

◇ Didier Verna
EPITA / LRDE
14-16 rue Voltaire
94276 Le Kremlin-Bicêtre Cedex
France
`didier (at) lrde dot epita dot fr`
`http://www.lrde.epita.fr/~didier`

Adapting ProofCheck to the author's needs

Bob Neveln and Bob Alps

Abstract

ProofCheck is a system for writing and checking mathematical proofs. Theorems and proofs are contained in a plain TEX or LATEX document. Parsing and proof checking are accomplished through Python programs which read the source file. Although the use of these programs has never been restricted to any particular logic or mathematical language, the work required to actually implement an author's choices in these matters, especially in the logic, and to make the necessary modifications of the supporting files have been sufficiently laborious as to pose an obstacle to the use of ProofCheck. This paper describes updates to the system whose purpose is to alleviate these labors to the extent possible so as to facilitate the use of ProofCheck in a logical and linguistic setting of the author's choice.

1 What is ProofCheck?

ProofCheck is a Python package, available at `http://www.proofcheck.org`. It checks mathematical proofs written in TEX or LATEX. Previous versions were described in [2] and [3]. The mode of operation of ProofCheck is extremely simple. It requires proofs to be structured into lines which have a justification at the right margin. Each line with its justification is matched against rules of inference from a list. If a match is found the line is checked. If all the lines of a proof check the proof checks.

This simplicity is reflected in the file structure of the system as shown in Figure 1. The rectangular boxes represent TEX files. The rule matcher and the unifier consist of Python code. We now briefly describe these TEX files.

1.1 (LA)TEX documents

An author's document may be written in either plain TEX or in LATEX. Figures 2 and 3 show a fictitious proof as it would appear in the source and output files respectively. Ordinary text may be used relatively freely in a proof. Mathematical expressions in TEX may also be included even if they play no role in the checking.

Structuring a proof so that it can be checked requires six proof structuring macros in TEX, five in LATEX. Figure 2 shows all those needed in LATEX. For plain TEX, these are used: `\chap`, `\prop`, `\note`, `\line[a-z]`, `\By`, `\Bye`. No other markup is needed in structuring proofs.

The logical and mathematical symbols shown

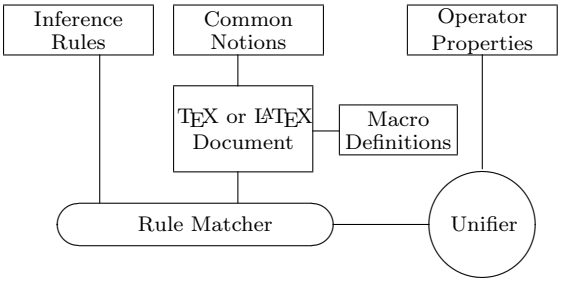


Figure 1: Main Proof Checking Files

```
\prop 4.23 $\Not (x,y,z \in \nats \setdif \{0\}
     \And 2 < n \in \nats
     \And x \toThe n + y \toThe n = z \toThe n)$
\lineb Proof:  We begin by taking as given:
\note 1 $(x,y,z \in \nats \setdif \{0\})$  \By G
\linea and
\note 2 $(2 < n \in \nats)$ \By G
\linea From these givens we get our contradiction that
\note 3 $(x \toThe n + y \toThe n = z \toThe n $
\lined $ \c \false)$ \By .1,.2
\lineb \Bye .3 H .1,.2
```

Figure 2: Fictitious Proof: Source File

4.23 $\neg(x, y, z \in \mathbf{N} \backsim \{0\} \wedge 2 < n \in \mathbf{N} \wedge x \hat{} n + y \hat{} n = z \hat{} n)$

Proof: We begin by taking as given:

.1 $(x, y, z \in \mathbf{N} \backsim \{0\})$ ‡G

and

.2 $(2 < n \in \mathbf{N})$ ‡G

From these givens we get our contradiction that

.3 $(x \hat{} n + y \hat{} n = z \hat{} n \rightarrow \perp)$ ‡.1,.2

Q.E.D. .3 H .1,.2

Figure 3: Fictitious Proof: Output

in the fictitious proof are those used in the rules of inference and common notions files, respectively. In sections 2.1 and 2.2 it is shown how to adapt ProofCheck so that it works with symbols of the author's choosing.

1.2 Macro definitions files

There are TEX definition files with extension `.tdf` or LATEX definition file with extension `.ldf` that need to be included at the beginning of a checkable source file using a TEX `\input` statement. These include the `utility.tdf` file which contains the proof structuring macros and sets up some math fonts and the `common.tdf` file which contains the TEX definitions of the symbols used in the rules of inference and common notions files. Further, any document which contains many new symbols should also have its own `.tdf` file. Besides TEX macros these files may also contain ProofCheck directives, which are TEX

comments beginning with '%' allowing authors to customize the behavior of the system:

ProofCheck directives

- Symbol substitution:
  `%def_symbol \forall \Each`
- Operator precedence specification:
  `%set_precedence \toThe 17`
- External file referencing:
  `%set_ref gr  graphs`
- Primitive term and formula specification:
  `%undefined_term: supremum`
- Term and formula definor specification:
  `%term_definor: =`
- LaTeX theorem section-level specification:
  `%major_unit: section`

1.3 Rules of inference file

The rules of inference file is a TeX file which is searched every time a line of a proof is checked. It consists of logical rules of inference.

The prime example of a rule of inference is the modus ponens rule. In the rules file it looks like this:

$$(\underline{p} \to \underline{q}) \ ; \ \underline{p} \vdash \underline{q}$$

In order for checkable proofs to be of reasonable length it is important that there be many rules. We now have over 1500 in the default `rules.tex` file. This file which has been supplied by default with the package is based on the logic in [1], which is not in wide use. The size of this file makes it difficult to simply edit desired changes. In section 2.5 an improvement is described which makes it possible to obtain a variety of standard and non-standard rules of inference files.

1.4 Operator properties file

When the rules of inference file is searched each rule is compared with the line to be checked using a unifier. A unifier is a program which determines whether there is some substitution which makes two given expressions the "same", which ordinarily means identical. But it is important to allow expressions such as $((A \wedge B) \wedge C)$ and $(A \wedge (C \wedge B))$ to be considered the same.

The unifier may assume that certain operators, such as conjunctions, are commutative and associative, or transitive, such as equality, if theorems to this effect are stored in the `properties.tex` file. This file can be edited and such theorems may be commented out if desired for the logic under consideration.

1.5 Common notions file

The common notions file consists of mathematics which is either taken for granted, or at least is outside the scope of the document.

2 Adapting ProofCheck

It is very desirable that an author be able to check mathematics without being required to use the same symbols as those in the rules of inference and common notions files.

2.1 Example of a symbol definition

In the source file shown in Figure 2, the macro `\setdif` produces the symbol '$\backsim$' in the output file shown in Figure 3. To obtain '$\mathbf{N} \setminus \{0\}$' instead of '$\mathbf{N} \backsim \{0\}$' in the output, while still using '`\setdif`' in the source file one can use the TeX definition:

```
\def\setdif{\setminus}
```

This modification affects only the output file and does not free the author of the need to match the symbol in the source file with the corresponding symbol in the rules or common notions files.

2.2 Example of a symbol substitution

Unlike symbol definition, symbol substitution allows an author to use a freely chosen symbol in the source file without forgoing a match with ProofCheck files. To use '`\setminus`' (for '\') in the source file and still match '`\setdif`' in the ProofCheck files an author could use the following symbol substitution:

```
%def_symbol \setminus \setdif
```

In cases where the issue can be resolved by a symbol-for-symbol replacement either a symbol definition or a symbol substitution can solve the problem. In propositional logic for example, since notation is almost universally infix, symbol-for-symbol replacements are enough. Since the rules of inference file consists of logic, and quantifiers like propositional logic share a common syntax, syntactical agreement with the rules of inference file can almost always be accomplished with such replacements. Semantic issues are discussed in section 2.5.

2.3 Setting operator precedence

Infix operators are constants which, whether logical or mathematical, require precedence values in order to avoid fully parenthesizing. To establish the precedence of a new infix operator, or reset that of an old one, a line is inserted into a macro definitions file. To set the precedence of the exponential operator used in Figure 2 to 17 the following line suffices:

```
%set_precedence \toThe 17
```

The `viewdfs` script may be run to see all currently established precedence values.

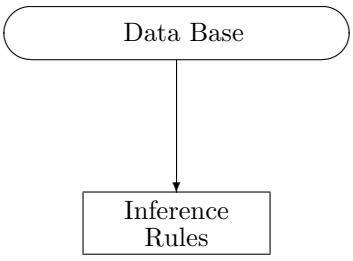


Figure 4: Rules File and Database

2.4 New terms and formulas

Much of the work of making mathematical proofs checkable lies in constructing definitions for the terms and formulas used in the work, which satisfy the requirements of formality. Definitions recognized by ProofCheck must have the form:

(⟨*definiendum*⟩ ⟨*definor*⟩ ⟨*definiens*⟩)

Term and formula definor symbols can be established using the directives:

```
%term_definor:
%formula_definor:
```

The parser then "learns" the author's terms and formulas when it reads a definition. A definition should be marked using `\prop` and given a `\By D` justification.

Terms and formulas may also be presented to the parser without definitions using ProofCheck directives:

```
%undefined_term:
%undefined_formula:
```

2.5 Rules database

The rules of inference used in the original implementation of ProofCheck are those of a logic system consisting of the sentence logic of tautologies and a free predicate logic which allows non-denoting terms and includes definite and indefinite descriptions. Recognizing that most potential users work with more traditional logic, ProofCheck has been modified to allow a user to select a rules of inference file which implements the user's preferred logic system. A database has been built to store rules of inference, where each rule is flagged with respect to the 20 different attributes shown in Figure 5. Rules have been added to the database which are valid only in more traditional logic. The user can create a query to select rules based on these attributes. We have identified 8 logic systems for which queries have been pre-defined. Each of the 8 systems uses tautologies for sentence logic, but differ in the area of predicate logic. The systems are as follows:

Sentence logic

- tau - tautology rule
- trf - rule contains True symbol (⊤) or False symbol (⊥)
- ent - entailment rule (relevance and necessity)
- tui - intuitionistic rule
- mdl - contains a modal operator

Predicate logic

- prd - predicate logic rule (contains a predicate or quantifier)
- std - standard predicate logic rule
- fre - free logic rule
- uni - universal logic rule
- equ - contains the equals symbol (=)
- idn - contains the identity symbol (≡)
- def - contains definite description (the)
- idf - contains indefinite description (an)
- cas - contains case symbol (◊)
- nul - contains the Nul symbol (Nul)
- exs - contains the exists predicate symbol (ex)
- unv - universalization rule

Miscellaneous

- prn - has parentheses as reference punctuators
- mlt - multi-goal rule
- gvh - Given : Hence rule

Figure 5: Attributes of Inference Rules

2.5.1 Eight logic systems

Here, SPL means Standard Predicate Logic, and FPL means Free Predicate Logic.

1. SPL without equality or descriptions (Kelley, Morse)
2. SPL with equality (Suppes)
3. SPL with definite descriptions (Bernays)
4. SPL with indefinite descriptions (Bourbaki)
5. FPL without identity, equality, or descriptions
6. FPL with identity and equality
7. FPL with definite descriptions
8. FPL with indefinite descriptions (Alps–Neveln)

2.5.2 Query examples

For example, to produce the rules of standard predicate logic with equality and without descriptions the following query is used:

```
(mdl=0 And (tau=1 Or (std=1 And def=0)))
```

As another example, to produce rules for Alps–Neveln logic, the needed query is:

```
(mdl=0 And (fre=1 Or tau=1))
```

2.6 The common notions file

Using a different logic affects the validity of theorems compiled in the common notions file. Therefore, these theorems must be reviewed and modified to suit the logic being used.

The work of the authors uses a set/class theory similar to that of Morse and Kelley, and this is reflected in the theorems of the common notions file. The use of a different set theory, such as Zermelo–Fraenkel, would require that the common notions be modified for consistency with the chosen set theory. Work is underway to create a common notions file compatible with SPL with definite descriptions and Zermelo–Fraenkel set theory.

2.6.1 External references in proofs

External references to the common notions file can be identified by a leading '`0`'. In the marginal justification of the following note, a theorem numbered 11.7 in the common notions file is referred to:

.12 $(x \in A \rightarrow x \in B)$ ‡011.7; .10

Multiple external reference files are permitted with an appended identification code established using a ProofCheck directive. The command

```
%set_ref gr  graphs
```

causes ProofCheck to refer to the file `graphs.tex` when references using `gr` are used as in the justification of the following note:

.12 $(x \in A \rightarrow x \in B)$ ‡11.7gr; .10

3 Conclusion

A widely-held view is that formal proofs are by necessity lengthy and intractable. A fairly recent logic text, for example, claims that:

> In principle, all of known mathematics can be formalized in terms of the symbols and axioms. But in everyday practice, most ordinary mathematicians do not completely formalize their work; to do so would be highly impractical. Even partial formalization of a two-page paper on differential equations would turn into a 50 page paper. For analogy, imagine a cake recipe written by a nuclear physicist, describing the locations and quantities of the electrons, protons, etc., that are included in the butter, the sugar, etc.[1]

Existing proof assistants[2] tend to support this notion, conveying the impression that "computer proofs" are massive, and the packages themselves tend to be massive.

But as in [2] and in [3] we again claim that checkable proofs can be done which are less than an order of magnitude longer than proofs which are considered rigorous by prevailing standards. We seek mathematicians interested in trying ProofCheck. We will gladly provide assistance to anyone seeking to use it.

References

[1] Robert A. Alps and Robert C. Neveln. A predicate logic based on indefinite description and two notions of identity. *Notre Dame Journal of Formal Logic*, 22(3), 1981.

[2] Bob Neveln and Bob Alps. Writing and checking complete proofs in TEX. *TUGboat*, 28(1), 2007, pp. 80–83. `http://tug.org/TUGboat/tb28-1/tb88neveln.pdf`.

[3] Bob Neveln and Bob Alps. ProofCheck: Writing and checking complete proofs in LATEX. *TUGboat*, 30(2), 2009, pp. 191–195. `http://tug.org/TUGboat/tb30-2/tb95neveln.pdf`.

[4] Eric Schechter. *Classical and Nonclassical Logics*. Princeton University Press, Princeton, New Jersey, 2005.

◇ Bob Neveln
Widener University
`neveln (at) cs dot widener dot edu`

◇ Bob Alps
Evanston, Illinois
`BobAlps (at) aol dot com`

[1] The quoted text is on page 28 of [4].

[2] See Wiedijk's list: `www.cs.ru.nl/~freek/digimath`

Approaching Asymptote

Michael Doob and Jim Hefferon

Asymptote is a relatively new tool to make graphics that is integrated with the TEX family. On its website[1] its developers, Andy Hammerlindl, John Bowman, and Tom Prince, characterize it as "a powerful descriptive vector graphics language inspired by METAPOST that features robust floating-point numerics, automatic picture sizing, native three-dimensional graphics, and a C++/Java-like syntax enhanced with high-order functions." It is free software, released under the GNU General Public License.

Those developers have described Asymptote's advanced capabilities and algorithms in several papers (for instance, [1], [3], and [4]) and presentations (see [2]). The comprehensive manual and additional documentation is on the website. Also check out Philippe Ivaldi's wonderful gallery and tutorial.[2]

We are not developers, we are users (specifically, mathematical users). This is a gentle introduction aimed at people who need to produce mathematically-oriented graphics, and who may find that this system fits their needs and how they think.

We will first briefly compare Asymptote with METAPOST, since that program may be familiar to readers. We will then introduce capabilities that are basic to Asymptote by using some figures, chosen both to be close to what a person might produce in everyday work and to illustrate the power of the system. We will finish by comparing this system with two others that have many of the same capabilities and are widely used in the TEX community, Ti*k*Z and PSTricks.

1 Tangent: Predecessor systems

METAFONT is a programming language written by Donald Knuth to define the fonts for TEX. METAPOST[3] is an extension of METAFONT targeted at PostScript output. It remains widely used for graphics for mathematics and related fields, and it is under active development.

METAPOST produces two-dimensional line art of very high quality. A number of innovative algorithms are built in. Its output is vector-based, not raster — that is, not dot-by-dot — so making a graphic larger or smaller is smooth. One of its key strengths is solving systems of equations. For instance, the language allows you to easily find the intersection of two lines, so if you label an intersection and later adjust one line a bit then the label moves with the intersection. METAPOST is integrated with TEX so that, for instance, you can produce labels that use the LATEX style of the target document.

The most important point about METAPOST (and Asymptote also) is that it is not mouse-driven. Instead, you write a program to produce the output. Below we will discuss some advantages of this — how this allows us to construct figures in a way that fits with our training — but one disadvantage is that because the syntax of the METAPOST language is unlike most other languages, users can find awkward switching from programming in more everyday languages to programming in this one.

2 Meeting Asymptote: Two dimensions

Asymptote has essentially all of the capabilities of METAPOST. Its syntax is Java-like. Asymptote has a more extensive built-in function set than METAPOST. And it comes with many add-on modules (METAPOST is relatively weak in this area). In the next section we will highlight one add-on for drawing in three dimensions but first we focus how Asymptote does at METAPOST's strength, two-dimensional line art. Since Asymptote develops on those capabilities it too makes those drawings with ease.

For a first taste we will walk through making an elementary school star, as shown. The file `star.asy` contains this Asymptote code.

```
size(.5inch);
path star;
for(int i=0; i < 5; ++i)
  star=star--dir(90+144i);
star=star--cycle;
draw(star);
```

For the first line, Asymptote expects that you typically want to specify the size of the graphic, and we've specified that its width is a half inch. In the second through fifth lines we construct the path with five line segments (as with METAPOST the `--` operator connects points with line segments while `..` makes Bezier curves) and close the path with `cycle`. Finally we draw that path to the output picture (the default line width is 1 PostScript point).

We compile that code to a graphic by running it through Asymptote. You should be able to try this also because there are versions for GNU/Linux, Mac, or Windows. We use the executable that comes with Ubuntu Linux but the commands below should work anywhere. The command line

```
$ asy star
```

produces the output file `star.eps` in Encapsulated PostScript. You can get just about any graphics format, such as PDF.

```
$ asy -fpdf star.asy
```

[1] `http://asymptote.sourceforge.net`
[2] `http://www.piprime.fr/asymptote`
[3] `http://www.tug.org/metapost.html`

LaTeX can use this output with the regular `graphicx` command

```
\includegraphics{star.pdf}
```

or you can instead embed the Asymptote code inside your LaTeX file and the graphic will automatically be inserted; see the documentation for that.

With that first taste of the system we can begin to go further. Asymptote has four basic operations that we will illustrate with four small listings.

The `draw` command is first. This `.asy` file

```
size(35pt);
path star;
for(int i=0; i < 5; ++i)
  star=star--dir(90+144i);
star=star--cycle;
draw(star,linewidth(2)+lightblue+beveljoin);
```

draws this star.

The `draw` command has many options. For instance, here we are drawing with a pen 2 points wide, in blue, and with line segments joined in a bevel.

Asymptote's second basic operation, `fill`, colors in a closed path.

```
size(35pt);
path star;
for(int i=0; i < 5; ++i)
  star=star--dir(90+144i);
star=star--cycle;
fill(star,lightblue);
```

Here we `fill` the star with blue.

Asymptote's third basic operation is `clip`. It omits from a shape the part that does not fit in the clip-to area. For instance, we can drop the parts of the star that lie outside a circle.

```
size(35pt);
path star;
for(int i=0; i < 5; ++i)
  star=star--dir(90+144i);
star=star--cycle;
fill(star,lightblue);
clip(scale(0.618)*unitcircle);
```

(Comparing this star to the prior one shows that after clipping Asymptote has expanded the shape to fit in the declared width.)

The fourth basic operation, `label`, brings in TeX text.

```
size(35pt);
path star;
for(int i=0; i < 5; ++i)
  star=star--dir(90+144i);
star=star--cycle;
draw(star,linewidth(2)+lightblue);
label("\footnotesize $0$",point(star,0),NE);
```

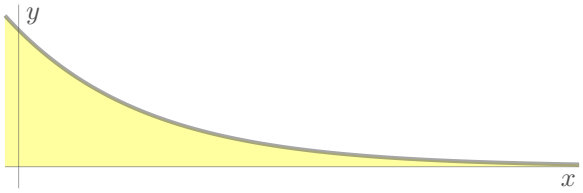
The label incorporates a LaTeX command to produce a subscript-sized label

that is northeast of the path's starting point (the initial point on the initial line segment is point 0 of the path `star`, the initial point on the following line segment is the path's point 1, etc.).

Asymptote embeds these four core capabilities inside of a powerful language with a familiar syntax.

We will illustrate with some examples of jobs that are typical for our work. For each we will first describe the figure and then examine the code listing in more detail.

The next graphic began life as an illustration for a calculus lecture.



For us, the natural way to produce this is not to use a mouse to try to get a good approximation of the true picture. Instead, we want to define the function $f(x) = e^{-x}$ and from that have the computer generate the graph and fill the area below it.

That thinking goes a long way toward writing the code, shown below. The key line is the fourth one where, after importing modules and declaring the horizontal size (*TUGboat* has columns that are about three inches wide), we define the function `f`. With that function, Asymptote makes the path that is the function's graph, fills the area between that graph and the x-axis, and finally draws the curve.

```
include "jh.asy";
import graph;
size(3inch);
real f(real x) {return exp(-x);}
real xmin=-0.1, xmax=4.1;
real ymin=-0.1, ymax=1/exp(-.1);
path g=graph(f,xmin,xmax,operator ..);
path c=buildcycle(g,
                  (xmax,f(xmax))--(xmax,0),
```

```
                (xmax,0)--(xmin,0),
                (xmin,0)--(xmin,ymax));
fill(c,THINPEN+FILLCOLOR+opacity(0.75));
xaxis("\footnotesize $x$",ymin,xmax,AXISPEN,
      above=true);
yaxis(Label("\footnotesize $y$",align=E),AXISPEN,
      above=true);
draw(g,FCNPEN);
```

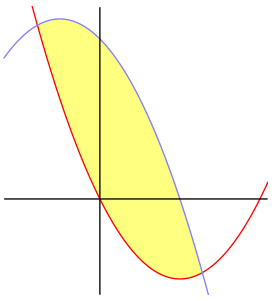
There are two things to note about this listing. The first is that to get the area to be filled we must close in the left and right sides. The `buildcycle` command creates the cyclic curve from the given sequence of bounding curves; the left and right sides are just line segments created with the `--` operator.

The second thing to note is that this code imports the standard Asymptote module `graph` to bring in the commands `graph`, `xaxis`, and `yaxis`. This is one of the add-on modules mentioned above to help with common tasks. This listing also has `include "jh.asy"` to bring in a local `.asy` file. It contains some of the authors' own commands and defines some constants to give a set of graphics a more uniform look. Here is `jh.asy`.

```
import fontsize; defaultpen(fontsize(9.24994pt));
import texcolors;
pen FILLCOLOR=lightyellow;

pen THINPEN=squarecap + linewidth(0.4pt);
pen AXISPEN=THINPEN + gray(0.3)
            +opacity(.5,"Normal");
pen FCNPEN=squarecap +linewidth(1.5pt) + gray(0.3)
            +opacity(.5,"Normal");
```

The prior example is written in a style where we declare what we want and Asymptote figures out how to draw it. The advantage of this over using a mouse-based painting program is like the advantage of separation of appearance from content that we get when using a system built on TEX rather than a word processor. The next example also illustrates this writing style.

To show the area between two curves

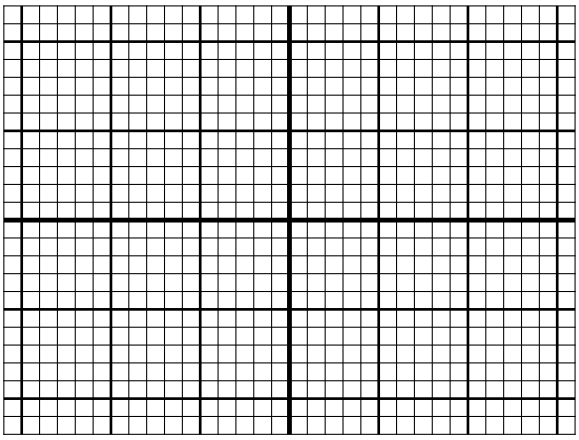


our inclination is to define the two functions and then ask Asymptote to use those definitions to generate the paths, find the intersections, and then construct and fill the desired area.

```
import graph;
size(0,1.5inch);

real p_up(real x) {return x^2-2*x+0;}
real p_down(real x) {return -x^2-x+2;}
real xmin=-1.2, xmax=2.1;
real ymin=-1.2, ymax=2.4;
path g_up=graph(p_up,xmin,xmax,operator ..);
path g_down=graph(p_down,xmin,xmax,operator ..);
pair ipoints []=intersectionpoints(g_up,g_down);
path c=
   graph(p_up,ipoints[0].x,ipoints[1].x,
         operator ..)
   --
   graph(p_down,ipoints[1].x,ipoints[0].x,
         operator ..)
   -- cycle;
fill(c,lightyellow);
draw(g_up,red);
draw(g_down,lightblue);
xaxis(above=true); yaxis(above=true);
path clippath = (xmin,ymin)--(xmax,ymin)
  --(xmax,ymax)--(xmin,ymax)
  --cycle;
clip(clippath);
```

In detail the code is much like the prior example. After importing the module, we set the graphic size (we set the height to be 1.5 inches; setting the width to 0 has the system find the natural width). We then define the functions `p_up` and `p_down`. We ask Asymptote to make paths that are the two graphs with given left and right endpoints, find where those two intersect, and construct the closed curve `c` between the two (`ipoints.x` gives the first component of `ipoints`). Asymptote fills the area inside, draws the two graph paths, and finally clips the ends of the parabolas that extend too far away from the part of the picture that we want to show.

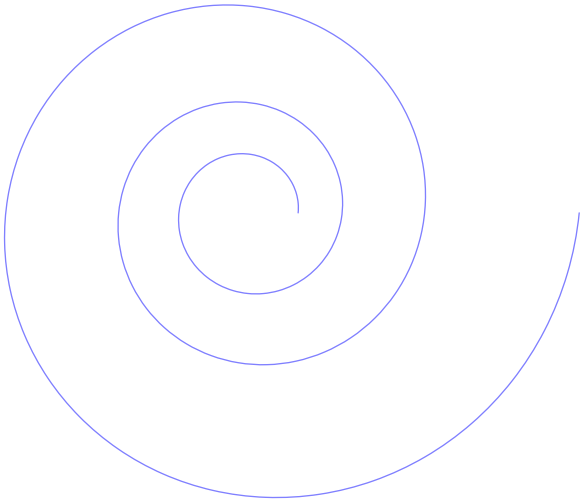
Asymptote includes a full suite of powerful programming constructs. The next illustration is rectilinear graph paper with three different line thicknesses.



The code is a simple illustration of looping. We won't expand on it since readers who have programmed in mainstream languages such as Java or Python will recognize it right away.

```
size(3inch);
pen thinpen=(linewidth(.4)+extendcap+miterjoin);
pen mediumpen=(linewidth(1)+squarecap);
pen thickpen=(linewidth(1.8)+squarecap);
int xmin=-16, xmax=16;
int ymin=-12, ymax=12;
// draw horizontals
for (int k=xmin; k<=xmax; ++k) {
    if (k==0) draw((k,ymin)--(k,ymax),thickpen);
    else if (k%5 ==0)
      draw((k,ymin)--(k,ymax),mediumpen);
    else draw((k,ymin)--(k,ymax),thinpen);
    }
// draw verticals
for (int k=ymin; k<=ymax; ++k) {
    if (k==0) draw((xmin,k)--(xmax,k),thickpen);
    else if (k%5 ==0)
      draw((xmin,k)--(xmax,k),mediumpen);
    else draw((xmin,k)--(xmax,k),thinpen);
    }
```

The final two-dimensional graphic has a polar flavor; it's a logarithmic spiral.



The source shows how easily we can define and combine functions, and again illustrates how the language fits with how a person with mathematical training thinks.

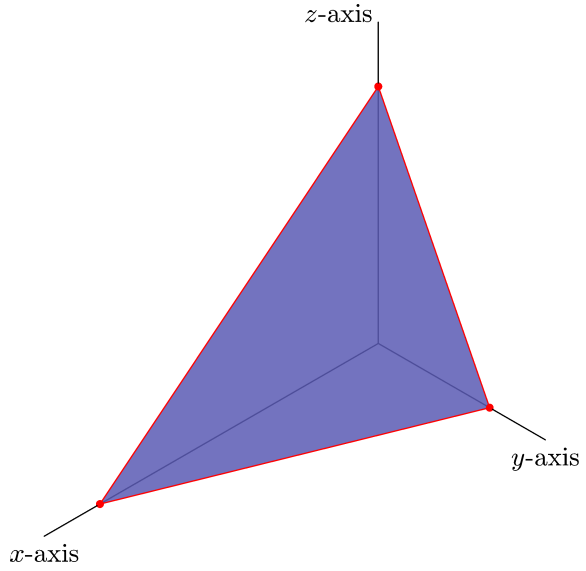
```
import graph;
size(3inch);
real pi=2*acos(0);
real a=0.5, b=0.1; // parameters for the shape
// polar to cartesian
real x(real t) { return a*exp(b*t)*cos(t);}
real y(real t) { return a*exp(b*t)*sin(t);}

pair f(real t) { return (x(t),y(t));}
draw(graph(x, y, 0, 6*pi, operator ..),lightblue);
```

In particular, note that the function `f` returns a pair of reals, where the coordinate functions were defined earlier in the listing.

3 In the limit: Three dimensions

In addition to its capabilities in two dimensions, Asymptote comes with modules targeted at three-dimensional graphics. These develop some of METAPOST's ideas from two dimensions. For instance, a core capability of METAPOST (and originally in METAFONT) is to produce useful and good-looking curves without requiring that the author completely specify those curves. Asymptote extends this to 3D.

Our first graphic is a straightforward image, another one that might appear in a calculus lecture.



Because it is a three-dimensional graph, we will import a different module, but the thinking behind the code is similar to what we've done earlier. In addition to drawing the axes, we declare where the plane intersects each axis, make a surface connecting those three points, and have Asymptote draw and fill the surface.

```
size(3inch);
import settings;
settings.render=10;
settings.maxtile=(50,50);
import graph3;
currentprojection=orthographic(2,2,2);
currentlight=(9,3,4);
// where plane intercepts x, y, and z axes
triple intercepts=(5,2,4);
path3 P = (intercepts.x,0,0)
         --(0,intercepts.y,0)
         --(0,0,intercepts.z)
         --cycle;
draw(surface(P), lightblue+opacity(.85));
draw(P,red);
```

```
dot(P,red);
axes3("\footnotesize $x$-axis",
      "\footnotesize $y$-axis",
      "\footnotesize $z$-axis",
      (0,0,0),(6,3,5));
```

The listing imports the 3D module `graph3`. This gives us a command to draw axes in three dimensions and also lets us define the projection to be orthographic (projection lines are orthogonal to the plane to which the image is projected, that is, the projection is from infinity) and to define the location of the light source.

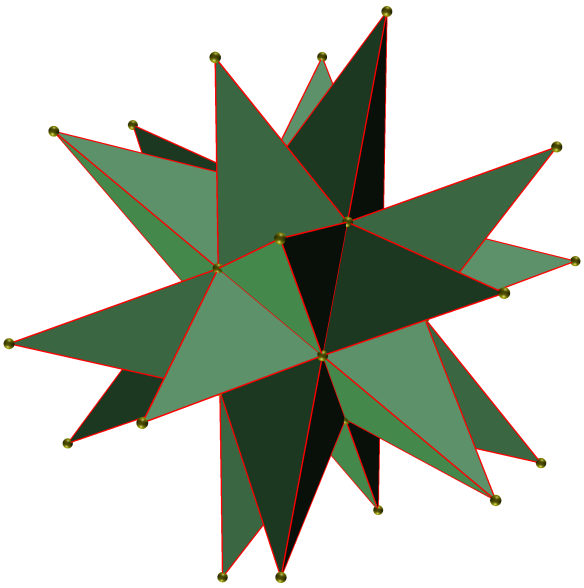
One more point about this graphic: we produced it as a `.png` file using this command line.

```
$ asy -fpng intercepts_plane
```

We chose this format to show well on the printed page; we used `.pdf` for the earlier figures but it doesn't show in our viewer because Asymptote's default behavior is to produce a figure that can be manipulated with the mouse — turned, or zoomed to — but that behavior relies on a viewer's capabilities, and some viewers lack that capability (*TUGboat*'s printed pages also don't have it!). Because of this change to the `.png` raster format we also changed Asymptote's defaults to adjust `render` for 10 pixels per big point to reduce jaggies and changed `maxtile` to get around a common bug in the graphics driver.

Our final two figures are less prosaic. We want to close with the message that Asymptote can indeed do some fancy things, often with very little code.

First, here is a stellated icosahedron.

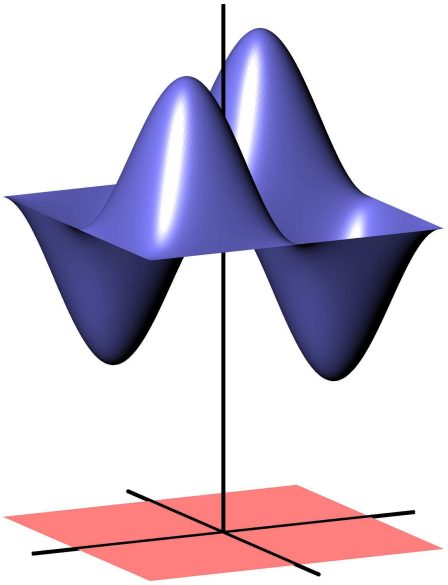


The construction is logical and easy to implement using Asymptote. First define the twelve points of the icosahedron, then use these points to define the twenty faces and, finally, create a function that will erect a pyramid on (i.e., stellate) each of these faces. Asymptote itself takes care of the projections and shading.

```
size(3inch);
import settings;
settings.render=10;
import three;
currentprojection=perspective(21,25,15);
currentlight=White;

real phi = (1+sqrt(5))/2;
// Vertices of the icosahedron are of the form
// (0, \pm 1, \pm\phi), (\pm\phi, 0, \pm 1),
// (\pm 1, \pm\phi, 0)
triple [] Pts = {
(  0,    1,  phi),
(  0,   -1,  phi),
( phi,   0,    1),
(  1,  phi,    0),
( -1,  phi,    0),
(-phi,   0,    1),
( phi,   0,   -1),
(  0,    1, -phi),
(-phi,   0,   -1),
( -1, -phi,    0),
(  1, -phi,    0),
(  0,   -1, -phi)
};

// Faces listed as triples (i,j,k) corresponding
// to the face through Pts[i], Pts[j] and Pts[k].
triple [] faces = {
// upper cap
(0,1,2), (0,2,3), (0,3,4), (0,4,5), (0,5,1),
// upper band
(11,6,7), (11,7,8), (11,8,9), (11,9,10),
(11,10,6),
// lower band
(10,1,2), (6,2,3), (7,3,4), (8,4,5), (9,5,1),
// lower cap
(3,6,7), (4,7,8), (5,8,9), (1,9,10), (2,10,6)
};
// draw the twelve vertices of the icosahedron
for (triple T: Pts)
  draw(shift(T)*scale3(.08)*unitsphere,
       yellow);
real t=3.0; // Scaling for stellation height
// Function to compute the stellation point
triple stell_point(triple u, triple v, triple w)
          {return t/3*(u+v+w);}

void stellate(triple Face) {
   int i=round(Face.x),
     j=round(Face.y),
     k=round(Face.z);
   triple S=stell_point(Pts[i], Pts[j], Pts[k]);
   draw(shift(S)*scale3(.08)*unitsphere,
        yellow);
   draw(S--Pts[i],red);
   draw(S--Pts[j],red);
   draw(S--Pts[k],red);
```

```
    draw(Pts[i]--Pts[j]--Pts[k]--cycle,red);
    draw(surface(S--Pts[i]--Pts[j]--cycle),
         lightgreen);
    draw(surface(S--Pts[i]--Pts[k]--cycle),
         lightgreen);
    draw(surface(S--Pts[j]--Pts[k]--cycle),
         lightgreen);
    }
for (triple Face: faces ) stellate (Face);
```

Finally, while the prior image is polyhedral, that is, the sides are flat, our closing example is a real surface in that it is curved.



As with the earlier listing, the code here merely defines a function `f` and a region over which the graph of that function will lie, and then asks Asymptote to produce the graph. (The `nx` value gives the mesh).

```
size(3inch);
import settings; settings.render=10;
import graph3;
currentprojection=orthographic(2,4,1);
currentlight=(5,4,4);
real pi=2*acos(0);
path3 P=(-1,-1,0)--(-1,1,0)--(1,1,0)--(1,-1,0)
       --cycle;
draw(surface(P),lightred,nolight);

real f(pair z)
   {return 2+sin(z.x*pi)*sin(z.y*pi);}
draw(surface(f,(-1,-1),(1,1),nx=128), lightblue);

pen axispen=(linewidth(1.5)+squarecap);
axes3((-1.3,-1.3,0),(1.3,1.3,3.3),axispen);
```

4 Convergence of technologies: Comparison with Ti*k*Z and PSTricks

The TEX community now has the luxury of choice among three very capable graphics systems, Asymptote, Ti*k*Z,[4] and PSTricks.[5] (We have left METAPOST off this list because at the moment three-dimensional graphics are an issue.) These three are similar. They are close in ability and very powerful, and all are under active development. All have extensive add-on sets that greatly increase their usefulness.

Any of the three can be a great choice for your projects. To some extent, which you choose will be dictated by which one has modules that fit your exact needs. It is also partly a matter of taste.

One point in favor of using Asymptote is that it is a stand-alone program. This may reduce the extent to which your document depends on the current software ecosystem because under natural development you make a stand-alone graphic. Put another way, Asymptote fits a bit better with the Unix philosophy of having a number of tools, each of which does one thing only, but does it well.

For us, a particularly appealing feature is that Asymptote lifts many METAPOST constructs from 2D to 3D. It also has additional advanced functions.

Finally, programming in Asymptote is in a style close to Java and C++, which you may find familiar. For us, graphics is something that we do only occasionally and so we must switch to this language from others. Having familiar constructs helps that switching.

References

[1] J. C. Bowman. Asymptote: Interactive TEX-aware 3D vector graphics. *TUGboat*, 31(2):203–205, 2010. `http://tug.org/TUGboat/tb31-2/tb98bowman.pdf`.

[2] John Bowman. Interactive TEX-aware 3D vector graphics. *TEX Users Group Annual Meeting*, 2010. `http://river-valley.tv/interactive-tex-aware-3d-vector-graphics`.

[3] John C. Bowman and Andy Hammerlindl. Asymptote: A vector graphics language. *TUGboat*, 29(2):288–294, 2008. `http://tug.org/TUGboat/tb29-2/tb92bowman.pdf`.

[4] John C. Bowman and Orest Shardt. Asymptote: Lifting TEX to three dimensions. *TUGboat*, 30(1):58–63, 2009. `http://tug.org/TUGboat/tb30-1/tb94bowman.pdf`.

◇ Michael Doob
University of Manitoba

◇ Jim Hefferon
Saint Michael's College
`jhefferon (at) smcvt dot edu`

[4] `http://sourceforge.net/projects/pgf`
[5] `http://tug.org/PSTricks`

The joy of \csname...\endcsname

Amy Hendrickson

Abstract

A surprisingly useful tool, \csname–\endcsname, offers many opportunities for interesting and useful macros, especially when it is convenient to dynamically generate a series of definitions.

Trivially a series of \csname definitions may be used to produce endnotes, but there are more interesting and complex constructions as well.

Another example shows how \csname may be used for on-line report generation. In this instance, we dynamically generate hyperlinked tabs for custom risk analyses of particular stocks chosen on-line by the client. We can use these named tabs to build a hyperlinked Table of Contents on the fly.

A similar process may be used to produce hyperlinked, tabbed, documentation.

The final example shows how definitions made with \csname can be used to send a set of definitions to an auxiliary file, where each new definition contains the current page number in its name, and a number as its definition.

This allows the dynamic redefinition of the command for a particular page, *within the auxiliary file*, depending on whether the value of the new definition is higher than the value of the previous definition for the same page.

Code will be shown for each of these methods to dynamically generate macros using \csname.

1 The basics

\csname and \endcsname are TEX primitives that allow us to define and call macros. If we compare a definition made with and without \csname we can see that initially a definition made with \csname is not much different than one made with \def. For instance, if we compare these two definitions,

1. **A definition with** \def:
   \def\puppy{Toy Poodle}
   and the new macro is called by writing: \puppy,
2. **A definition with** \csname...\endcsname:
   \expandafter\def\csname puppy\endcsname
   {Toy Poodle}
   called with: \csname puppy\endcsname

we find the results in either case to be 'Toy Poodle'. So, why bother with \csname...\endcsname?

2 Useful characteristics

As we will see, \csname has several of characteristics making it uniquely useful:

1. We can use \csname to find out if a command has been defined, since an undefined command is equal to \relax.

   As an example, we can make a conditional that tests to see if a command has been defined and make choices based on the result:

```
\expandafter\ifx
\csname anycommand \endcsname\relax
<do this>\else<do that>\fi
```

2. \csname allows us to define and call commands that may be composed of numbers, symbols, and other commands, unlike the basic command for making definitions, \def, which must use only letters for the name of a new definition.

   For example, this is a valid definition that uses symbols and a number in its name:

```
\expandafter\def\csname $&3\endcsname{Hi!}
```

   Commands made with non-letters must be called using \csname–\endcsname. In this case, \csname $&3\endcsname must be used, and produces: 'Hi!'.

   Another example comes from latex.ltx, the basic LaTeX macro set, a definition that uses \csname for making macros that might have characters other than letters in their name:

```
\def\@namedef#1{\expandafter\def\csname
                #1\endcsname}
```

   \@namedef{} is used widely in LaTeX code. As one example, \@namedef{} is used in the macro for making labels for cross-referencing. This is why you can make a label that looks like this, \label{fig1}, where the argument includes a number.

3. A macro argument may be used within \csname. As an example, again from latex.ltx:

```
\def\setcounter#1#2{\@ifundefined{c@#1}%
  {\@nocounterr{#1}}%
  {\global\csname c@#1\endcsname#2\relax}}
```

   used, e.g.: \setcounter{page}{201}

   The macro checks to see if there is a counter called \c@#1. If there is no counter with that name it will give an error message; if there is a defined counter, it uses \csname to call the counter and sets it to the number given as the second argument. In this example, it sees that a counter named \c@page exists, so sets it equal to the second argument, '201' in this example.

   Generic macro. The example above shows \csname...\endcsname being used to make a kind of generic macro since it will have the flexibility to be used with any previously defined LaTeX counter.

4. **Expand commands within \csname.** Here's where things get interesting. We can expand commands within a definition name made with \csname...\endcsname. This opens up many complex possibilities. For one set of possibilities, we can include a counter in the name of a new definition.

In this article we'll explore a number of ways in which we can use \csname...\endcsname with counters.

2.1 Dynamic macro building

We can use a counter within \csname...\endcsname to make a series of macros, a new macro every time the counter is advanced.

We do this by including a definition, made using \csname...\endcsname with a counter in its name, within the body of another definition. The outer definition advances a counter every time it is used, producing a new and unique inner macro every time it is called.

For example, we can make a command that will make more commands in this way:

```
\newcount\applenum
\def\applename#1{\global\advance\applenum by 1
  \expandafter\def\csname apple\the\applenum
  \endcsname{**#1**}}
```

Every time we use the \applename{} macro, we define a new macro, named \apple1, \apple2 and so on.

Using a loop to call the macros

To access the newly made inner macro we can use a loop, which advances a counter in each iteration, and calls the inner macro using the current state of the counter as part of the macro name.

To call the macros made with the \applename macro above, we test to see if \applename⟨*number*⟩ is defined. If defined, we call the command using the current state of the \loopnum counter in the body of the name of the command; else, end the loop.

```
\newcount\loopnum
\loopnum=1
\loop\expandafter\ifx
\csname apple\the\loopnum\endcsname\relax
\else
  \csname apple\the\loopnum\endcsname\
  \global\advance\loopnum by 1
\repeat
```

Used:

```
\applename{Macintosh}\applename{Gala}
```

Results:

Macintosh **Gala**

3 Endnotes example

For our first real world example we will use this tool to make endnotes. In this example we want to change the definition of \footnote so that it produces endnotes rather than footnotes. We do this by making an endnote definition that makes a new definition every time it is called.

We start with a new counter to be used by our endnotes, \endnum. In the \endnote macro we advance the \endnum counter, then raise and print the number in the text for our endnote number.

Next we make a construction with \csname that builds a new definition, using the current state of the \endnum counter. This new definition will be used to save the text of the endnote.

```
\newcount\endnum
\def\endnote#1{\global\advance\endnum by 1
  $^{\the\endnum}$%
%%
%% Here we make the new definition using
%% \the\endnum in the definition name so that
%% each new definition is unique:
%%
\long\expandafter
\def\csname endnote\the\endnum\endcsname{%
  \small\leftskip=12pt\relax\parindent=-12pt
  \indent\hbox to 12pt{\the\loopnum.\hfill}%
  %%
  %% Here we save the text of the endnote:
  #1%
  \strut\vskip2pt}}
```

Now we set \footnote to be equal to \endnote, so every time \footnote is used, the command actually called is \endnote:

```
\let\footnote\endnote
```

To print the endnotes, we make a loop that advances a counter with every iteration. That counter is used within the name of the definition made with \csname...\endcsname. The loop continues until it comes to an undefined endnote, thus cycling through every defined endnote. An example is shown in figure 1.

```
\newcount\loopnum
\def\printendnotes{\global\loopnum=1
%%
%% Test to see if any end notes have been
%% defined; If so, provide the title and
%% start loop; if not, do nothing.
%%
\expandafter\ifx
\csname endnote\the\loopnum\endcsname\relax
            \else
\subsection*{Endnotes}\everypar{}
\vskip6pt
\small\leftskip=12pt
```

```
``A day of dappled seaborne clouds.%
\footnote{Quotation from James Joyce's
`Portrait of the Artist as a Young Man'.}
The phrase and the day and the scene
harmonised in a chord. Words. Was it
their colours? He allowed them to glow
and fade, hue after hue: sunrise gold, the
russet and green of apple orchards, azure
of waves, the greyfringed fleece of
clouds.\footnote{The Bloomsday
celebration in Dublin this year features a
concert of compositions honoring Joyce.}

\printendnotes
```

"A day of dappled seaborne clouds.[1] The phrase and the day and the scene harmonised in a chord. Words. Was it their colours? He allowed them to glow and fade, hue after hue: sunrise gold, the russet and green of apple orchards, azure of waves, the greyfringed fleece of clouds.[2]

Endnotes

1. Quotation from James Joyce's 'Portrait of the Artist as a Young Man'.
2. The Bloomsday celebration in Dublin this year features a concert of compositions honoring Joyce.

Figure 1: Testing the endnote commands

```
%% Loop continues until it finds an
%% undefined endnote
%%
\loop\expandafter\ifx
\csname endnote\the\loopnum\endcsname\relax
                  \else
%% Print endnote
\csname endnote\the\loopnum\endcsname
\vskip2pt
%%
%% Reset: redefine current endnote to \relax
%% preventing this definition from being
%% used the next time \printendnotes is called.
%%
\global\expandafter
\let\csname endnote\the\loopnum\endcsname\relax
%%
\global\advance\loopnum by 1
\repeat
\fi
%% \fi ends test at beginning of this macro
%% to see if any endnotes have been defined.
}
```

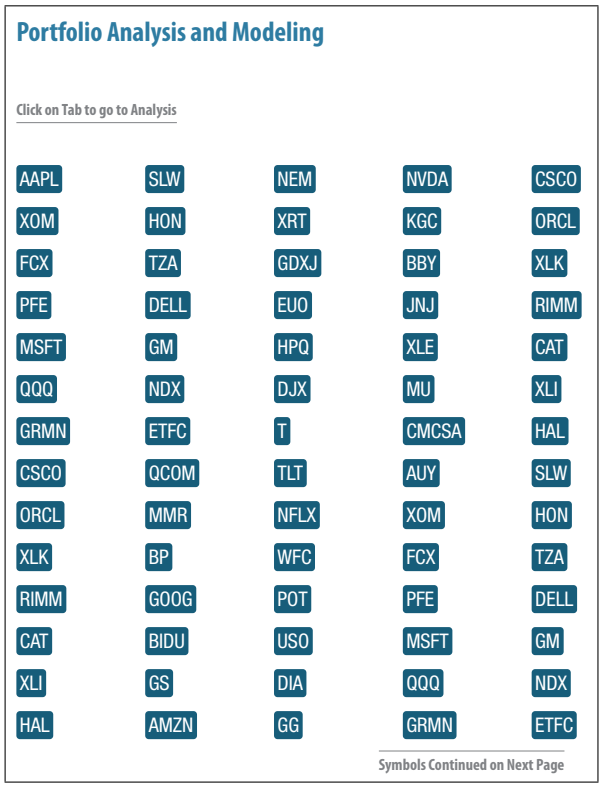


Figure 2: One form of automated online report generation; this is a draft version of customized financial reporting. Each symbol is automatically generated and is hyperlinked to the appropriate page of the report. The company analyzed depends on input from the client; the symbols and their linking is done through macros utilizing `\csname`.

4 Example: On-line report generation

A somewhat similar construction may be used to make hyperlinked tabs for on-line report generation (figure 2). (The actual reports use color, too.)

This set of macros is used to automate the naming of hypertargets so that we can hyperlink to them on the first page of the report, using a `\csname` construction and a loop, and using Ti*k*Z for making the hyperlinked tab.

The name and number of companies analyzed is determined by the client who submits a request online. Each company's analysis will start on a titled new page. Part of the definition for the title of a report will be the command `\maketab{#1}`.

`\maketab` takes a stock symbol as its argument, and generates a hypertarget so that we can link to it from the beginning of the report, in the equivalent of the table of contents page, using the same `\codenum` counter. Then it makes a new definition with `\csname` and the `\codenum` counter in its name,

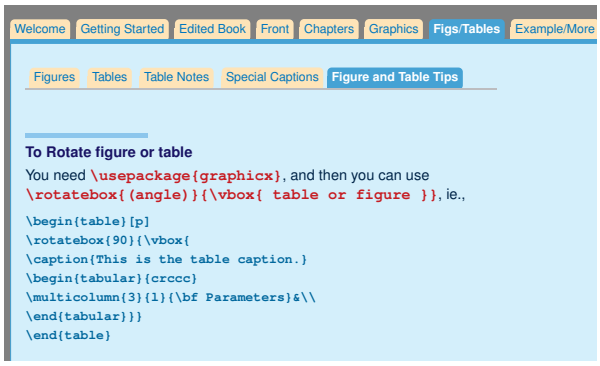


Figure 3: A similar technique is used for making hyperlinked tabbed documentation, where the hypertargets are made with `\csname` and a counter, accessed with hyperlinks named with `\csname` and a counter. The tabs in this example are also made with Ti*k*Z.

with the stock symbol as its definition, and sends it to the `.aux` file.

```
\def\maketab#1{\global\advance\codenum by 1
  \hypertarget{link\the\codenum}{}%
  \immediate\write\@auxout{\string\expandafter
    \string\gdef\string\csname\space
    tab\the\codenum\string\endcsname{#1}}}
```

Once we have this in place we can use our loop construction for the first, and possibly continuing, pages to build the hyperlinked tabs. `\gettabs` uses a loop to call the individual tabs, as long as there is one defined. This can continue over a number of pages if necessary.

```
\begin{multicols}{5}
\loopnum=1 \gettabs
\end{multicols}
```

As you can see, `\gettabs` is where the work is done. Here is how it is defined:

```
\def\gettabs{\loop
\expandafter\ifx
\csname tab\the\loopnum\endcsname\relax
            \else
\vskip6pt\hbox to 1in{%
%%
%% \hyperlink takes two arguments;
%% the first the name of the hypertarget,
%% and the second, the text that will link
%% to the hypertarget when clicked:
\hyperlink{link\the\loopnum}%
{\plaintab{\csname tab\the\loopnum\endcsname}%
 \hskip12pt}%
\hfill}%% <== end \hbox started above
\global\advance\loopnum by 1
            \repeat}
```

If you are interested in how to make the tab with Ti*k*Z, here is that code:

```
\definecolor{dkblue}{cmyk}{.9,.53, .32, .2}
\def\plaintab#1{%
\hbox{\normalsize\sf
\begin{tikzpicture}
[rounded corners=3pt, inner sep=3pt]%
   \node[rectangle,fill=dkblue]
{\Large\sf\color{white}
\vrule depth 3pt width 0pt height 15pt \relax
#1};
\end{tikzpicture}}}
```

A similar technique can be used to produce tabbed documentation, as shown in figure 3. For the full example, please see `http://www.texnology.com/docs.pdf`.

5 Another `\csname` technique, for classification levels

The problem:

When producing a classified document, the highest level of classification (secret, top secret, etc.) on any particular page must appear at the top of that page. When the classification level is given, the user doesn't know the page on which it will appear. In addition, the user doesn't know in advance whether this particular classification level is the highest on that page.

The solution:

We can use a `\write` to be sure that we know the page number where the markup has appeared, so the macro for making classification level markup will send the level along with the page number to an auxiliary file, using a `\write` command.

Now we have a page number and a level appearing on that page. However, we still don't know which level is highest for the particular page.

The second part of the solution involves defining the highest level for a particular page, *in the auxiliary file*, by also sending code for comparing levels on a particular page, and making a definition for the particular page *only if the present level is the highest for that page number*. When the auxiliary file is input, the next time LaTeX is run on the root file, the definition of the highest level on each page has been defined.

There are many more complications in the full problem. For instance, how do we pass information on the level of a paragraph that has broken over pages, so that the part of the paragraph on the second page will contribute to the calculation of the highest level on the second page? For the sake of brevity, we'll consider only the general mechanism here.

5.1 Setting up

We use a `\write` for every instance where a classification level is written in the text with the command `\secmark`. `\write` is only activated after the page is made up, so we are sure that we will be using the correct page number when we send the information to the auxiliary file. Since we will have many `\write` commands in the `.tex` document, we will write to a new auxiliary file, `\jobname.lev`, instead of using standard LaTeX auxiliary file, `\jobname.aux`. We name the new write:

```
\newwrite\collect
```

5.2 The counter to be used

The next item we need is a counter to use when defining our `\csname` commands. Since we want a command that has the current page number in its name, we would be tempted to use the LaTeX page counter, `\c@page`.

However, in the common case where the beginning of the document uses roman numerals, and the body of the document uses arabic numerals, we would have the unfortunate result of having multiple pages with the same page number.

So instead, we make a new counter, and call it `\superpage`:

```
\newcount\superpage
```

5.3 Using `\shipout`

`\shipout` is the TeX primitive that is called every time a page is completed. We use `\shipout` to generalize this solution, so that this system will work independently of any page style, and its headers and footers.

We can use `\shipout` to advance the counter called `\superpage`. This gives us a new number for every page, continuous through the document. Now `\shipout` can be used to print the classification term on the top and bottom of the page using `\superpage` as the counter found in the name that has been defined with `\csname`...`\endcsname`.

5.4 Doing the `\write`s

The `\secmark` macro works by sending a definition for the classification level on a particular page to `\jobname.lev` file, using a `\write` associating the page number with the level given. The `\write` will not be activated until the page is made up, so we are guaranteed to have the correct page number sent to the `\jobname.lev` file. This works as well for figure or table floats, since `\write` will send out the information to the `.lev` file only after the page is made up, and the page where the floats will appear has been determined.

The `\write` sends information to the auxiliary file, `\jobname.lev`, including several conditional tests. The command looks messy and verbose because when the write is made, we have to stop the expansion of many commands by preceding each one with `\string`, except for those few commands that we want to expand immediately; in this case, the super page number:

```
\write\collect{%% ^^J makes a blank line
%% in the \jobname.lev file so that
%% it is easier to see where each test ends:
^^J^^J
%%
\string\expandafter\string\ifx\string\csname%
\space LevelOnSuperPage\the\superpage
\string\endcsname\string\relax
\string\expandafter\string\gdef\string\csname
\space LevelOnSuperPage\the\superpage
\string\endcsname{#1}
\string\else
\string\ifnum \string\csname\space
LevelOnSuperPage\the\superpage\string\endcsname
\string< #1
\string\expandafter\string\gdef\string\csname
\space LevelOnSuperPage\the\superpage
\string\endcsname{#1}\string\fi\string\fi
^^J}%
```

... which might make more sense when we see how the code looks by the time it is expanded and appears in the `\jobname.lev` file. Here, the level sent for page 5 is '2'.

```
\expandafter\ifx
\csname LevelOnSuperPage5\endcsname\relax
\expandafter\gdef
\csname LevelOnSuperPage5\endcsname{2}
\else\ifnum\csname LevelOnSuperPage5\endcsname<
2 \expandafter\gdef
\csname LevelOnSuperPage5\endcsname{2}
\fi\fi
```

This process can be repeated as many times as needed for each page, with only the highest number, determined by each test, being used to define `\csname LevelOnSuperPage?\endcsname`.

Then, when `\jobname.lev` is brought into the base `.tex` file the next time LaTeX is run on the document, it will include a series of unique macros, one for each page in the document where a classification mark has been used, defining the highest number given for that page. Since the definition is made with `\csname`...`\endcsname` we can have the superpage number contained in the name of the definition. This allows us to call the definition using the current superpage number in the `\csname`...`\endcsname`, in the shipout.

5.5 Using the level information

We can use these definitions with every shipout, with the macro `\makeclassification` being called at the top and bottom of the page. Here is its definition:

```
\def\makeclassification{%
\vbox{\baselineskip=12pt
%% Is there a definition for this page?
\expandafter\ifx
\csname LevelOnSuperPage\the\superpage
\endcsname\relax
%% if not:
\centerline{}
\else
%% if there is a definition:
\centerline{%
\ChangeNumIntoClassification{%
\expandafter\csname
LevelOnSuperPage\the\superpage
\endcsname}}\vskip3pt\fi}}
```

`\ChangeNumIntoClassification`, seen above, uses the definition of

```
\csname LevelOnSuperPage\the\superpage
\endcsname
```

as its argument, which will yield a number from 1 to 4. This allows us to use `\ifcase` to trivially change that number into the classification term:

```
\def\ChangeNumIntoClassification#1{%
\ifcase#1\or Unclassified \or Classified
\or Secret \or Top Secret
\else ! Please Run LaTeX Again to Get the
Classification Level !
\fi}
```

And now we will have the highest classification level reliably appearing on top of each page.

Summary: Ways in which `\csname` is exceptionally useful

1. Testing to see if a macro has been defined.
2. Making a macro that has characters other than letters in its name, e.g., a cross referencing label.
3. Making a generic macro that can be modified with the argument of another macro.
4. Generating new macros by using a counter in the name made with `\csname`...`\endcsname`.
5. Calling macros made with `\csname` with a loop. The loop may be stopped by testing to see if the most recent `\csname` ⟨*counter*⟩`\endcsname` combination has been defined.

   Using this method to stop looping has the advantage that we don't need to know in advance how many definitions were made, and we will cycle through all available definitions before ending the loop.
6. A `\csname`...`\endcsname` definition including a counter in its name can be used to generate a series of hypertext targets automatically.
7. Definitions can be made using the page number as part of the name, which can be called by the output routine.
8. Finally, we have the technique of sending information to an auxiliary file with a `\write` and making new `\csname` ⟨*counter*⟩`\endcsname` definitions in the body of the auxiliary file, depending on the results of a conditional test. When the auxiliary file is input into the root `.tex` file, we can then use the resulting definition in a variety of ways.

`\csname` in the future

More than a coding oddity, `\csname`...`\endcsname` is a workhorse, allowing many constructions that wouldn't otherwise be available.

Likely there are many more opportunities to use these techniques, particularly with off-label uses for LaTeX such as report generation, or building e-documents on the fly, and other web-oriented macro writing projects.

Enjoy!

(The slides for the TUG 2012 conference talk are available at `http://www.texnology.com/talk.pdf`.)

◇ Amy Hendrickson
57 Longwood Avenue
Brookline, MA 02446
USA
amyh (at) texnology dot com
http://www.texnology.com

TUG 2012 abstracts

Bill Cheswick

An iTEX update

An update on the iTEX project for ebook publishing with TEX, described in *TUGboat* 32:2.

Federico Garcia

Documentation in TEXnicolor

My package `colordoc` builds on Frank Mittelbach's `docstrip` system of documentation, adding some utilities to use color in the code: matching delimiters (`{` and `}`) are colored the same, just as matching `\if`–`\fi` pairs. Commands are made red, bold, and italics, when they are being `\def`ined, just as variables when they are being declared (`\newcount`, `\newif`, etc.). These tools have saved me a lot of time and trouble when editing or trying to understand a code. In the presentation I also describe the interesting general lines of the workings of both `doc` and `colordoc`.

Troy Henderson

User-friendly web utilities for generating LATEX output and MetaPost graphics

The full article was printed in *TUGboat* 33:1. The online previewers are available at:

`http://www.tlhiv.org/ltxpreview`	LaTeX
`http://www.tlhiv.org/mppreview`	MetaPost
`http://www.tlhiv.org/mpgraph`	Function Grapher

Sherif Mansour & Hossam Fahmy

Experience with Arabic and LuaTEX

This is an experience report of an attempt to include the AlQalam font for Arabic script within LuaTEX. We describe the problems we faced trying to figure out how to use a new right-to-left font within LuaTEX. We also describe how to call the many different shapes that are defined via parameters in the original font. We also present some ideas to modify the line breaking algorithm of TEX to allow the use of different shapes for the same character in order to justify the line. This is still work in progress.

Frank Mittelbach

E-TEX: Guidelines for future TEX extensions, revisited

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

> Now it is time, after ten years' experience, to step back and consider whether or not TEX 3.0 is an adequate answer to the typesetting requirements of the nineties.
>
> Output produced by TEX has higher standards than output generated automatically by most other typesetting systems. Therefore, in this paper we will focus on the quality standards set by typographers for hand-typeset documents and ask to what extent they are achieved by TEX. Limitations of TEX's algorithms are analyzed; and missing features as well as new concepts are outlined.

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

[Slides available at `www.latex-project.org/papers`.]

Steve Peter

Metafont as a design tool

Well-written Metafont sources provide a font designer with a nearly unparalleled tool to explore variations on a typographic theme. Paired with TEX in an advanced environment, the designer can explore serif structure, bracketing, weight variations and more in the context in which the font will be used: real textual matter. I'm going to ignore the production problems inherent to Metafont (not to mention the various possible solutions) to concentrate on the design aspects of this amazing tool.

Norbert Preining

Typesetting with Kanji — Japanese typography

Japanese typography is very particular and demanding in several respects: four writing systems mixed together (Kanji, Hiragana, Katakana, Roman letters); vertical and horizontal typesetting; traditional grid layout versus a mixture of writing systems. This all led to a spin-off TEX implementation called "Publishing TEX" (pTEX) that can deal with these specifics.

Until 2011 there was an independent distribution of TEX for Japanese users, first based on teTEX, later on TEX Live (`ptetex`, `ptexlive`). TEX Live 2011 and 2012 introduced all of the necessary tools and features and we hope that with TEX Live 2012 the need for a special setup for Japanese users is past.

In this talk we give an overview of the specialities of Japanese typography, presenting the difficulties met in modern texts. Continuing, we present the solutions provided by TEX Live to some of these problems, and discuss further development.

Norbert Preining

TEX Live 2012: Recent developments

TEX Live will be released in early summer 2012 and brings a couple changes that have been in the works for a long time: a "multi-`updmap`" that reads several `updmap.cfg` files, and multi-repository support for the TEX Live Manager `tlmgr`.

The `updmap` program generates the necessary

configuration files for `dvips`, `dvipdfm(x)`, `pdftex`, and `pxdvi` to display PostScript Type 1 fonts. It reads a configuration file that lists several map files, and combines all the font definitions from these map files. Until now local font maps had to be integrated into this `updmap.cfg` file, and so could easily be overwritten or otherwise be lost.

The new implementation has a long history. The original Perl version was written by Fabrice Popineau for Windows, later extended by Reinhard Kotucha and Karl Berry and used, starting last year, on all platforms supported by TEX Live. The code has now been extended to deal with multiple configuration files in a transparent way.

This allows a clear separation of `updmap.cfg` file parts. One `updmap.cfg` file now can (but does not have to) provide information about only the `texmf` tree it resides in. In other words, fonts installed into, for example, the `TEXMFLOCAL` tree can be activated by an entry in the `updmap.cfg` file *in this tree*.

We will discuss this new functionality and provide usage examples and advise on transition from the old system.

The other big change in TEX Live this year is the extension of the TEX Live Manager with the capacity of reading multiple repositories. In recent years, a few alternative TEX Live repositories have come into existence with a wide range of usage patterns: distribution of local packages (Japanese TEX related packages in `tlptexlive`, Korean TEX User Group repository), TEX Live infrastructure testing (in `tlcritical`), provision of development and non-free packages (in `tlcontrib`), etc.

Until now a user had to go through all desired repositories one by one passing the necessary parameters for each in turn. The new `tlmgr` supports use of several sources at the same time. The selection of packages appearing in multiple repositories is done by "pinning" packages to a repository.

We will present this new functionality, give usage examples, and a guided tour through setting up and using this new feature.

We will close with an overview of other changes in TEX Live 2012.

Will Robertson & Frank Mittelbach
LATEX3: From local to global — a brief history and recent developments
The original source code for LATEX3 dates to the early 1990s. Key aspects of its development occurred during that decade, but it was not until the late 2000s that the project began delivering code that was widely used by mainstream LATEX users. What happened in this time? This talk will discuss how LATEX3 development evolved over the decades and how it reached a state of being used to produce real users' documents whether or not they are actually aware of it. LATEX3 can be thought to consist of separate 'layers', and the programming layer known as `expl3` is starting to be used to solve problems in and write packages for LATEX 2_ε. Our plans are not restricted to such 'under-the-hood' measures, however, and we have discussed layers of LATEX3 that will have more visibility at the user interface. Our talk will discuss these separate layers and where our plans lead in the future, and will conclude with a demonstration of what's new in the current code. [Slides available at `www.latex-project.org/papers`.]

Will Robertson
Lineage and progeny of fontspec and unicode-math
My first LATEX package, `fontspec`, was written in 2004 before I knew how to program in LATEX and in truth before I knew how to program at all. This trial-by-fire introduced me to the lovely world of TEX programming and after some time I ended up writing a smattering of other works. (All the while actually starting to learn what this whole 'programming' thing was all about, including how to please and displease people who were just trying to get work done, thank you very much.) Some time later I foolishly tried 'planning' an ambitious new package, `unicode-math`, that took significantly longer to release. In the course of writing that package I learned really just how little I actually knew, and as a side-effect somehow ended up helping to write code for the LATEX3 project. In this talk I will talk about the motivation for writing these two packages, discuss recent developments with them, and finally touch on how LATEX3 influenced their development.

Herbert Schulz
Workshop: Introduction to TeXShop
A workshop introducing some of the more obscure and less used features of TeXShop for users who wish to become more proficient in its use to produce LATEX documents.

Christina Thiele
Almost 30 years of using TEX
It's not just TEX that's gotten older and more seasoned ... Reflections on changes in TEX and friends as used in a small typesetting company: software and hardware, of course, but also procedures and skills, resources that went from zero to virtually infinite, all of it interwoven with life and personal change. It's not earth-shaking news, but we've come far enough that looking back yields some interesting comparisons.

Die TEXnische Komödie 2–3/2012

Die TEXnische Komödie is the journal of DANTE e.V., the German-language TEX user group (`http://www.dante.de`). [Editorial items are omitted.]

Die TEXnische Komödie 2/2012

Marco Daniel, Das Paket `xparse` Dokumentenmakros auf Basis `expl3` [The `xparse` package — Documenting macros on the basis of `expl3`]; pp. 39–47

Writing bigger documents in LATEX usually makes an author define his or her own macros and environments. The most common method for authors is to use the `\newcommand` or `\newenvironment` command. The `xparse` package provides functions for the definition of macros with various optional arguments, starred versions or combinations of macros.

Dominik Wassenhoven, `biblatex` mit `tufte-latex` verwenden [Using `biblatex` with `tufte-latex`]; pp. 51–53

Triggered by a question on TEX.sx ('Can I use biblatex with Tufte classes?' at `http://tex.stackexchange.com/questions/45934`), I noticed some issues if the `biblatex` package is used with the Tufte document classes. Fortunately there is a solution which will be presented here.

Dominik Wassenhoven, Aufrechte Klammern in kursivem Text [Straight brackets in italic text]; pp. 51–53

In *The Elements of Typographic Style* Robert Bringhurst recommends typesetting brackets always straight even if they are placed inside italic text: "Use upright (i.e., 'roman') rather than sloped parentheses, brackets and braces, even if the context is italic."

Jürgen Hanneder, Der TEXnische Fortschritt und seine Tücken [TEXnical progress and its pitfalls]; pp. 54–60

The development of TEX in the last decades: fun and frustrations ...

Patrick Gundlach, Strichcodes erzeugen mit LuaTEX [Creating barcodes with LuaTEX]; pp. 61–71

(English version published in *TUGboat* 33:1.)

Christine Römer, Bücher schreiben mit der LATEX-Dokumentenklasse memoir [Writing books with the `memoir` document class]; pp. 72–82

Memoir is a modern and powerful document class, little known in the German TEX world. This may be due to the powerful competitor KOMA-Script, the sub-optimal documentation or the misleading name. In the presence of two modern document classes for typesetting complex publications (`scrbook` and `memoir`) it is not obvious why introductory LATEX books still recommend using the obsolete `book` class.

Herbert Voss, Anwendung aktiver Zeichen [The usage of active characters]; pp. 83–87

It is common knowledge that TEX knows as basic elements primitives and — at the user level — macros, which consist after complete expansion only of primitives. Macros as well as primitives are used with a leading backslash. A special role is taken by active characters which — without the leading backslash — can behave like macros.

Die TEXnische Komödie 3/2012

Markus Kohm, KOMA-Script wird volljährig [KOMA-Script comes of age]; pp. 10–16

In 1994 not only LATEX 2_ε but also KOMA-Script was born, so in 2012 it will be 18 years old. Compared with LATEX 2_ε which was completely developed at birth KOMA-Script had to grow in the last 18 years. It started as a baby, not just complaining at each incident, but keeping its environment busy. His father, me, lost some nerves; he will take the opportunity to reflect on the last 18 years.

Markus Kohm, Kopfzeilentricks mit `scrpage2` [Headline tricks with `scrpage2`]; pp. 17–21

For the 4th edition of the KOMA-Script book I have used a special version of the headline, where the page number is separated from the column title by a small black bar. The page number goes far into the margin. This was created using the KOMA-Script package `scrpage2`.

Clemens Niederberger, Das `xtemplate` Paket [The `xtemplate` package]; pp. 22–30

One of the key steps in the development of LATEX3 is the separation of implementation, layout design and user interface. The `xtemplate` package stands between the first two, in the "designer interface foundation layer". Based on a not necessarily real world example we show the ideas of the package and its usage.

Christine Römer, Lyrik mit LATEX [Lyric Poetry with LATEX]; pp. 31–39

Using the `verse` package and its extensions in `memoir`, poems can be typeset in an appealing way and interpretation aids for the textual analysis can be introduced. There are still some reservations about the visualisation of the metre.

Uwe Ziegenhagen, europass Lebensläufe setzen mit LaTeX [Typesetting europass CVs with LaTeX]; pp. 40–49

In this article two packages, `ecv` and `europecv`, are introduced. Both allow typesetting CVs following the European europass template.

Uwe Ziegenhagen, Briefumschläge beschriften und frankieren mit LaTeX [Printing and stamping envelopes with LaTeX]; pp. 50–54

Letters can be stamped not only using common stamps but also via stamps bought on the Internet. Even with "mobile stamps", letters can be sent. In this article it is shown how envelopes can be stamped using digital stamps bought online.

Rolf Niepraschk, Randnotizen im Literaturverzeichnis [Marginal notes in bibliographies]; pp. 55–59

Examples and motivation for occasional typesetting of marginal notes in a bibliography.

Ralf Meyer, Meine ersten Schritte mit LuaTeX [My first steps with LuaTeX]; pp. 60–67

I will explain how I used LuaTeX to analyse and print results from external tables in an appealing way. I will describe a few issues with LuaTeX concerning special characters and locales that I encountered. In particular, I will mention an issue with a recently published article by Herbert Voß.

Patrick Gundlach, Worttrennungen überprüfen mit LuaLaTeX [Checking hyphenation with LuaLaTeX]; pp. 68–70

For LuaLaTeX there are two packages which help the author to check the hyphenation of his or her text. Using these packages one can avoid bad or faulty hyphenations.

Herbert Voss, TeX Gyre Pagella Math; pp. 71–72

Sample output and usage of the TeX Gyre Pagella OpenType math fonts.

[Received from Herbert Voß.]

ArsTEXnica #13 (April 2012)

ArsTEXnica is the journal of GUIT, the Italian TeX user group (`http://www.guit.sasup.it/`).

Gianluca Pignalberi, Editoriale [From the editor]; p. 3

Claudio Fiandrino, Graphviz e Ti*k*Z [Graphviz and Ti*k*Z]; pp. 4–10

Graphviz is a very powerful tool for drawing graphs. This article tries to explain how to export such graphs as a Ti*k*Z picture in a very simple way.

Gianluca Pignalberi, Salvatore Schirone, Jerónimo Leal, Ancora sugli strumenti per grecisti classici [Again on tools for hellenists]; pp. 11–16

This paper inspects two topics left unsolved in our previous article, namely the transliteration of Greek texts with XeLaTeX and the composition and the translation with parallel text into another language. One or more solutions to both topics will be given.

Claudio Beccari, LaTeX e le lingue romanze alpine [LaTeX and Romance Alpine languages]; pp. 17–24

LaTeX is used to typeset in a variety of languages: at the time of writing, it can handle 74 different languages (plus many variants), some of them used by very few people, some by millions. At the moment, there are no facilities for typesetting documents in the various more or less official Alpine Romance languages. This paper would be a first step in order to extend the LaTeX language capabilities to the various romance languages that are being used by large communities in the European Alps.

Enrico Gregorio, Scrivere un pacchetto per LaTeX3. Il pacchetto `kantlipsum` [Writing a package for LaTeX3. The package `kantlipsum`]; pp. 24–29

The `kantlipsum` package is used as an example to illustrate some programming techniques with the LaTeX3 language.

Font e TeX, Fonts and TeX; pp. 30–31

Italian translation of `http://tug.org/fonts`.

[Received from Gianluca Pignalberi.]

MAPS 42 (2011)

MAPS is the publication of NTG, the Dutch language TEX user group (`http://www.ntg.nl`).

[Received from Wybo Dekker.]

The Treasure Chest

This is a list of selected new packages posted to CTAN (`http://ctan.org`) from April to August 2012, with descriptions based on the announcements and edited for brevity.

Entries are listed alphabetically within CTAN directories. A few entries which the editors subjectively believed to be of especially wide interest or otherwise notable are starred; of course, this is not intended to slight the other contributions.

We hope this column and its companions will help to make CTAN a more accessible resource to the TEX community. Comments are welcome, as always.

◇ Karl Berry
`http://tug.org/ctan.html`

fonts

`adobecaslon` in `fonts/psfonts/adobe`
Metric support for Adobe Caslon.

`bguq` in `fonts`
The Begriffsschrift quantifier character.

`countriesofeurope` in `fonts`
Support for the Countries of Europe font.

`imprintmtshadow` in `fonts`
Support for the Monotype Imprint Shadow fonts.

`lsabon` in `fonts`
Support for the Linotype Sabon fonts.

`minion2newtx` in `fonts`
Add-on for `newtx`: metrics for MinionPro v2.

* `newtx` in `fonts`
Improved metrics and options for `txfonts`.

`sansmathaccent` in `fonts`
Correct accent placement in `beamer` and `sfmath`.

graphics

`hf-tikz` in `graphics/pgf/contrib`
Ti*k*Z highlighting of formulas, in part or completely.

`pgf-blur` in `graphics/pgf/contrib`
Ti*k*Z support for blurred/faded/fuzzy shadows.

`pst-ode` in `graphics/pstricks/contrib`
PSTricks support for solving initial value problems for sets of ordinary differential equations.

info

`latexfileinfo-pkgs` in `info`
Comparison of packages for file version information.

macros/generic

`dowith` in `macros/latex/generic`
Apply a command to elements of a list.

macros/latex/contrib

`articleingud` in `macros/latex/contrib`
Support for articles in *Ingenieria Review*.

* `autonum` in `macros/latex/contrib`
Automatically number only those equations which are referenced.

`basque-book` in `macros/latex/contrib`
Document class for books in Basque.

`basque-date` in `macros/latex/contrib`
Current date in Basque, handling declination issues.

`calculator` in `macros/latex/contrib`
Handle algebraic operations and evaluate elementary functions and derivatives.

`cellwise` in `macros/latex/contrib`
Build tables cellwise with freely formatted individual cells.

`codicefiscaleitaliano` in `macros/latex/contrib`
Test consistency of the Italian personal fiscal code.

`embrac` in `macros/latex/contrib`
Upright parentheses and brackets within italic text.

`enotez` in `macros/latex/contrib`
Support for endnotes, possibly nested.

`famt` in `macros/latex/contrib`
Support for FAMT Institute reports and notices.

`fitr` in `macros/latex/contrib`
Support for the PDF FitR destination type.

`fixmetodonotes` in `macros/latex/contrib`
Highlight and manage FIXME-like notations.

`fnpct` in `macros/latex/contrib`
Footnote kerning.

`footnotebackref` in `macros/latex/contrib`
Hyperlink from a footnote to its occurrence in the main text.

`foreign` in `macros/latex/contrib`
Typeset foreign words in different typefaces.

`frege` in `macros/latex/contrib`
Support for Frege's Begriffsschrift notation.

`GS1` in `macros/latex/contrib`
Support for the GS1 codes and some barcodes.

`harnon-cv` in `macros/latex/contrib`
CV class with support for a vertical timeline of experience.

`ifthenx` in `macros/latex/contrib`
Define conditionals such as `\isinteger`, `\isnumber`, `\fileexists`, `\classloaded`, and many others.

`incgraph` in `macros/latex/contrib`
Including graphics on the full paper size, including bookmarking.

lastbib in macros/latex/contrib
Provide \LastBib for the total count of references.

lgrx in macros/latex/contrib
Typesetting Greek in the LGR encoding.

lmac in macros/latex/contrib
Including support files.

lstaddons in macros/latex/contrib
Add-ons for listings.

multienv in macros/latex/contrib
Multiple environments using a key–value syntax.

mwe in macros/latex/contrib
Support for creating minimal working examples.

mycv in macros/latex/contrib
Curriculum vitae with many layouts and decorations.

pgfkeyx in macros/latex/contrib
Handle active characters in pgfkeys, and more.

poetrytex in macros/latex/contrib
Typesetting anthologies of poetry.

python in macros/latex/contrib
Embed Python code and output in a LaTeX document.

*regexpatch in macros/latex/contrib
Generalized macro patching, based on l3regex.

shadowtext in macros/latex/contrib
Produce customizable drop shadow for text.

showcharinbox in macros/latex/contrib
Show characters inside a box.

statrep in macros/latex/contrib
Display SAS code and its results together.

substitutefont in macros/latex/contrib
Combine font families.

typeface in macros/latex/contrib
Generalized setup of default Type 1 fonts.

xpinyin in macros/latex/contrib
Automatically add pinyin to Chinese characters.

xpunctuate in macros/latex/contrib
Insert punctuation only if necessary, as with xspace.

zhnumber in macros/latex/contrib
Expandable printing of Chinese representations of numbers, using LaTeX3.

macros/latex/contrib/babel-contrib

friulan in m/l/c/babel-contrib
Babel support for Friulan.

romansh in m/l/c/babel-contrib
Babel support for Romansh (Rumantsch Grischun).

macros/latex/contrib/beamer-contrib

dynblocks in m/l/c/beamer-contrib
Customize aspect ratios and dimensions of blocks in a presentation.

hobete in m/l/c/beamer-contrib
Unofficial beamer theme for the Univ. of Hohenheim.

macros/latex/contrib/biblatex-contrib

biblatex-bwl in m/l/c/biblatex-contrib
biblatex support for FU-Berlin.

biblatex-phys in m/l/c/biblatex-contrib
biblatex support for the AIP and APS styles.

biblatex-swiss-legal in m/l/c/biblatex-contrib
biblatex support for Swiss legal citation standards.

macros/luatex

chickenize in macros/luatex/generic
Manipulate input or output tokens of any Lua(LA)TeX document.

lua-check-hyphen in macros/luatex/latex
Make it easy to spot incorrect hyphenations.

luatexja in macros/luatex/generic
Plain and LaTeX 2ε support for typesetting Japanese with Lua(LA)TeX.

luaxml in macros/luatex/generic
Reading and serializing XML files.

odsfile in macros/luatex/latex
Typeset OpenDocument Spreadsheet (.ods) files as tables.

macros/xetex

imscls in macros/xetex/latex
Iranian Mathematical Society proceedings support.

support

csvtolatex in support
Visual Basic script converting CSV data to LaTeX.

eitl in support/texlive
Install current TeX Live and add-ons over the net.

systems

bakoma-mac in systems/mac
New TeX distribution supporting Mac OS X.

Book review: *About more alphabets: The types of Hermann Zapf*

Boris Veytsman

Jerry Kelly, *About more alphabets: The types of Hermann Zapf*. Preface by Robert Bringhurst. The Typophiles. 2011. 112 pp., 4.5″ × 7″. Hardcover, US$35.00. ISBN 9780984274406.

The name of Hermann Zapf, the Wizard of Fonts and permanent honorary member of the Board of TUG, is well known in our community. This small book describes his life and work, and is a very welcome addition to the Wizard's *Alphabet Stories,* which was reviewed by Hans Hagen and Taco Hoekwater in 2007 (*TUGboat* 28:2, p. 174). The book is written by Jerry Kelly, and, like Kelly's other books, has a very interesting text accompanied by well-chosen illustrations.

Kelly divides Zapf's work into three periods: the early period during the metal type era, the middle period, when new technologies like photocomposition were dominating, and the later period when digital technologies became ubiquitous. For each period Kelly selects Zapf's fonts that best show the evolution of the master, from the early but already elegant Gilgengart to the breathtaking in its innovation and beauty Zapfino. The examples are meticulously chosen and illustrate the main qualities of Zapf's talent: the deep understanding of technology, the attention to details and the striving for perfection.

Usually the font design field tends to be conservative (after all we call the fonts designed in the late 18th century 'Modern'). The more amazing is the way Hermann Zapf has always been open to new ideas. He saw the growth of the major typographic technologies of the last and current centuries, and actively participated in it, contributing ideas that shaped the industry. Kelly shows examples of this throughout Zapf's career.

A TEX user would be interested in the parts of the book describing Zapf's work with DEK on Euler fonts and his contributions to the typesetting algorithms (some of them are implemented in the modern TEX engines).

About more alphabets is beautifully illustrated and accompanied by type specimens of more than two dozen of Zapf's fonts. It is well typeset in Comenius, a great and rare font (of course, designed by Zapf). There are a thousand copies of this book printed, plus seventy-five additional copies specially bound and signed by the author. I am sure each copy of both editions will become a prized rarity.

The book has a well-written foreword by another great figure of modern typography, Robert Bringhurst. It is noted there, "We evidently think (in defiance of all logic) that what we read or write matters far more than how it's read or written, and that letterforms are just a way to get there, as a door knob is a way to open a door. At their best, though, letterforms are more like sailboats and cellos. They are works of art that beg to be used as well as admired." These words, like the sound of a tuning fork, set the tone of the book. *About more alphabets* is a fitting tribute to the art of letterforms and to its master, Hermann Zapf. I must note that the book itself is a piece of art, splendidly produced by Jerry Kelly. I would recommend it to anybody interested in letterforms, typography and book design — or just being able to appreciate and enjoy beautiful books.

◇ Boris Veytsman
Computational Materials Science Center, MS 6A2
George Mason University
Fairfax, VA 22030
USA
borisv (at) lk dot net
http://borisv.lk.net

TEX 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^o 2^a
08015 Barcelona, Spain
+34 932267827
Email: `m.aicart (at) ono.com`
Web: `http://www.edilatex.com`
We provide, at reasonable low cost, LaTeX or TeX page layout and typesetting services to authors or publishers world-wide. We have been in business since the beginning of 1990. For more information visit our web site.

Dangerous Curve
PO Box 532281
Los Angeles, CA 90053
+1 213-617-8483
Email: `typesetting (at) dangerouscurve.org`
Web: `http://dangerouscurve.org/tex.html`
We are your macro specialists for TeX or LaTeX fine typography specs beyond those of the average LaTeX macro package. If you use XeTeX, we are your microtypography specialists. We take special care to typeset mathematics well.

Not that picky? We also handle most of your typical TeX and LaTeX typesetting needs.

We have been typesetting in the commercial and academic worlds since 1979.

Our team includes Masters-level computer scientists, journeyman typographers, graphic designers, letterform/font designers, artists, and a co-author of a TeX book.

Hendrickson, Amy
Brookline, MA, USA
Email: `amyh (at) texnology.com`
Web: `http://www.texnology.com`
LaTeX macro writing our speciality for more than 25 years: macro packages for major publishing companies, author support; journal macros for American Geophysical Union, Proceedings of the National Academy of Sciences, and many more.

Scientific journal and e-journal design and production.

Hendrickson, Amy (cont'd)
LaTeX training, at MIT, Harvard, many more venues. Customized on site training available.

Please visit our site for samples, and get in touch. We are particularly glad to take on adventurous new uses for LaTeX, for instance, web based report generation including graphics, for bioinformatics or other applications.

Latchman, David
4113 Planz Road Apt. C
Bakersfield, CA 93309-5935
+1 518-951-8786
Email: `texnical.designs (at) gmail.com`
Web: `http://www.elance.com/s/dlatchman`
Proficient and experienced LaTeX typesetter for books, monographs, journals and papers allowing your documents and books to look their possible best especially with regards to technical documents. Graphics/data rendered either using Ti*k*Z or Gnuplot. Portfolio available on request.

Moody, Trent
1981 Montecito Ave.
Mountain View, CA 94043
+1 650-283-7042
Email: `trent.moody (at) ymail.com`
Construction of technical documents with mathematical content from hand written (or partially formatted) sources. Delivered documents will be `.tex` and `.pdf` files produced with TeX or/and LaTeX. Delivered documents can be publication ready manuscripts, macro libraries for modular document development, or mathematical libraries for document reuse.

I am an independent contractor with a PhD in mathematical physics from the University of California, Santa Cruz.

Peter, Steve
295 N Bridge St.
Somerville, NJ 08876
+1 732 306-6309
Email: `speter (at) mac.com`
Specializing in foreign language, multilingual, linguistic, and technical typesetting using most flavors of TeX, I have typeset books for Pragmatic Programmers, Oxford University Press, Routledge, and Kluwer, among others, and have helped numerous authors turn rough manuscripts, some with dozens of languages, into beautiful camera-ready copy. In addition, I've helped publishers write, maintain, and streamline TeX-based publishing systems. I have an MA in Linguistics from Harvard University and live in the New York metro area.

Shanmugam, R.
No. 38/1 (New No. 65), Veerapandian Nagar, Ist St.
Choolaimedu, Chennai-600094, Tamilnadu, India
+91 9841061058
Email: rshanmugam92 (at) yahoo.com
As a Consultant, I provide consultation, training, and full service support to individuals, authors, typesetters, publishers, organizations, institutions, etc. I support leading BPO/KPO/ITES/Publishing companies in implementing latest technologies with high level automation in the field of Typesetting/Prepress, ePublishing, XML2PAGE, WEBTechnology, DataConversion, Digitization, Cross-media publishing, etc., with highly competitive prices.
I provide consultation in building business models & technology to develop your customer base and community, streamlining processes in getting ROI on our workflow, New business opportunities through improved workflow, Developing eMarketing/E-Business Strategy, etc. I have been in the field BPO/KPO/ITES, Typesetting, and ePublishing for 16 years, handled various projects. I am a software consultant with Master's Degree. I have sound knowledge in TEX, LATEX2ε, XMLTEX, Quark, InDesign, XML, MathML, DTD, XSLT, XSL-FO, Schema, ebooks, OeB, etc.

Sharma, Ganesh Kumar
A - 251 / 1, Opposite Primary School,
Shastri Nagar, Delhi 110052, India
+91 9810748682, 9013121611
Email: ganeshsharma (at) yahoo.com
I am a Master of Computer Applications (MCA) degree holder. I am well versed with MetaPost, HTML, MathML, Java, CSS, PHP, Unix shell scripting, C++, Ti*k*Z, Gnuplot and PostScript etc.

As a consultant and service provider, I am handling LATEX and XƎLATEX composition to technical illustration, editorial services for: project management of conference proceedings; class/style files creation for LATEX publications; a full management service for journals including correspondence with authors and issue make-up, including manuscript Preparation (pagination / composition, copy editing and proof reading), scanning and graphics designing, origination from handwritten manuscript or use of author-supplied code (TEX or word processor), and author support; the supply of HTML, PDF files (including hyperlinks and bookmarks) and other coding for electronic publication. I can typeset the books in Sanskrit and Hindi languages using LATEX very well.

Currently, I am giving editorial services to many universities, reputed publishers and multinational companies, research groups etc.

Sievers, Martin
Im Treff 8, 54296 Trier, Germany
+49 651 4936567-0
Email: info (at) schoenerpublizieren.com
Web: http://www.schoenerpublizieren.com
As a mathematician with ten years of typesetting experience I offer TEX and LATEX services and consulting for the whole academic sector (individuals, universities, publishers) and everybody looking for a high-quality output of his documents.

From setting up entire book projects to last-minute help, from creating individual templates, packages and citation styles (BIBTEX, biblatex) to typesetting your math, tables or graphics — just contact me with information on your project.

Sofka, Michael
8 Providence St.
Albany, NY 12203
+1 518 331-3457
Email: michael.sofka (at) gmail.com
Skilled, personalized TEX and LATEX consulting and programming services.

I offer over 25 years of experience in programming, macro writing, and typesetting books, articles, newsletters, and theses in TEXand LATEX: Automated document conversion; Programming in Perl, C, C++ and other languages; Writing and customizing macro packages in TEX or LATEX; Generating custom output in PDF, HTML and XML; Data format conversion; Databases.

If you have a specialized TEX or LATEX need, or if you are looking for the solution to your typographic problems, contact me. I will be happy to discuss your project.

Veytsman, Boris
46871 Antioch Pl.
Sterling, VA 20164
+1 703 915-2406
Email: borisv (at) lk.net
Web: http://www.borisv.lk.net
TEX and LATEX consulting, training and seminars. Integration with databases, automated document preparation, custom LATEX packages, conversions and much more. I have about seventeen years of experience in TEX and thirty years of experience in teaching & training. I have authored several packages on CTAN, published papers in TEX related journals, and conducted several workshops on TEX and related subjects.

TUG Institutional Members

American Mathematical Society,
Providence, *Rhode Island*

Aware Software, Inc.,
Midland Park, *New Jersey*

Banca d'Italia,
Roma, *Italy*

Center for Computing Sciences,
Bowie, *Maryland*

Certicom Corp.,
Mississauga, *Ontario*, *Canada*

CSTUG, *Praha*, *Czech Republic*

diacriTech, *Chennai*, *India*

Florida State University,
School of Computational Science
and Information Technology,
Tallahassee, *Florida*

IBM Corporation,
T J Watson Research Center,
Yorktown, *New York*

Institute for Defense Analyses,
Center for Communications
Research, *Princeton*, *New Jersey*

LAMFA CNRS UMR 6140,
Amiens, *France*

MacKichan Software, Inc.,
Washington/New Mexico, *USA*

Marquette University,
Department of Mathematics,
Statistics and Computer Science,
Milwaukee, *Wisconsin*

Masaryk University,
Faculty of Informatics,
Brno, *Czech Republic*

MOSEK ApS,
Copenhagen, *Denmark*

New York University,
Academic Computing Facility,
New York, *New York*

Springer-Verlag Heidelberg,
Heidelberg, *Germany*

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*

Calendar

2012

Aug 23 – 26 TEXperience 2012 (5th TEXperience Conference, organized by CSTUG), Morávka, The Czech Republic. katedry.osu.cz/kma/TeXperience2012

Sep 4 – 7 ACM Symposium on Document Engineering, Paris, France doceng2012.wp.institut-telecom.fr

Sep 16 – 21 XML Summer School, St Edmund Hall, Oxford University, Oxford, UK. www.xmlsummerschool.com

Oct 1 *TUGboat* **33**:3, submission deadline (regular issue)

Oct 5 – 7 Oak Knoll Fest XVII, and Fine Press Book Association annual meeting, New Castle, Delaware. www.oakknoll.com/fest

Oct 8 – 12 EuroTEX 2012, 6th International ConTEXt user meeting, and DANTE Herbsttagung and 47th meeting, "Recreational Uses of TEX", Breskens, The Netherlands. meeting.contextgarden.net/2012

Oct 10 – 14 Association Typographique Internationale (ATypI) annual conference, Theme: "between black and white", Hong Kong. www.atypi.org/hong-kong-2012

Oct 12 – 13 American Printing History Association's 37th annual conference, "At the Crossroads: Living Letterform Traditions", Columbia College, Chicago, Illinois. www.printinghistory.org/programs/conference/conference_2012.php

Oct 18 Beatrice Warde Memorial Lecture, "You Can't Repeat the Past", by Paul Barnes, St Bride Library, London, England. stbride.org/events

Oct 19 – 20 TYPO London, "Social", sponsored by FontShop. www.atypi.org/events/typo-london

Oct 20 UK-TUG Annual Meeting, Oxford, UK. uk.tug.org

Oct 26 – 30 ASIS&T 2012, 75th Annual Meeting, "Information, Interaction, Innovation: Celebrating the Past, Constructing the Present and Creating the Future", American Society for Information Science and Technology, Baltimore, Maryland. www.asis.org/asist2012

Oct 27 TEX Conference Japan 2012, Kyoto University, Japan. oku.edu.mie-u.ac.jp/texconf12/

Oct 27 GuIT 9th Annual Meeting 2012, Conference Center of the University of Naples Federico II, Italy. www.guitex.org/home/meeting

Nov 9 – 10 13. IBG-Jahrestagung in München, "Das E-Book. Herausforderung und Chance für die Buch- und Verlagswelt", Internationalen Buchwissenschaftliche Gesellschaft, München, Germany. www.buchwiss.de

2013

Jan 15 Conference,"The Design of Understanding", St Bride Library, London, England. stbride.org/events

Mar 11 *TUGboat* **34**:1, submission deadline (regular issue)

Jul 8 *TUGboat* **34**:2, submission deadline (regular issue)

Jul 18 – 21 SHARP 2013, "Geographies of the Book", Society for the History of Authorship, Reading & Publishing, University of Pennsylvania, Philadelphia. www.library.upenn.edu/exhibits/lectures/SHARP2013

Sep 26 – 27 The Eleventh International Conference on the Book, Universität Regensburg Universitätsbibliothek, Regensburg, Germany booksandpublishing.com/the-conference

Status as of 20 August 2012

For additional information on TUG-sponsored events listed here, contact the TUG office (+1 503 223-9994, fax: +1 815 301-3568. e-mail: office@tug.org). For events sponsored by other organizations, please use the contact address provided.

A combined calendar for all user groups is online at texcalendar.dante.de.

Other calendars of typographic interest are linked from tug.org/calendar.html.

Introductory

Intermediate

Intermediate Plus

Advanced

Contents of publications from other TEX groups

Reports and notices