TUGboat

Volume 25, Number 1 / 2004
Practical TEX 2004 Conference Proceedings

	2	Lance Carnes / *Highlights of the Practical TEX 2004 conference*
	4	Conference program, delegates, and sponsors
	6	Karl Berry / *Welcome to Practical TEX 2004*
Keynotes	7	Nelson Beebe / *25 Years of TEX and METAFONT: Looking back and looking forward — TUG 2003 keynote address*
	31	Peter Flynn / *TEX and the interfaces — Practical TEX 2004 keynote address*
Talks	35	Hàn Thế Thành / *Micro-typographic extensions of pdfTEX in practice*
	39	Eitan Gurari / *TEX4ht: HTML production*
	48	Hans Hagen / *The state of ConTEXt*
	52	Steve Grathwohl / *A simple book design in ConTEXt*
	58	Steve Peter / *TEX and linguistics*
	63	Brooks Moses / *MetaPlot, MetaContour, and other collaborations with METAPOST*
	71	William Richter / *TEX and scripting languages*
	89	Nelson Beebe / *A bibliographer's toolbox*
Reports	105	Taco Hoekwater / *MetaPost developments*
	105	Giuseppe Bilotta / *The ℵ (Aleph) project*
	108	Hans Hagen / *The TEX Live 2004 collection*
News & Announcements	112	Calendar
	114	TUG 2005, 23–25 August 2005, Wuhan, China
	c3	Practical TEX 2005, 14–17 June 2005, Chapel Hill, North Carolina
TUG Business	120	Recognition of support from Apple
	115	TUG membership application
	116	Institutional members
Advertisements	116	TEX consulting and production services
	114	*The LATEX Companion*, 2nd edition, by Frank Mittelbach et al.
	117	*Easy Table*, Khanh Ha
	118	River Valley Technologies
	118	MacKichan Software, Inc.
	119	Carleton Production Centre
	119	Cheryl Ponchin Training
	120	Personal TEX, Inc.

TEX Users Group

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

Memberships and Subscriptions

2004 dues for individual members are as follows:

- Ordinary members: $75.
- Students/Seniors: $45.

The discounted rate of $45 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: `http://www.tug.org`.

TUGboat subscriptions are available to organizations and others wishing to receive *TUGboat* in a name other than that of an individual. Subscription rates: $85 a 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. For further information, contact the TUG office (`office@tug.org`) or see our web site.

TEX is a trademark of the American Mathematical Society.

Printed in U.S.A.

Addresses

General correspondence, payments, etc.
TEX Users Group
P. O. Box 2311
Portland, OR 97208-2311
U.S.A.

Delivery services, parcels, visitors
TEX Users Group
1466 NW Naito Parkway
Suite 3141
Portland, OR 97209-2820
U.S.A.

Telephone

+1 503 223-9994

Fax

+1 503 223-3960

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`

World Wide Web

`http://www.tug.org/`
`http://www.tug.org/TUGboat/`

Problems not resolved?

The TUG Board wants to hear from you:
Please email `board@tug.org`.

[printing date: March 2005]

Practical TEX 2004 Proceedings

San Francisco, California, USA
July 19–22, 2004

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP
TUGboat Editor Barbara Beeton
Proceedings Editor Karl Berry

Volume 25, Number 1 • 2004
Portland • Oregon • U.S.A.

Highlights of the Practical TEX 2004 Conference

Lance Carnes
Personal TEX, Inc.
725 Greenwich Street, Suite 210
San Francisco, CA 94133, USA
lcarnes@pctex.com

Delegates came from the US and from around the world — Ireland, England, The Netherlands, Germany, Australia, and Vietnam — to the Holiday Inn Fisherman's Wharf in San Francisco, the site of the first annual PracTEX meeting, July 19–22, 2004.

The purpose of the conference was to focus on the practical, day-to-day use of LATEX, TEX, ConTEXt, and other applications. Speakers gave talks and held workshops on a wide range of topics, including TEX on the Web, basic and intermediate LATEX, ConTEXt (a comprehensive TEX-based documentation system), and more.

The conference began with a welcome by TUG President Karl Berry, and a keynote address by Peter Flynn titled "TEX and the Interface". The conference program consisted of numerous presentations — a few are presented in the first issue of the new TUG online publication, The PracTEX Journal (`http://tug.org/pracjourn/2005-1`), and the bulk are published in this volume of *TUGboat*.

On Monday afternoon long-time TUG contributor Wendy McKay and Apple Computer Product Manager Ernest Prabhakar held an Special Interest Group meeting on TEX running on Mac OS X. The meeting was video conferenced using technology lent by Marratech.

For beginning users and those who wanted a refresher, workshops on beginning and intermediate LATEX were offered each morning of the conference, in parallel with talks. Sue DeMeritt and Cheryl Ponchin (Center for Communications Research, and TUG board members) presented these popular workshops.

During the conference there were daily Q&A sessions, and on the final day a panel discussion was held. These sessions were valuable for getting input from attendees. The consensus seemed to be a need for more education in the use of LATEX and TEX, and the need for more sources of practical information.

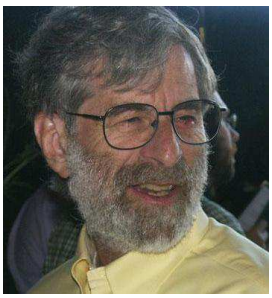
On the day after the conference three one-day courses were held: Cheryl Ponchin and Sue DeMeritt conducted a workshop on Intermediate LATEX; Peter Flynn (Silmaril Ltd., Ireland) taught "TEX on the Web"; and Hans Hagen (Pragma ADE, The Netherlands) taught an introductory class on his ConTEXt system.

Hans Hagen, Practical TEX Man of the Year

Keynote speaker Peter Flynn

Attendees were treated to three social events: an opening reception, a San Francisco treasure hunt, and a Chinese banquet. The treasure hunt challenged teams of attendees to decipher clues as they explored San Francisco's North Beach and Chinatown districts. (When it was learned that Don Knuth was entering a team, a special Wizards Hunt was added with more difficult clues.)

Don Knuth



Leslie Lamport

The banquet was at the Empress of China restaurant, which has a commanding view of the northern area of San Francisco, including Alcatraz, the Bay Bridge and the financial district. Some special guests at the banquet were Don and Jill Knuth, and Leslie Lamport. Barbara Mastrian of Rutgers introduced Barbara Miller, a pioneer TEX

user who over the years has taught many others how to navigate the boxes and glue.

Barbara Mastrian and TEX pioneer Barbara Miller

Treasure Hunt winners (Heidi Sestrich, Alistair Smith, Jenny Levine, Terri Spence, Alan Wetmore)

There were two coffee breaks and lunch each day, which provided for plenty of attendee interaction during the three-day conference. There was a cybercafe with an ample number of computers, lent by Apple Computer, which delegates used for checking email and experimenting with new applications.

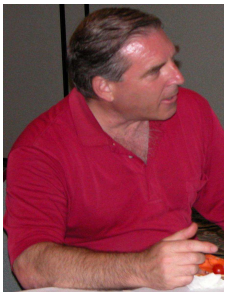
It was great to see several TEX and TUG pioneers again: Cal Jackson, Dave Fuchs, Art Ogawa, Nelson Beebe, and Peter Flynn.

From the follow-up comments it seemed everyone had a good conference and an enjoyable visit to San Francisco. See `http://tug.org/practicaltex2004/post.html` for more photos and information.

The next PracTEX conference will be held June 14–17, 2005, in Chapel Hill, North Carolina, hosted by Steve Grathwohl and the Duke University Press. See the web site for information and registration: `http://tug.org/practicaltex2005`.

Karl Berry, TUG President

Robin Laakso, TUG Executive Director



Lance Carnes, conference host

(Photo credits: Tim Null, Alan Wetmore, and `mistersf.com`.)

Practical TEX 2004

Sponsors

TEX Users Group ■ Personal TEX, Inc. ■ Apple Computer
Addison-Wesley ■ Carleton Production Centre ■ Marratech, Inc. ■
MacKichan Software, Inc. ■ River Valley Technologies

Thanks to all!

Acknowledgements

and heartfelt thanks to:

- William Adams, for the Apple acknowledgement and help with the mug image.
- Duane Bibby, for the typically wonderful conference and notepad drawings.
- Wendy McKay, for organizing the Mac OS X session.
- Kevin O'Malley, for the Mac OS X slide show.

Conference committee

Karl Berry ■ Lance Carnes ■ Sue DeMeritt ■ Robin Laakso ■ Steve Peter ■ Cheryl Ponchin

Participants

David Allen, University of Kentucky
Mitchell Bakos, Professional Publications, Belmont, CA
Kaveh Bazargan, TUGIndia and River Valley Technologies
Nelson Beebe, University of Utah
Karl Berry, TEX Users Group
Linda Bethel, Graduate School of Business, Stanford University
Jeannie Brown, University of California, Irvine
Brian Carnes, Personal TEX, Inc.
Lance Carnes, Personal TEX, Inc.
Barry Dale, Caloundra, Australia
Sue DeMeritt, Center for Communications Research, La Jolla, CA
Michael Dickerson, Pomona College, CA
Sandra Farrier, Washington, DC
Ronald Fehd, Atlanta, GA
Peter Flynn, Silmaril Consultants
David Fuchs, Palo Alto, CA
Steve Grathwohl, Duke University Press
Eitan Gurari, Ohio State University
Hàn Thế Thành, University of Education, Ho Chi Minh City
Hans Hagen, NTG and Pragma ADE
Joseph Hesse, Saint Paul College
L. Carole Holbrook, Oak Ridge Laboratory, TN
Baden Hughes, University of Melbourne
Ned Hummel, University of Nebraska Lincoln
Susan Huot, Journal of Financial and Quantitative Analysis, Seattle
Calvin Jackson, California Institute of Technology
David Jones, American Mathematical Society
N.G. Kalivas, Westminster School, UK
Richard Koch, University of Oregon
Robin Laakso, TEX Users Group
Alice Leonhardt, Rutgers University, NJ
Jenny Levine, Duke University Press
Donna Magnani, American Physical Society, Ridge, NY
Barbara Mastrian, Rutgers University, NJ
Denise McCall, Institute for Defense Analysis, Princeton, NJ
Wendy McKay, Control and Dynamical Systems, California Institute of Technology
Barbara Miller, University of California, Riverside
Brooks Moses, Stanford University
Noelle Noble, Fred Hutchinson Cancer Research Center, Seattle
Tim Null, San Jose, CA
Arthur Ogawa, TEX Consultants, Three Rivers, CA
Elliott Pearl, Toronto, Canada
Steve Peter, Beech Stave Press, NJ
Cheryl Ponchin, Center for Communications Research, Princeton, NJ
Andrew Porter, Livermore, CA
K. David Prince, College of Engineering, University of Washington
William Richter, Texas Life Insurance Co.
Volker R.W. Schaa, Dante e.V.
Catherine Schrott, Professional Publications, Belmont, CA
Anita Schwartz, University of Delaware
Heidi Sestrich, Carnegie-Mellon University
Kathy Sheldon, Pomona College, CA
Alistair Smith, Sunrise Setting, Devon, UK
Doug Smylie, York University, Toronto
Terri Spence, Duke University Press
Anne Taub, ISIS, University of California, Irvine
Larry Thomas, Saint Peter's College, NJ
Alan Wetmore, US Army, Adelphi, MD
Faye Yeager, University of California, Berkeley

Practical TEX 2004 — program and information

Sunday July 18	3–5 pm	*registration*	
	5–7 pm	*reception*	

▪ **Track 2: Introduction to LaTeX class.** Starting at 10:30 am Monday, and 9 am Tuesday and Wednesday, and ending at lunchtime each day, Sue DeMeritt and Cheryl Ponchin will teach a continuing class on beginning and intermediate LaTeX, with no prerequisites. Participants can choose whether to attend this class or the morning talks.

▪ **Mac OS X & TeX session.** Starting at lunch time Monday and continuing into the afternoon, a round-table discussion on Mac OS X, led by Hans Hagen, Wendy McKay, and Ernest Prabhakar from Apple.

Monday July 19	9 am	Karl Berry, TeX Users Group	*Welcome*
	9:15 am	Peter Flynn, Silmaril Consultants	*Keynote address: TeX and the interface*
	10:15 am	*break*	
	10:30 am	Eitan Gurari, Ohio State University	*TeX4ht: HTML production*
	11:15 am	Kaveh Bazargan, River Valley Technologies	*LaTeX to MathML and back: A case study of Elsevier journals*
	11:45 pm	Hans Hagen, NTG, Pragma ADE	*The pros and cons of PDF*
	12:30 pm	*lunch*	
	2:00 pm	Jenny Levine, Duke University Press	*Label replacement in graphics*
	2:30 pm	David Allen, University of Kentucky	*Screen presentations, manuscripts, and posters from the same LaTeX source*
	3:15 pm	*break*	
	3:30 pm	Baden Hughes, University of Melbourne	*TeX and XML*
	4 pm	Hàn Thế Thành, University of Education, Ho Chi Minh City	*Micro-typographic extensions of pdfTeX in practice*
	4:45 pm	q & a	*moderator: Lance Carnes*
Tuesday July 20	9 am	Volker R.W. Schaa, Dante e.V.	*pdfTeX and XML workflow for conference proceedings*
	9:45 am	Anita Schwartz, University of Delaware	*Paperless dissertations at the University of Delaware*
	10:30 am	*break*	
	10:45 am	Hans Hagen	*MetaPost: More than math and fonts*
	11:45 am	Brooks Moses, Stanford University	*MetaPlot, MetaContour, and other collaborations with MetaPost*
	12:30 pm	*lunch*	
	1:30 pm	Cheryl Ponchin, Ctr. for Comm. Research	*LaTeX survey*
	2:15 pm	Steve Grathwohl, Duke University Press	*What is ConTeXt, that we should be mindful of it?*
	3 pm	*break*	
	3:15 pm	William Richter, Texas Life Insurance Co.	*TeX and scripting languages*
	4 pm	q & a	*moderator: Karl Berry*
		social events	
	5 pm	*treasure hunt*	
	7:30 pm	*banquet*	
Wednesday July 21	9 am	Nelson Beebe, University of Utah	*A bibliographer's toolbox*
	9:45 am	David Jones, American Mathematical Soc.	*The amsrefs package*
	10:30 am	*break*	
	10:45 am	Steve Peter, Beech Stave Press	*TeX and linguistics*
	11:30 am	Hans Hagen	*ConTeXt*
	12:30 pm	*lunch*	
	1:30 pm	Steve Grathwohl	*70 years of the Duke Mathematical Journal online*
	2:15 pm	q & a	*moderator: Baden Hughes*
	3 pm	*break*	
	3:15 pm	panel: Digital Publishing	*moderator: Lance Carnes; Kaveh Bazargan, Karl Berry, Peter Flynn, David Fuchs, Hans Hagen.*
	4 pm	*end*	
Thursday July 22		**additional courses**	
		Peter Flynn	*Practical TeX on the Web*
		Sue DeMeritt, Cheryl Ponchin	*Intermediate and Advanced LaTeX*
		Hans Hagen	*Introduction to ConTeXt*

Welcome to Practical TEX 2004

Karl Berry
TEX Users Group
P. O. Box 2311
Portland, OR 97208-2311
USA
karl@freefriends.org
http://freefriends.org/~karl/

The theme of this conference is practical TEX, and indeed, TEX has always been superbly practical. Some might say *too* practical. When Donald E. Knuth devised TEX to typeset his monumental *Art of Computer Programming* volumes, he did not originally expect it to have such universal application — these days to almost any kind of document printed in virtually any language. As a result, some of his more ad hoc decisions, perfectly reasonable for his original purpose, have had ramifications of unfortunately long standing. For instance, the rather idiosyncratic input syntax and difficult extension language.

Nevertheless, TEX has remained a viable program for document production for over two decades, with no end in sight. I don't know of any other widespread application software that has had such a lifetime. In large part, this is because Knuth had the foresight to make TEX extensible in many ways. For example, the core TEX program knows nothing of graphics; yet it has been adapted to essentially all the new graphics programs and file formats as they have come along. A number of the papers here will focus on this.

With the advent of the World Wide Web, a new trend has arisen: the desire to reuse the same document source in multiple contexts: in print, for online display, as data for searches, and more. TEX documents, and especially LATEX and ConTEXt documents, have always had the capability to be logically structured, and thus have adapted well to our new Internet world.

In recent years, the emphasis on logical document markup and structure has grown ever stronger, and some of the typesetting processes invented for TEX have been formalized by the W3C and other standards bodies. These new initiatives suffice for many purposes, and the conference discussed them and their connections with TEX at length. Still, for achieving Knuth's goal of the very highest quality typographic output, to my knowledge TEX remains unsurpassed — a practical tool of the highest order.

A brief introduction to Peter Flynn, our keynote speaker, is in order. Peter has been involved with TEX for many years from his post at University College in Cork, Ireland. We recently devoted an entire issue of *TUGboat* to his excellent introduction to and discussion of LATEX, entitled *Formatting Information*. He has worked extensively with HTML, XML, SGML and many other markup languages, including integration with TEX. In addition to the paper presented in this volume, he taught a workshop at the conference on Practical TEX on the Web.

Lastly, some acknowledgements. On behalf of TUG and the conference, I first thank our corporate sponsors: Personal TEX for major support, and Apple for the loan of the computers. I also thank Addison-Wesley, Carleton Production Centre, Marratech, MacKichan Software, and River Valley Technologies for their important contributions.

On the personal side, my mom, who attended the conference opening, helped me extensively with this small debut as a public speaker ... among other things. Thanks Mom! Also thanks to Duane Bibby for the wonderful drawings, Wendy McKay for organizing the Mac OS X session, Robin Laakso for her extraordinary organizational efforts, as well as generally being such a pleasure to work with on all things TUG, all the members of the conference committee, and especially Lance Carnes, for much work on the local arrangements, and for dreaming up the idea in the first place.

25 Years of TEX and METAFONT: Looking Back and Looking Forward
TUG 2003 Keynote Address

Nelson H. F. Beebe
University of Utah
Department of Mathematics, 110 LCB
155 S 1400 E RM 233
Salt Lake City, UT 84112-0090
USA
WWW URL: http://www.math.utah.edu/~beebe
Telephone: +1 801 581 5254
FAX: +1 801 581 4148
Internet: beebe@math.utah.edu, beebe@acm.org, beebe@computer.org

Abstract

TEX has lasted longer than many other computer software technologies.

This article reviews some of the history of TEX and METAFONT, how they have come to be used in practice, and what their impact has been on document markup, the Internet, and publishing.

TEX has several design deficiencies that limit its use and its audience. We look at what TEX did right, and with 25 years of hindsight, what it did wrong.

We close with some observations about the challenges ahead for electronic representation of documents.

Summary

1 Some historical highlights

2 What we've accomplished
- 2.1 Books and journals
- 2.2 Software archives
- 2.3 Document archives
- 2.4 Bibliography archives

3 What did TEX do right?
- 3.1 Open software
- 3.2 Typesetting kernel
- 3.3 Extensible typesetting language
- 3.4 Device-independent output
- 3.5 Font independence
- 3.6 Open font specification
- 3.7 Boxes, glue, and penalties
- 3.8 Compact markup for common cases
- 3.9 Nonsignificant spaces
- 3.10 Identical results on all systems
- 3.11 Dimension independence
- 3.12 Dynamic loading of files
- 3.13 Redefinition of character meaning
- 3.14 No system call
- 3.15 Last definition holds
- 3.16 \special command
- 3.17 Stability and reliability
- 3.18 Illustrations by Duane Bibby

4 What did TEX do wrong?
- 4.1 No rigorous grammar
- 4.2 Macro, not programming, language
- 4.3 Too much hard coded
- 4.4 Too many fixed-size objects
- 4.5 Too many global variables
- 4.6 Too little tracing
- 4.7 Name collision
- 4.8 Inadequate I/O
- 4.9 Character set limits
- 4.10 No input filters
- 4.11 No color state
- 4.12 No graphics
- 4.13 One page at a time
- 4.14 Multicolumn deficiency
- 4.15 Not general enough for all writing directions
- 4.16 No DVI output pipe
- 4.17 No sandbox
- 4.18 Uncaught arithmetic overflow
- 4.19 32-bit precision too limiting
- 4.20 No floating-point arithmetic
- 4.21 No conventional arithmetic expressions
- 4.22 No word and line boundary markers
- 4.23 No paper size

Editor's note: This important paper was inadvertently omitted from the regular TUG 2003 proceedings, *TUGboat* 24(1). Our apologies.

4.24 No absolute page positioning
4.25 No grid typesetting
4.26 No comments in DVI files
4.27 No rotated material

5 What did METAFONT do right?
5.1 Open software
5.2 Font-design kernel
5.3 Programming language
5.4 'Meta' fonts
5.5 Shaped pens
5.6 Open font formats

6 What did METAFONT do wrong?
6.1 Bitmap output premature
6.2 Pen shapes
6.3 Curve representations
6.4 Inadequate I/O
6.5 Font sizes
6.6 Inflexible character numbering
6.7 Not adopted by font designers

7 Future directions
7.1 XML directions
7.2 Unicode directions

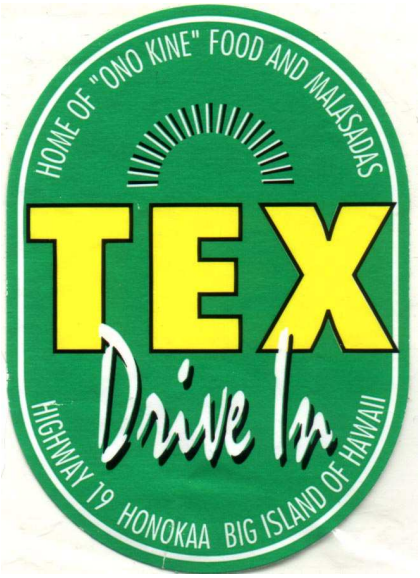


Figure 1: The TEX drive-in has two locations on the Big Island, one in Honokaa in the northeast, and one in Pahala, near the south center. It is noted for malasadas, a puffy hole-less donut, brought to Hawaii by Portuguese agricultural workers. One Web site reports that "Tex sells more malasadas than some McDonald's outlets sell hamburgers."

Foreword

Some of the material in this article may seem old hat to veteran TEX users, but I am writing it with the intention that it can also be read by people who are unfamiliar with TEX and METAFONT, but are nevertheless interested in learning something about the design and history of those programs.

Introduction

The TUG 2003 Conference at the Waikoloa Beach Marriott Hotel on the northwest coast of Hawaii (the Big Island) celebrated the 25th anniversary of TEX and METAFONT, and the 24th anniversary of the TEX Users Group. It was amusing to discover that TEX already had a commercial presence there: see Figure 1.

Donald Knuth enjoys finding numerical patterns, so I looked for some in connection with this meeting. The year 2003 contains the first two primes, and two zeros. Two is the base of most computer number systems, and 2003 is also the first prime in this millenium. In base 2, $2003 = 11\,111\,010\,011_2$ and $25 = 11\,001_2$: their five low-order bits are mirror images of each other. The number 25 is 5^2, or $(\text{third prime})^{(\text{oddest prime of all})}$.

1 Some historical highlights

Document production by humans goes back a rather long way, as shown in Tables 1 and 2. Although paper was invented about 6000 years ago, it was not until the middle of the 19th Century that wood pulp became the primary source of paper, and it took a few decades longer for paper to become widely available at low cost.

Gutenberg's invention predated Columbus' discovery of the Western World by just 40 years, and made large-scale book production practical. Before Gutenberg, each book was copied by hand; after Gutenberg, literacy was no longer restricted to a privileged class, and societal progress was poised for a huge leap forward.

It took about 30 years after the invention of digital computers for the first effective document formatting and typesetting systems to be developed. TEX and METAFONT were first implemented during Donald Knuth's 1977–78 sabbatical year. They were written in the SAIL[1] programming language, which was available only on DEC-10 and DEC-20 computers with PDP-10 CPUs. In 1978 or 1979, I had the good fortune to hear a talk that he gave at Xerox PARC about TEX, and I realized immediately that older document production systems, like IBM's ATS,

[1] Stanford Artificial Intelligence Lab/Language

Table 1: Notable historical events BC (before computers).

Year	Event
4000 BCE	Egyptians invent papyrus from woven reeds
105	Ts'ai Lun invents bark/hemp/rags-based paper in China
1009	First European paper mill, in Xativa, Spain
1411	First paper mill in Germany
1452	Johannes Gutenberg invents movable type
1680	First paper mill in New World, in Culhuacan, Mexico
1690	First paper mill in English colonies, near Philadelphia
1798	Nicholas Robert invents first paper-making machine, in France
1850–1879	Paper from wood pulp perfected
1889–1900	Economical mass-produced paper

DEC's `runoff`, and my own `document`, would be obsolete as soon as TEX were widely available. We had a DEC-20 at Utah, so we had early access to the Stanford software.

The excitement that TEX and METAFONT generated among Donald Knuth's colleagues, and at the American Mathematical Society, led to a redesign and reimplementation of both in Pascal, released in 1982, and tweaked a bit in 1989. At the time, the only other widely-implemented programming languages were Fortran and Cobol, neither particularly suitable for writing typesetting software. Nevertheless, there was at least one early implementation of the SAIL version of METAFONT in Fortran [48], with the goal of producing fonts for Burmese.

By the late 1980s, the C programming language was becoming widely available: it became an ISO Standard in 1989. While C has a number of drawbacks, it has many fewer limitations than Pascal. A successful manual translation of TEX from Pascal to C by Pat Monardo at the University of California, Berkeley, about 1990, encouraged a collaborative effort on the Web2C translation system. Web2C recognizes just the subset of Pascal used in TEX, METAFONT, and their associated utility programs, and translates it to C. Today, most implementations are based on the C translations, but the original Pascal source code remains definitive. System-dependent changes to the software are handled through change

Table 2: Notable historical events AC (after computers).

Year	Event
1940s	First digital computers
1968–1973	Niklaus Wirth invents Pascal language
1969–1970	Dennis Ritchie invents C language
1970s	`roff`, `script`, `runoff`, `document`
1975–1978	`eqn` (B. W. Kernighan and L. Cherry)
1976	`nroff` and `troff` (J. Ossanna),
1978	`bib` and `refer` (M. Lesk)
1977–1978	classic TEX and METAFONT in SAIL (D. Knuth)
1978–1980	SCRIBE (B. Reid)
1979	`tbl` (M. Lesk)
1981	`pic` (B. W. Kernighan)
1982	`ideal` (C. Van Wyk)
1982	'final' TEX and METAFONT in Pascal
1983–1985	LATEX (L. Lamport)
1984	BIBTEX (O. Patashnik)
1984	PostScript (Adobe Systems)
1986	`grap` (J. Bentley and B. W. Kernighan)
1989	'new' TEX and METAFONT (8-bit characters et al.)
1989–1991	HTML and HTTP at CERN (T. Berners-Lee)
1990	METAPOST (J. Hobby)
1991	World-Wide Web at CERN
1993	xmosaic browser (NCSA: M. Andreeson)
1993	PDF (Adobe Systems)
1994	LATEX 2_ε (F. Mittelbach et al.)
1994	Ω (Y. Haralambous and J. Plaice) and Λ
1995–2000	WeBWork (University of Rochester)
1996	PDFTEX (Hán Thế Thánh)
1997	ε-TEX (P. Breitenlohner et al.)
1998	$\mathcal{NTS}$ (K. Skoupý)
2000	XMLTEX (D. Carlisle)
2001	JadeTEX (S. Rahtz)
2002	Donald Knuth celebrates $1{,}000{,}000_2^{\text{th}}$ birthday
2003	ANT (ANT is not TEX: A. Blumensath) (OCaml: 24K lines)
2003	Nottingham font conversion project (D. Brailsford)

files that the tools `tangle` and `weave`, or their C equivalents, `ctangle` and `cweave`, turn into revised source code and documentation.

Karel Skoupý's $\mathcal{NTS}$ is a reimplementation of TEX in Java with the goal of improving modularization, and making possible experiments with new ideas in typesetting. Achim Blumensath's ANT (for ANT *is not TEX*) system has similar goals, but is done in the high-level OCaml language. Most of the coding of Ω, the extension of TEX for Unicode, is being done in C++, effectively a superset of C, and now about as widely available as C.

Although the Bell Laboratories' typesetting systems did not influence TEX very much, and were not available outside their development environment at the time that Knuth began his work on TEX and METAFONT in 1977, he was certainly aware of them [38, Chapter 2]. The Unix small-is-beautiful software-design methodology was similarly inapplicable on other operating systems, so TEX and METAFONT are each monolithic programs, of about 20,000 lines of prettyprinted Pascal code each. While relatively large programs at the time, they are dwarfed today by code projects that run to millions (e.g., GNU C compiler and library) and tens of millions (most modern operating systems) of lines.

What has set TEXware apart from most other software projects is the high degree of stability and reliability. Knuth attributes this to his development of *literate programming*[2] [55, 39, 37], which is so nicely illustrated in the TEX and METAFONT program source code [30, 32].

2 What we've accomplished

While TEX users are substantially outnumbered by users of desktop-publishing systems, TEX's open markup and document longevity, and its ability to handle mathematical and music markup, and scripts in many languages, and its possibility of identical output on all platforms from desktops to supercomputers, has ensured its continued use in some fields. In this section, we examine some of its achievements.

2.1 Books and journals

More than a hundred books have been published about TEX and METAFONT,[3] and many thousands of books, and numerous journals, have been published with TEX acting behind the scenes as the typesetting engine.

[2] `http://www.math.utah.edu/pub/tex/bib/index-table-l.html#litprog`.

[3] `http://www.math.utah.edu/pub/tex/bib/index-table-t.html#texbook3`.

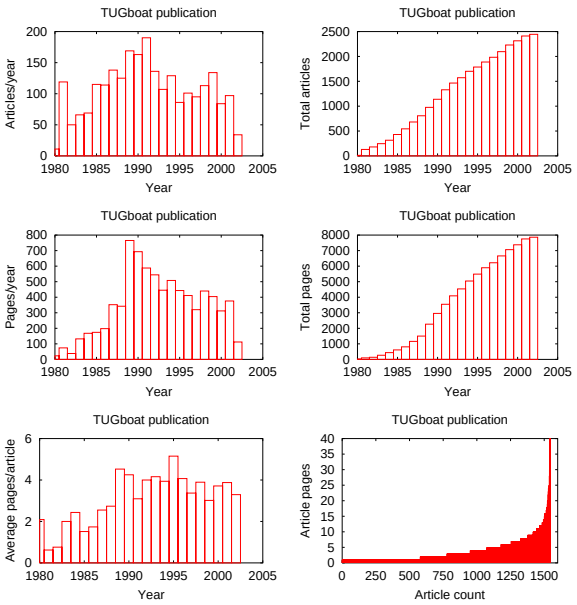


Figure 2: *TUGboat* publication statistics.

At least four journals, *TUGboat*, *Electronic Publishing—Origination, Dissemination, and Design*, *Markup Languages: Theory & Practice*, and *Serif*, have been devoted to typography, markup, and fonts; the bar charts in Figure 2 illustrate the activity in the first of these.

Today, many journals in computer science, mathematics, and physics use LATEX markup for author-submitted manuscripts. Sometimes, that material is converted by publishers into SGML or XML markup that TEX then typesets.

Several major publishers, including Addison-Wesley, Elsevier, Oxford, and Springer, have used TEX in book production.

Importantly for researchers, TEX markup has become a *de facto* standard in several technical fields, and because it requires nothing more than plain ASCII, it can even be used in e-mail messages.

2.2 Software archives

There are huge archives of TEXware in the CTAN (Comprehensive TEX Archive Network) collections, with three master hosts,[4] and about 75 mirror sites around the world. *TUGboat* issues have at least twice been accompanied by CD-ROM copies of the CTAN archives.

The existence of many mirror sites makes it hard to document CTAN activity, but the logs from just two of the master hosts record about 275,000

[4] `ftp://ftp.dante.de`, `ftp://ftp.tex.ac.uk`, and `ftp://tug.ctan.org`.

hits per week over the last seven years, from over a million Internet hosts.

In mid-2003, the CTAN archives contained nearly 80,000 files in about 6,000 directories, with close to 3,000,000 lines of class and style files. To put these numbers into perspective, Knuth's original `plain.tex` is only 1235 lines, and `manmac.tex`, which supplies additional macros needed for typesetting the TEXbook [29], is only 715 lines. Knuth's Computer Modern fonts are programmed in about 260 files, but the CTAN archives now hold nearly 6500 other METAFONT font programs. Wonderful, and skilled, volunteers did all of the rest of the work!

2.3 Document archives

The rapid spread of the Internet led Paul Ginsparg in the early 1990s to create the Los Alamos archive of current research in high-energy physics. This archive is now hosted at Cornell University,[5] and access statistics show that at times, the archive has had more than a million hits a week. For many physics researchers, the archive, not print journals, *is* the current literature. Ginsparg was awarded a prestigious MacArthur Fellowship (worth US$500,000) in 2002 for this work.

The success of the physics archive has led to creation of similar projects in mathematics, nonlinear sciences, computer science, and quantitative biology, all reachable from links from the main archive Web site. Related efforts include the Networked Computer Science Technical Reference Library (NCSTRL)[6] and the Open Archives Initiative.[7] Collectively, these archives contains several hundred thousand research papers, most written in LATEX or TEX markup.

2.4 Bibliography archives

In 1991, I began to record bibliographic data for my books and journals in BIBTEX markup. This evolved into the TEX Users Group Bibliography Project[8] covering the literature of much of computer science, electronic document production, and numerical mathematics. In 1995, we started the BibNet Project[9] with the more limited goal of recording complete publication bibliographies for leading researchers in numerical mathematics.

These collections now amount to more than 366,000 bibliographic entries in about 540 separate bibliographies, each of which is accompanied by additional files for spell checking, indexing, and typesetting the complete bibliographies. Because the data has been collected from many sources and extensively checked, it has considerably higher quality than most other collections. All BIBTEX files are prettyprinted, sorted, ordered, checksummed, and documented with a consistent comment header. Where possible, bibliographic entries contain hypertext links to the source of the data, and to online electronic documents.

In my department at Utah, the `bibsearch`[10] utility provides very fast access to these collections, plus another 203,000 in mathematical biology, 187,000 in computer science from the Digital Bibliography and Library Project (DBLP) at Universität Trier, Germany, and more than 1,261,000 in computer science from the world-wide computer science bibliography archive at Universität Karlsruhe. The two projects hosted at Utah are mirrored daily to the Karlsruhe archive. Archive statistics at Karlsruhe record about 300,000 hits per month, and at Utah, about 21,000 per month.

Both the American Mathematical Society MathSciNet database[11] and the European Mathematical Society E-Math database[12] now offer search results in BIBTEX markup.

The bibliography-archive work is supported by about 137,000 lines of code in the `awk` programming language, about 15,000 lines of code in Emacs Lisp, several thousand lines of Unix shell scripts, and several tens of thousands of lines of C code. Notable standalone tools in the collection include `bibcheck`, `bibclean`, `bibdup`, `bibextract`, `bibjoin`, `biblabel`, `biblex`, `biborder`, `bibparse`, `bibsearch`, `bibsort`, `bibsplit`, `bibunlex`, `citefind`, `citesub`, and `citetags`.

Many journal publishers now provide Web sites with publication data for recent issues. The JSTOR Project[13] provides complete coverage of all issues of about 360 journals in science and the humanities; it has allowed the preparation of complete bibliographies for the American Mathematical Monthly back to the first issue in 1894.

The increasing availability of publication data on the Web has made it feasible to develop tools

[5] `http://arxiv.org/`.
[6] `http://www.ncstrl.org/`.
[7] `http://www.openarchives.org/`.
[8] `http://www.math.utah.edu/pub/tex/bib/index-table.html`, `http://www.math.utah.edu/pub/tex/bib/idx/`, and `http://www.math.utah.edu/pub/tex/bib/toc/`.
[9] `http://www.math.utah.edu/pub/bibnet/`.
[10] `http://www.math.utah.edu/pub/mg/mg-1.3x/bibsearch/`.
[11] `http://www.ams.org/mathscinet/`.
[12] `http://www.emis.de/ZMATH/` and `http://zb.msri.org/ZMATH/`.
[13] `http://www.jstor.org/`.

for automatic conversion of such data, which tends to be quite similar across all journals from a single publisher. An essential tool for this work has been the HTML prettyprinter, `html-pretty`,[14] which allows fast, rigorous, and highly-reliable conversion of HTML documents to a standard layout of markup. Simpler tools, usually written in `awk`, can readily process that data to produce rough BibTeX markup that can be further cleaned up in a pipeline of some of the other tools listed above. A certain amount of manual cleanup is still required, notably bracing of proper nouns in document titles, but the bulk of the work has been done completely automatically.

Although the individual bibliographic and HTML tools are freely available, the Web page conversion software is not: it tends to require frequent maintenance as whimsy overtakes good sense at publisher Web sites, and in any event, is really needed only at the site doing the HTML-to-BibTeX conversion work.

I have collaborated with several publishers and journal editors, so that these BibTeX archives receive prominent links from their journal Web sites, and so that I get prompt notification of the appearance of new journal issues. In extremely favorable cases, the BibTeX data can be available about ten minutes after a journal issue announcement.

This bibliographic activity is a substantial effort on my part, but the reward is that the entire Internet community gets quick access to the data. Also, BibTeX, LaTeX, and TeX get free advertising in communities that might be entirely unaware of them, and I can at last find material in the thousands of journal issues on my own shelves.

BibTeX markup is extremely flexible, even if the associated style-file language is somewhat arcane. Publisher interest in XML markup led to the BibTeXML project at the Swiss Federal Institute of Technology (ETH) in Zürich, Switzerland, which has developed software for conversion between XML and BibTeX markup of bibliographic data; at the time of writing, the project Web site is not accessible. Because XML is not extensible at the document level, such a conversion is not as simple as it might first appear.

3 What did TeX do right?

Twenty-five years is a very long time in the rapidly-developing computing industry, and it is appropriate to look back over the use of TeX in that time, and comment on its successes, and its failures.

While a list of such points is necessarily a matter of personal opinion, I believe that it is worthwhile to enumerate the ones that I have found significant. Although I concentrate mainly on TeX, some of the remarks should be interpreted to include the associated utilities that I call TeXware.

3.1 Open software

The most important done-right feature of TeX is that it is an *open-source literate program*. Without quibbling over the exact meaning of open source, the essential point is that *anyone* can use TeX for any purpose, commercial or noncommercial, as long as they don't change the non-system-dependent parts of it without changing its name. One of the TUG 2003 conference speakers, Ajit Ranade, noted that TeX is responsible for a multimillion dollar typesetting-support industry in India that employs thousands of people.

This openness should be contrasted with the abysmal state of the desktop-publishing industry where markup is generally keep secret and proprietary, holding user documents hostage to marketing whims of software vendors, and making it impossible to achieve consistent output when documents are moved between platforms, or to archive documents for long-term access. This deplorable situation is getting worse, not better: one desktop-publishing vendor is moving towards encrypted file formats that can be decoded only by that vendor's products; the marketing justification is called *document security*, but it also locks out competition and gives the vendor, not the user, control over document access.

3.2 Typesetting kernel

TeX provides a small kernel of primitives that are specialized for the complex, and often idiosyncratic, job of typesetting. The best-known analogue of this software model is the PostScript page-description language, which provides primitives that are highly suited to placing marks on a page, assuming that some other software package has first decided where they go.

3.3 Extensible typesetting language

Although most of TeX's typesetting kernel is rather low level, dealing with object positioning and selection, through a powerful, albeit arcane, macro language, TeX can be extended to provide higher-level markup. Such extensions significantly lessen the demand for changes in the underlying software, shielding it from the creeping featurism that plagues most commercial software, and prevents reliability and stability.

The most successful of these extensions is the

[14] `http://www.math.utah.edu/pub/sgml/`.

LATEX document preparation system [41, 42] and the many packages built on top of it [47, 18, 19]. LATEX, like BIBTEX, is strongly influenced by Brian Reid's pioneering SCRIBE system.

The key feature of LATEX markup is that typesetting objects such as title and author information, tables of contents, abstracts, chapters, sections, subsections, equations, figures, tables, glossaries, indexes, and so on, are general concepts shared by most documents, and are thus marked up with commands that tell *what to do*, rather than *how to do*. This is usually termed *logical markup*, as opposed to *physical*, or *visual, markup*. Most desktop-publishing software is modeled on physical markup: the what-you-see-is-all-you've-got approach is burdensome on the user, and makes it nearly impossible to achieve formatting consistency.

In principle, the meaning of logical markup can be changed simply by declaring a new document class at the start of the top-level file. In practice, other changes may be required as well, but they are usually limited to issues of front-matter markup, which can be quite complex for some documents, and to issues of how literature citations are used in the text (e.g., the numbered style used in this article, versus the author-date style common in the humanities). Nevertheless, the changes needed to switch document style are generally a small part of the total effort of document production.

The need for presentation of the same information in different formats is often not appreciated by authors, but publishers at past TEX Users Group conferences have reported that they have sometimes been able to reuse information in a dozen ways, each of them generating income. Document reuse is a major driving force in the use of SGML and XML markup for long-term document storage in the publishing industry. For example, while most article submissions to the American Chemical Society (the world's largest publisher of chemistry literature) are in word-processor formats, all submissions are converted to SGML with a combination of in-house format-conversion software and manual touch-ups.

3.4 Device-independent output

The Bell Laboratories' `troff` system produced output for one particular typesetting device, now long defunct, and its design was heavily influenced by the capabilities and limitations of that particular device.

TEX's output pays homage to no particular device: it is a compact device-independent format, called a *DVI* file. The job of converting that file for any particular output device is left to separate software, called a DVI driver. This task is far from trivial: current drivers for single devices are 22,000 (`dvips`) to 29,000 (`xdvi`) lines of code, and my own DVI driver family, which supports dozens of devices, is about 97,000 lines. These are all larger than TEX itself. Popular output formats have evolved during TEX's life, but TEX remains blissfully independent of that evolution.

Current analogues of TEX's design choice are the virtual machine definition underlying the Java and C# programming languages, and the virtual machine layer provided on IBM mainframes since the early 1970s. More recently, the VMware[15] system on the Intel IA-32 platform permits multiple operating systems to be simultaneously active on the same hardware, and IBM PowerPC systems now support sharing of CPUs by separate operating systems.

Although they were not part of the original design of TEX, Geoffrey Tobin's excellent `dv2dt` and `dt2dv` tools included in many modern TEX distributions provide a way to convert the compact DVI format to a human-readable, and thus, editable, form, and back again.

3.5 Font independence

The lack of good-quality vendor-independent fonts forced Donald Knuth to develop an outstanding font-design system, METAFONT. However, he was careful to isolate most of the details, so that TEX only needs to know about the *dimensions* of the fonts, through the compact *TEX font metric* (TFM) files. TEX remains completely unaware of character shapes and how they are represented in font files.

This independence has proved a great virtue, allowing TEX to use almost any font, subject to an unfortunate limitation on the number of characters in a font, provided that metrics are available. Some commercial font vendors in the early days of TEX would not release that information, and some still do not permit its free distribution.

With virtual-font technology [26], composite fonts can be created whose glyphs come from other fonts, even other virtual fonts. This makes it possible, for example, to remap glyphs into the order expected inside TEX, avoiding the need to redefine numerous macros that assign names to font characters. PostScript Type 1 fonts are generally accessed via virtual fonts.

3.6 Open font specification

Unlike most commercial fonts of the 1970s, METAFONT's output file formats are openly, and well, documented. Besides the TFM file of font metrics,

[15] `http://www.vmware.com/`.

METAFONT produces a file of character bitmaps, called a *generic font* (GF) file.

When superior font compression techniques were developed by Tom Rokicki in 1985 [53], he was able to write a clean font-format translation tool, `gftopk`. Today, most TEX sites store only the more compact PK format.

3.7 Boxes, glue, and penalties

A great insight in the design of TEX is the concept of boxes of text, and flexible space between them, called *glue* (though *springs* might have been a better characterization). TEX builds up lines, paragraphs, and page galleys as lists of boxes separated by glue, where the glue has user-defined stretchability and shrinkability.

Centered, ragged-right, and ragged-left typesetting are all straightforward to implement with glue.

For further control, users can insert penalties at suitable points to encourage or discourage line breaks and page breaks. TEX can then apply a mathematical optimization technique to determine the best way to typeset pages; few other systems, before or since, do as good a job.

3.8 Compact markup for common cases

The first thing that one sees when comparing SGML or XML markup with TEX markup is that the first two are painfully verbose. TEX makes common cases simple and compact. For example, instead of wrapping paragraphs with SGML commands `<paragraph>` ... `</paragraph>`, TEX simply assumes that a line that is blank or empty separates paragraphs. When this is not convenient or possible, it offers the `\par` control word to mark the boundary.

Other examples of TEX's brevity are the use of braces for grouping, dollar signs around mathematics markup (Knuthian humor: mathematics was traditionally expensive to typeset), and caret and underscore for mathematical superscripts and subscripts.

Consider the lowly, but important, period (dot or full stop): it is the smallest character in a font, but it can mean end of initial, end of abbreviation, end of sentence, decimal point, Internet hostname separator, filename separator, or be part of an ellipsis. TEX has a simple rule for its interpretation when it is followed by an input space: if it follows a capital letter, it ends an initial, and otherwise, it ends a sentence. This heuristic is almost always correct, but careful LATEX typists will write `Back in the USSR\@.` to instruct TEX that an intersentence space, rather than an interword space, is called for, or use a tie (`~`) or literal space (`\␣`) when only an interword space is needed. The special spacing required for ellipses is handled by standard control words: `\cdots`, `\ldots`, and `\ddots`.

3.9 Nonsignificant spaces

In `troff`, spaces are significant: two spaces in the input produces two spaces in the output, plus or minus a bit to justify lines. Consequently, indentation cannot be used in `troff` input to improve readability and highlight nested document structure.

TEX avoids this significant design flaw by treating a sequence of spaces as equivalent to a single space. Similarly, any sequence of empty or blank lines is equivalent to a single paragraph break. When this behavior is unwanted, TEX offers verbatim environments.

3.10 Identical results on all systems

A huge virtue of TEX is the possibility of getting identical output on all platforms. Indeed, as long as all of the input macro packages and font metrics are identical, output *is* identical. No commercial desktop-publishing system even comes close.

TEX achieves this platform-independence by carrying out all arithmetic identically: it uses exact fixed-point arithmetic in all computations that affect line and page breaking. For dimensions, an underlying 32-bit integer word is split into fragments $1 + 1 + 14 + 16$: a sign bit, an overflow bit, a 14-bit integer part ($2^{14} = 16384$), and a 16-bit fractional part. The largest dimension is about the width of a room, and the smallest spacing is less than the wavelength of visible light: computational rounding errors are invisible in the output. TEX also supports pure 32-bit integer arithmetic in computations with `\count` registers.

3.11 Dimension independence

Systems of measurement differ between different cultures and countries: TEX allows dimensions to be specified in any of nine different units that cater to the majority of needs.

3.12 Dynamic loading of files

TEX permits temporary redirection of its input stream to another file, via the `\input` control word. That file can in turn contain other `\input` commands, with the result that packages of commonly-used commands are easily supported, and users can break up large documents into manageable pieces. This saves processing time, since only those pieces being worked on need to be typeset, and also reduces the effect of global editing disasters.

3.13 Redefinition of character meaning

TEX assigns each input character one of sixteen category codes that controls further interpretation of each character, and those assignments can be changed at any time via the `\catcode` command. Although few users directly exploit the feature, its existence makes it possible for TEX to typeset documents written in quite different markup, such as control words that begin with `@` in Texinfo, and angle-bracket delimited words in SGML and XML (e.g., `jadetex` and `xmltex`).

3.14 No system call

Because TEX is expected to run on many different platforms, it does not offer any internal way of communicating with the underlying operating system, such as via a `\system{...}` command.

While such an ability can be handy, allowing, for example, a book index to be prepared immediately before it is typeset, experience on the Internet has shown that such power is too frequently exploited for nefarious purposes. The words *worm* and *virus* are now familiar to the general population, even those who have never used a computer. When the mere viewing of a document can cause arbitrary commands to be executed, security and stability are impossible.

3.15 Last definition holds

Although it seems trivial, in TEX, as in most programming languages, the last definition or assignment of an object is the one that is used.

SGML, by contrast, uses a first-definition-holds rule. Instead of being able to load up a base package of command definitions, and then make minor tweaks to it by subsequent redefinitions and assignments, each package modification must be defined as a complete new package, with altered definitions preceding an inclusion of the original package.

3.16 `\special` command

In order to allow support of unforeseen features, TEX provides the `\special` command whose argument is recorded verbatim in the DVI file. Graphics, color, and hypertext links are common examples that use `\special`. Of course, DVI drivers then need to be taught how to handle such material.

3.17 Stability and reliability

TEX is quite possibly the most stable and reliable software product of any substantial complexity that has ever been written by a human programmer. Although its development has not been free of errors, most TEX users have never seen an error, much less a crash, in TEX.

What errors have been found have been well studied and documented in *The errors of TEX*: see [35] and an update in [37, pp. 243–339].

3.18 Illustrations by Duane Bibby[16]

In a remarkable collaboration that has lasted nearly two decades, the gifted illustrator Duane Bibby has worked with members of the TEX community to prepare wonderful drawings for not only Knuth's *Computers and Typesetting* series, but also for several LATEX books, and numerous T-shirts and mugs at TEX Users Group conferences.

It was such a pleasure for many of us to meet Duane for the first time at this meeting, to hear his talk on his long collaboration with Donald Knuth, and to see how the TEX lion and METAFONT lioness evolved.

Duane Bibby's drawings, and Donald Knuth's wit and superb writing skill, add light and humor to what might otherwise be a dry and daunting manual on the complex subject of typography.

4 What did TEX do wrong?

It is now time to drop our laudatory stance, and become a grinch: nothing is perfect, not even TEX, and with 25 years of hindsight, it is time to assess what it did wrong.

There has been a lot of computer technology developed since 1977 that few could have predicted then, including personal computers, high-quality printers, PostScript, PDF, window systems, and the World-Wide Web. Hardware costs have dropped, and speed and capacity have grown, at a pace that is unparalleled in the history of human development. Most of the criticisms of this section would have been both unfair and unforeseen when TEX was first designed.

4.1 No rigorous grammar

The biggest deficiency in TEX that has always bothered me is that it is not based on a rigorous programming-language grammar. This is particularly puzzling when its author is the founder of modern LR parsing [28], and that work is cited among the great papers in computer science [43]. When I asked Don about this once, he jokingly responded that he didn't believe in grammars!

Most programming languages designed since

[16] This section of my address was written long before I found out that Duane would be attending the conference.

Algol 60 have been based on grammars, but TEX is not. Lack of an independent grammar means that the only truly reliable parser of TEX input is TEX itself, yet long experience with other programming languages has shown that rigorous grammars can lead to highly-reliable machine-generated parsers (Unix `yacc`, Berkeley `byacc`, GNU `bison`, and Holub's `occs` [24] are good examples of parser generators). Parsers are needed not only in compilers, like TEX, but also in prettyprinters, syntax checkers, tag extractors (`ctags` and `etags`), and other tools that operate on the input language for purposes other than its execution.

Precisely because human programmers are fallible, it is important to have multiple independent implementations of any significant program, and to have implementations available on several machine architectures. Only when those programs produce identical output can one have any confidence in their results. Languages like `awk`, Cobol, Fortran, C, C++, Common Lisp, and Java enjoy this diversity, while others, like `axiom`, C#, `delphi`, `maple`, `mathematica`, `perl`, `python`, `ruby`, `reduce`, `sas`, `spss`, `tcl`, and Visual Basic, do not. Since modern compilers are usually very much bigger than the programs that they process, when a bug or other unexpected behavior surfaces, it is legitimate to ask whether the bug is in the compiler, or in the program. If the misbehavior disappears when the compiler or platform changes, suspicion rests on the compiler.

The lack of diversity for TEX has been less of a problem, simply because of the enormous talent behind it. Nevertheless, it is good to see the appearance of $\mathcal{NTS}$ as an independent implementation of TEX.

4.2 Macro, not programming, language

TEX's extension language is a macro language, not a proper programming language. Every TEX macro programmer who has implemented a nontrivial operation has suffered from the difficulty of TEX's macro language. Things that are simple to do in other languages designed for text processing are frequently very painful in TEX. Macro languages have too many side effects, and lack the power of true programming languages.

Lisp programmers would argue that the proper approach is to eliminate that separation between programs and data: making data look like programs means that the data can *be* a program, and that has been found to produce great power and generality. Luigi Semenzato and Edward Wang at the University of California, Berkeley, investigated a Lisp-like interface to TEX [54].

4.3 Too much hard coded

The limited 18-bit address space of the 36-bit DEC PDP-10 architecture[17] of TEX's original development platform, and severe limitations of the Pascal programming language, constrained many aspects of the program.

There are many examples of objects inside TEX whose size is fixed when TEX is compiled for a particular platform: the dreaded `TeX capacity exceeded` message is familiar to every serious TEX user.

Fortunately, in recent years, the Web2C implementation has replaced many of these fixed limits by configurable limits settable in startup files, although other implementations may not be so flexible. Nevertheless, modern programming practice in the GNU system and others is that software should have no hard limits, other than those found to be available at run time, or enforced by the underlying architecture.

It isn't just table sizes that are hard coded in TEX: many algorithms are too, notably, hyphenation, line breaking, page breaking, and float placement. While TEX provides powerful ways to influence these algorithms, the algorithms cannot be ripped out and replaced dynamically at run-time, and they cannot be changed in TEX itself without producing a program that is no longer allowed to call itself TEX.

4.4 Too many fixed-size objects

The need to squeeze a large program and data into the small PDP-10 address space led to further economizations in TEX that are difficult to remove, because the sizes of various objects themselves are encoded in bitfields of other data structures. If only 8 bits are available for the size, then only $2^8 = 256$ objects can be created. This limitation is seen in the number of various types of TEX boxes, category codes, token lists, registers, and skips. Worse, it permeates TEX's internal source code, making it very hard to eliminate.

Enlarging these limits was one of the major design goals of ε-TEX, which is extended from TEX with a change file that is about a third the size of TEX itself. For comparison, pdfTEX augments TEX's DVI output with a completely new, and very complex, output form: PDF; its change file is about 40% the size of TEX.

[17] In modern terms, 1.3M characters or 0.25M words.

4.5 Too many global variables

In the 60-year history of computing, program after program has foundered in complexity arising from too much global state. When code depends on global variables that can be changed arbitrarily at any place in the code, it becomes impossible to reason about the correctness of the program. Redesigns of the relatively small and simple Unix operating system in the form of Mach, Solaris, and GNU/Linux all attempted to sharply limit the problem of global state, by repackaging the software into independent layers with simple, and well-defined, interfaces.

TEX has too many global variables and macros that can be changed anywhere, at any time, by any user.

4.6 Too little tracing

For debugging, and for understanding unfamiliar packages, it is desirable to be able to request tracing of the use and (re)definition of commands, files, and registers. Although TEX provides a few tracing commands, their output usually overwhelms the user, because there is no way to restrict the tracing to a specified set of names. Also, TEX provides no way to record *where* definitions happen, yet precisely that information is needed to prepare customizations.

4.7 Name collision

Older programming languages, such as Algol, Fortran, Cobol, Pascal, and C all have very limited control over name visibility. Usually, this amounts to just *local* variables (defined in a function or subroutine), and *global* variables, known throughout the program. C adds a bit more control in offering *file-global* variables that are known throughout a single source file, but inaccessible elsewhere.

Descendants of these languages, such as Ada, Fortran 90, Cobol 2002, Modula, C++, Java, and C# all introduce additional constraints through modules or namespaces to allow compartmentalization and control of names.

Sadly, TEX has effectively only one level of naming: global. Although names can be defined inside braced groups, the size of groups can be severely constrained by internal buffer sizes, and in any event, groups are anonymous: you cannot refer to names in other groups except by nasty or tricky subterfuges that make program maintenance impossible.

Lisp too had this defect, but it became common practice to use long descriptive names that incorporate a package prefix, such as `LaTeX-pageref-with-completion` from my LATEX editing-support package for Emacs. However, with default category codes, TEX limits command names to just the Latin letters, so the only way to avoid inadvertent collisions with names in other packages is to use long names, and the only way to make them readable is to use mixed capitalization, such as (in the LATEX kernel) `\DeclareRobustCommand`.

The lack of hooks into the entry and exit of commands means that packages are often forced to redefine macros used by other packages. Loading of multiple packages sometimes results in puzzling, and hard-to-debug, interactions of these redefinitions.

4.8 Inadequate I/O

Like Fortran, TEX's I/O model is based on lines, rather than characters, with the additional restriction that braces (or characters of that category) must be properly nested. By contrast, the C programming language provides `getc()` and `putc()` primitives to read and write single characters, and I/O of higher-level objects can be built from these alone.

Java, C++, and C# go a step beyond C in generalizing I/O to a stream of data in which processing filters can be arbitrarily, and transparently, inserted. The streams need not even be directed to and from physical files, but can instead refer to strings in memory, or network devices, or display devices, or virtual files.

Unlike most other programming languages, TEX does not offer anything analogous to Fortran's formatted I/O or C's equivalent of `fscanf()` and `fprintf()`, which provide detailed control over the interpretation of input, and the appearance of output. A satisfactory I/O library is astonishingly complex: it deserves to be rigorously defined, standardized, and incorporated in every implementation of the programming language.

The deceptively-easy task of expressing binary fractional numbers as human-readable decimal numbers led TEX's author to write a famous paper called *A Simple Program Whose Proof Isn't* [36]. Related articles that finally properly solved the number-base conversion problem have appeared only since 1990 [1, 7, 21, 56, 57].

4.9 Character set limits

TEX processing is based entirely on a model of characters that can be represented as a single 8-bit byte, in the ASCII encoding. This was not much different from other software of the 1970s, and at least, TEX could handle lowercase letters, and with the help of control symbols and control words, even

decorate letters with accents.

However, it was clear that $2^8 = 256$ characters are not enough, not even for European use. ISO has already standardized ten different ISO-8859-*n code pages* based on the Latin alphabet, with a few more to support Arabic, Cyrillic, Hebrew, and Thai.

Two separate efforts to standardize much larger character sets began in the early 1990s: Unicode and ISO 10646. After some divergence, the two groups are fortunately now coordinated, with Unicode guaranteed to be a strict subset of ISO 10646. Several operating systems have adopted Unicode as their standard character encoding.

Unicode developers hold that 21 bits are sufficient to encode all real writing systems in human history. This is a repertoire of about two million characters, of which 96,283 are encoded in the latest version [59]. Of these, 70,203 are ideographic characters from Chinese, Japanese, Korean, Yi, and historical Vietnamese [44, 45].

The large number of Unicode characters means that neither 8-bit nor 16-bit values suffice. 24-bit words are impractical in current computer architectures, and native 32-bit words waste a third of the bits. In practice, then, Unicode is represented in one of several encodings that require one or more 8-bit or 16-bit chunks to represent a single character. One of these, UTF-8, contains ASCII and most ISO-8859-1 characters in their normal positions, but uses up to four bytes for some other characters. UTF-8 was developed at Bell Laboratories for the Plan 9 operating system, and because most of the world's existing computer files are in either ASCII or ISO-8859-1, they are already Unicode-conformant when interpreted in the UTF-8 encoding.

Existing programming languages have usually been defined with the assumption that a single character can be held in an 8-bit byte, and the effects of that assumption are deeply rooted. Most languages are only beginning to address the problem of how to deal with Unicode data, and none of the initial attempts, in C, C++, C#, and Java, are yet very satisfactory.

TEX too requires massive changes to handle Unicode, and although the work on Ω was originally based on a change file for TEX, its developers report that a completely new program, in C++, will probably be needed.

4.10 No input filters

The code-page morass mentioned in the previous subsection has an immediate impact on TEX documents written in a variety of European languages. In particular, the encoding must be known, and then translated to TEX's internal expectations, before TEX can process the document. Filesystems rarely record character-set information, so documents have to do so themselves.

TEX processing would be easier if it had input filters that could transparently supply the needed translations. It would have been relatively easy to implement them by making the internal `xchr[]` array [30, §21, p. 10] accessible to the TEX user.[18] Instead, TEX erroneously assumes that character sets are a unique property of each platform, and therefore, can be hard-coded into the implementation on each system. That assumption was correct for EBCDIC on IBM mainframes versus ASCII everywhere else, but it was false as soon as code pages were introduced.

The need for input translation is so strong that Ω developers from an early stage in its design introduced Ω *translation processes* (OTPs).

4.11 No color state

In 1977, there were no low-cost color output devices, and some commercial typesetting systems were incapable of handling color. Consequently, TEX is completely ignorant of color. However, color is a text attribute much like the current font, although it can also be a page-background or region-shading attribute. In particular, once set, the current colors should remain in effect across line breaks and page breaks, just like the current font does.

In order to make it possible to process selected pages of DVI files, TEX records in the DVI postamble a list of all required fonts, and at the start of each page description in the DVI file, it records the current font, so that the page can be rendered independently of all preceding pages.

In the same way, the current colors are needed at page start. Since TEX doesn't provide for color, its support has to come through the `\special` command. However, TEX lacks a hook (code fragment) to be executed when a page is shipped out to the DVI file (actually, it does have one, but output routines are very fragile, and depend on the macro package in use), so there is no clean way to record the current colors on each page. This means that DVI drivers that support color are now forced to read the entire DVI file, parse all of its `\special` commands, and build up a list of starting colors for each page, even if the user just wants to display a single page. Fortunately, modern machines are fast, so most users probably never notice the delay.

[18] Some implementations have extended TEX to provide such access.

4.12 No graphics

In the 1970s, commercial typesetters had no graphics capabilities. Although researchers in computer graphics had produced a rudimentary draft graphics standard [2] a few months before work began on TEX, it took two more years for a more detailed draft standard [3], and much of that specification could not be implemented portably. Sadly, graphics standardization splintered and foundered in the 1980s, and today, the ISO graphics standards that exist are largely ignored by programmers.

Nevertheless, a small number of graphics operations, notably dot and line primitives, an elliptical arc primitive, and a region fill, would have been sufficient to represent most technical drawings. They could have been compactly encoded in the DVI file, and reasonably easily translated by DVI drivers.

LATEX's `picture` mode can emulate curves by drawing tiny squares along the curve path, but their internal representation is extremely inefficient, and even a few such curves soon exhaust TEX's memory.

Today, most graphics in TEX documents is represented by PostScript (released in 1984) figure inclusions.

4.13 One page at a time

Memory constraints forced TEX to handle page creation by building up a sequence of objects on the main vertical list, asynchronously invoking an output routine as the list contents get big enough to fill a single output page.

However, in high-quality manual typesetting, page-layout design proceeds in pairs of facing pages. With a lot of work, it would be possible to program a TEX output routine for handling pages in pairs, but the task is decidedly nontrivial, and in LATEX, the output routine is viewed as a fragile part of the kernel that cannot be altered without great risk of breaking other parts of LATEX.

4.14 Multicolumn deficiency

Multicolumn output is challenging, for three main reasons:

1. Narrow columns make it difficult to maintain right-margin justification without excessive hyphenation, or objectionable whitespace. Every reader of a printed newspaper is familiar with these problems.
2. Esthetic considerations require columns to be balanced, perhaps all with the same length, or with all but the last of uniform size.
3. Complex documents may require changing column formatting within a single page. In these proceedings, for example, article front matter is in one-column mode, but article text is in two-column mode. Some American Physical Society journals have particularly complex column-formatting practices, and newspaper layout is even more difficult.

Although it has been possible to prepare style files to support multicolumn typesetting, such as Frank Mittelbach's `multicol` package [46] and David Carlisle's and Arthur Ogawa's REVTEX package [50] designed for the American Physical Society, and also style files to support the flowing of text around figures [17, §6.4], only a handful of TEX wizards are capable of creating such packages. I suspect that all of these packages can be broken in certain cases: while they may be quite good, they are not robust.

Had TEX been designed from the beginning to typeset into a list of regions of arbitrary user-specifiable shapes, instead of into a single rectangular page, even the most complex magazine and newspaper layouts could readily be accommodated.

As the doctoral work of Michael Plass [40, 51] and Stefan Wohlfeil [61] has shown, line-breaking and page-breaking algorithms are extremely difficult. In my view, they should be implemented in a dynamically-loaded module in the programming language that TEX doesn't have.

4.15 Not general enough for all writing directions

TEX expects text to be laid out horizontally from left to right. While this works for many languages, it doesn't handle right-to-left Semitic languages, or vertical writing directions (left to right across the page, or the reverse) needed for Chinese, Japanese, Korean, and Mongolian.

Donald Knuth and Pierre MacKay were able to extend the TEX program to handle mixtures of left-to-right and right-to-left text [27], producing the TEX--XET derivative, and ε-TEX now includes that extension. In Semitic languages, numbers are written from left to right, so those languages always need bidirectional typesetting. Similar extensions have allowed TEX derivatives to handle some of the vertical typesetting needs in East Asia, although I suspect that Mongolian remains unsupported.

In the `troff` world, Becker and Berry [5] were able to extend that program for tri-directional typesetting, but could not handle a fourth writing direction.

In our global community, a typesetting system must be able to handle all traditional writing directions, including mixtures of all of them in a single

document (e.g., a Chinese literary article citing English, Arabic, and Mongolian texts).

4.16 No DVI output pipe

TEX expects to write an entire DVI file under its exclusive control, relinquishing it to others only when TEX is done. Unix pipelines demonstrate that it is very useful to sample program output before it is complete, but Unix was only just emerging from Bell Laboratories when TEX design began.

In the early 1980s, in unpublished work demonstrated at early TEX Users Group conferences, David Fuchs at Stanford University ported TEX to the cramped and crippled environment of IBM PC DOS, and then in a tremendous feat, extended TEX to allow immediate communication with a screen previewer. His work became a commercial product, MicroTEX, but its vendor shortly thereafter withdrew it from the market.

Later, Blue Sky Research produced Lightning Textures on the Apple Macintosh, which retypesets the document as changes are made in its input files.

More recently, Jonathan Fine in Cambridge, UK, developed `texd` [12], a permanently-running resident daemon TEX that can be called upon at any time to typeset a fragment of TEX code, and return DVI code.

Certainly, TEX's descendants should learn from these extensions.

4.17 No sandbox

When TEX was born in 1977, the Arpanet was eight years old, but still an infant, with fewer than a hundred hosts and a few thousand users with a strong community spirit. Today, the Internet has hundreds of millions of hosts, and is a decidedly hostile environment.

TEX does not offer a way of restricting its actions to benevolent ones. Although we noted earlier that TEX cannot run external programs, it can nevertheless (over)write a file anywhere in the file system that the user has write access to.

PostScript also has this problem, but the designers of the `ghostscript` implementation of that language added the `SAFER` option to limit filesystem access.

The TEX Live implementation of TEX contains changes to restrict output to the current directory, and prevent writing special configuration files that might open security holes, but most other implementations lack these features.

4.18 Uncaught arithmetic overflow

To enhance reliability, TEX catches arithmetic overflow from multiplication, but curiously, not from addition [30, §104], which you can readily demonstrate with this small example:

```
\count0 = 2147483647
\advance \count0 by \count0
\message{ADD: count0 = \the \count0}

\count0 = 2147483647
\multiply \count0 by 2
\message{MULTIPLY: count0 = \the \count0}

\bye
```

TEX reports in the output log:

```
ADD: count0 = -2
! Arithmetic overflow.
l.8 \multiply \count0 by 2
MULTIPLY: count0 = 2147483647
```

When I asked Don about this, he responded that there were too many places in TEX where integer addition was done. In my view, that is simply a design flaw: all of those additions should be done in one function that is called wherever needed. It is much better to get the right answer a little slower, than to get the wrong answer fast.

TEX is not alone in partly, or wholly, ignoring integer overflow: many CPU architectures, operating systems, and programming languages do too. Several major civilian, government, and military disasters have subsequently been attributed to arithmetic overflow [49].

4.19 32-bit precision too limiting

Although TEX's design choice of using fixed-point arithmetic was critical in achieving its goal of identical results everywhere, there are applications where 32 bits are insufficient. One of them is the implementation of trigonometric functions needed for computing text rotations, and another is fixed-point division.

4.20 No floating-point arithmetic

When TEX was designed, floating-point arithmetic systems varied widely, with 24-bit, 32-bit, 36-bit, 48-bit, 60-bit, and 64-bit sizes on various platforms. Worse, their implementations were sometimes seriously flawed, with anomalies like $(z + z) \neq 2 \times z$, $z \neq 1 \times z$, $y \times z \neq z \times y$, and **if** $(z \neq 0.0)$ $x = y/z$ terminating with a zero-divide error. It would thus have been untenable for TEX to use native hardware floating-point arithmetic for calculations that affect output appearance.

Starting in the late 1970s, a much-improved floating-point arithmetic system was designed by

an IEEE working group. Although the system was not standardized until 1985 [25], it was first implemented in the Intel 8087 already in 1980. This IEEE 754 floating-point arithmetic system has been part of almost every new computer architecture designed since then.

Despite standardization, variations in internal details of specific hardware implementations of IEEE 754 prevent getting identical results everywhere. However, but for TEX's seniority, TEX could have had its own *software* implementation of IEEE 754 arithmetic that behaved identically everywhere, and it could have had its own repertoire of elementary functions (trigonometric, logarithmic, exponential, and so on) that would have greatly simplified some typesetting applications.

4.21 No conventional arithmetic expressions

TEX's commands for integer arithmetic, illustrated in the overflow example given earlier, are inconvenient: it would have been much better to have a conventional arithmetic-expression facility. It is possible, though difficult, to do so in TEX's macro language [20]. The LATEX `calc` package only provides arithmetic expressions in a limited context.

Expression parsing of one of the first examples given in books on compiler design, and is not particularly difficult to implement; it should have been in TEX from the beginning.

4.22 No word and line boundary markers

With any document formatting or markup system or typesetting system, one sometimes needs to extract text from the formatted output, perhaps because the input is not available, or cannot be deduced from the input without implementing a complete parser.

While input usually reflects output, this need not be the case, as shown by David Carlisle's seasonal puzzle [8]. His plain TEX input file from the CTAN archives[19] is reproduced in Figure 3, and has no apparent relation to the typeset output, a well-known poem.

Unix tools such as `antiword`, `dehtml`, `deroff`, `detex`, `dexml`, `dvi2text`, `dvi2tty`, `pdftotext`, `ps-2ascii`, and `pstotext` attempt to do this text extraction for common document formats, but none is entirely successful, because they all have to apply fragile heuristics to deduce word and line boundaries from spacing.

Becker and Berry [5, p. 134] pointed out that they could not apply their method for tridirectional text to TEX because of its lack of line-boundary markers.

When text is typeset, TEX usually knows where each word begins and ends, and where it has used hyphenation for improved line breaking; sadly, this important information is lost in the DVI output.

Given the complexity of deciphering TEX input, checks for spelling, doubled-word, and delimiter-balance errors really should be done on text extracted from the DVI file.

4.23 No paper size

TEX's view of a page is a galley of material of unknown width and indefinite height, measured in a left-handed coordinate system beginning at an offset from the top-left corner of the page. The main body of each page is expected to fill a rectangle of size `\hsize` × `\vsize` (in LATEX, `\textwidth` × `\textheight`). The corner offset of its $(0, 0)$ point is, regrettably, a parochial one inch from each edge.

TEX never admits that there might be a physical page of one of several common standard sizes and names (e.g., A, A4, quarto, foolscap, JIS-B5, ...), and consequently, users who want to adjust their text dimensions to suit a particular paper size may have to tweak several different dimension parameters.

4.24 No absolute page positioning

Because of the developing page galley, within the input document, there is no way to refer to an absolute position on the output page, such as for typesetting a company logo, or displaying a red-lettered document-security classification. The only way that this can be achieved is to hook into the fragile output routine, most safely by attaching the desired material to running headers. Even that is not foolproof, because those headers might be

```
\let~\catcode~‘76~‘A13~‘F1~‘j00~‘P2jdefA71F~‘7113jdefPALLF
PA’’FwPA;;FPAZZFLaLPA//71F71iPAHHFLPAzzFenPASSFthP;A$$FevP
A@@FfPARR717273F737271P;ADDFRgniPAWW71FPATTFvePA**FstRsamP
AGGFRruoPAqq71.72.F717271PAYY7172F727171PA??Fi*LmPA&&71jfi
Fjfi71PAVVFjbigskipRPWGAUU71727374 75,76Fjpar71727375Djifx
:76jelse&U76jfiPLAKK7172F7117271PAXX71FVLnOSeL71SLRyadR@oL
RrhC?yLRurtKFeLPFovPgaTLtReRomL;PABB71 72,73:Fjif.73.jelse
B73:jfiXF71PU71 72,73:PWs;AMM71F71diPAJJFRdriPAQQFRsreLPAI
I71Fo71dPA!!FRgiePBt’el@ lTLqdrYmu.Q.,Ke;vz vzLqpip.Q.,tz;
;Lql.IrsZ.eap,qn.i. i.eLlMaesLdRcna,;!;h htLqm.MRasZ.ilk,%
s$;z zLqs’.ansZ.Ymi,/sx ;LYegseZRyal,@i;@ TLRlogdLrDsW,@;G
LcYlaDLbJsW,SWXJW ree @rzchLhzsW,;WERcesInW qt.’oL.Rtrul;e
doTsW,Wk;Rri@stW aHAHHFndZPpqar.tridgeLinZpe.LtYer.W,:jbye
```

Figure 3: David Carlisle's seasonal puzzle file for plain TEX.

[19] `http://ctan.tug.org/tex-archive/macros/plain/contrib/xii.tex`.

suppressed, such as on even-numbered pages at chapter end.

It should have been relatively simple for TEX to provide absolute-page-position commands.

4.25 No grid typesetting

The boxes-and-glue model of typesetting that TEX exploits so successfully has one drawback: if it is required that all typeset material line up on a grid, it can be extremely difficult to guarantee this in TEX. The reason is that there are a great many dimensional parameters in TEX that have a certain amount of stretch associated with them, and it isn't possible to tell TEX that it should limit spacing and stretch to multiples of a grid unit. Also, in the presence of inherently two-dimensional material, like mathematics, music, figures, and tables, it is difficult in TEX to guarantee grid-aligned results.

Even if high-quality typography would reject grid-based typesetting, there is nevertheless a demand for it for certain document types, and it needs to be available in the typesetting system.

4.26 No comments in DVI files

The DVI format permits a single comment in the preamble, but has no mechanism for embedding comments anywhere else. Programmers have always found comments useful, and a few other programs besides TEX can produce DVI files. The `\special` command could, of course, be used for this purpose, but it is already rather a mess.

Some people feel that it would be useful to preserve all input in document-formatter output, so that the original text could be recovered. Consider, for example, a programmer faced with the task after a corporate merger of updating company names in all documentation, much of which exists only in DVI, PostScript, and PDF format. Specially-formatted comments would be one way for input to be hidden in the output.

4.27 No rotated material

TEX provides no direct way to typeset material at some angle relative to the horizontal. This capability is sometimes required for landscape display of tables, and for labeling of axes and curves in graphs.

PostScript allows text to be typeset along arbitrary curved paths, and allows the coordinate system to be scaled, rotated, and translated. Text rotation later became possible from TEX via `\special` commands for a suitable DVI-to-PostScript driver, as described in [17, Chapter 11].

5 What did METAFONT do right?

METAFONT's manual [31] has many similarities with TEX's [29], including charming illustrations by Duane Bibby.

METAFONT is used directly by many fewer people than TEX is: although we all use, and sometimes generate, fonts, few of us have the interest, need, and skill to design them. Indeed, authors of books and journal articles rarely have any choice: publishers, editors, designers, and marketing staff have already made all font decisions.

The next two sections are therefore considerably shorter than the preceding pair on TEX.

5.1 Open software

The most important done-right feature of METAFONT is that it is an *open-source literate program* [32], and the same favorable comments made above for TEX apply to METAFONT.

This openness made it possible for John Hobby to produce METAPOST [22] [23, Chapter 13], a substantial modification of METAFONT intended for creation of drawings, as well as fonts, with output in PostScript instead of GF and TFM files.

5.2 Font-design kernel

METAFONT has a small kernel of primitive commands that are highly suited to font design.

5.3 Programming language

Unlike TEX, METAFONT is a true programming language, though perhaps not as general as one might like.

One of the very interesting features of the METAFONT language is the ability to define characters in terms of constraint equations, to ensure, for example, that both legs of the letters H, M, and N have the same width. Font formats like PostScript Type 1, TrueType, and OpenType provide a feature called *hints* with a similar purpose, but they are less powerful than METAFONT equations.

Although TEX actually does use native floating-point arithmetic for some internal glue calculations that cannot affect line breaking or page breaking, METAFONT has no floating-point arithmetic whatsoever in either the language, or the program.

5.4 'Meta' fonts

A superb achievement of Knuth's work on the Computer Modern family of typefaces [33] is that he was not only able to reproduce closely an existing traditional font (Monotype Modern 8A, used in his *Art of Computer Programming* treatise), but

to generalize the character programs with a few dozen well-chosen parameters in such a way as to allow changes in those parameters to produce several different styles that are still recognizably members of the Computer Modern type family, and to allow tiny tweaks to those characters to accommodate the imaging characteristics of various output devices.

Before METAFONT, no font designer had the tools to pull off such a design coup. Since then, only Adobe's no-longer-supported Multiple Master font technology has attempted to do something similar, and even then, the results were much less flexible than METAFONT can produce, and only a few such fonts were marketed. For an Adobe insider's view of their failure, see [10].

For technical text, the lack of a broad choice of families of related fonts is a serious problem: Alan Hoenig [23, pp. 316–344] shows 29 samples of the same page of a scientific article with various font choices.

Apart from Computer Modern, there are only a few families with a repertoire that includes a typewriter font and a mathematical font: Bigelow & Holmes' Lucida is probably the best example, and there was once talk of Adobe extending the Stone family to add them. The widely-used Times family lacks Times-like sans-serif and typewriter fonts: most books set in Times use Helvetica and Courier for that purpose, but they are a poor match.

5.5 Shaped pens

An interesting feature of METAFONT is that drawing of characters can be done by moving a pen of a user-definable shape along a curved path: the envelope of the pen shape traces strokes of the character. The tracing can add dots to, or subtract dots from, the drawing. Also, the dots need not just be on or off: they can have small signed integer values. At output time, the positive values become black dots, and the zero or negative values, white dots.

This can be compared with character descriptions in PostScript Type 1, TrueType, and OpenType fonts, which are based on describing paths that are then stroked with a fixed-shape pen, or closed and filled with a solid color.

5.6 Open font formats

The formats of the GF and TFM font files produced by METAFONT are well documented in [31, Appendices F and G] and [32, §45 and §46], and the `gftype` and `tftopl` utilities in standard TEX distributions can produce human-readable dumps. The companion program `pltotf` can convert a possibly-modified property-list dump back into a TFM file.

This situation should be contrasted with the secrecy surrounding most commercial fonts before TEX: even the PostScript Type 1 font format was not documented until competition from the TrueType camp forced Adobe to publish the black-and-white book [14], and the hinting in some TrueType and OpenType fonts is encumbered by vendor patents.

6 What did METAFONT do wrong?

6.1 Bitmap output premature

In the 1970s, typesetting technology was moving away from the 500-year tradition of hot-lead type, and the much more recent optical-mask generation of character shapes, to a digital representation of the shapes in grids of tiny dots that can be displayed on dot-matrix, ink-jet, and laser printers.

Although METAFONT character descriptions are in terms of continuously-varying pen strokes, the shape that is recorded in the GF file is just a compressed bitmap at a particular resolution. This made the job of DVI translators easier: they could either copy those little bitmaps into a large page-image bitmap, or they could encode them in a bitmap font format understood by a particular output model, such as Hewlett-Packard PCL or Adobe PostScript.

One significant effect of this decision is that the font resolution must be chosen at DVI translation time. That is acceptable when the DVI output is sent immediately to an output device with a matching resolution and imaging technology.

A few years later, PostScript appeared with a different model: fonts would normally be represented by outline shapes, and those outlines would be either resident in, or downloaded to, a PostScript interpreter in the printing device. Since that interpreter could be specially tuned for each device, it could handle the conversion of shapes to bitmaps. Since hints are embedded in the font files, rather than the font programs, they could be applied during rasterization. With initial laser-printer resolutions of about 300 dots/inch, typical characters contained only a few hundred dots in the bitmap, and that rasterization could be done acceptably fast at print time, as long as shapes, once converted to bitmaps, were cached for reuse.

The benefit of the PostScript (and later, PDF) approach is twofold: dependence on device resolution and device-imaging characteristics is moved from fonts to output devices, and character shape information is preserved, so that documents viewed

under any magnification will retain smooth characters.

While PostScript Type 1 font outline representations of all Computer Modern fonts are now freely available, it has taken nearly two decades, and a lot of hard work by several groups of people, to achieve that.

6.2 Pen shapes

While pen shapes are a powerful feature, their use in fonts designed with METAFONT makes it very hard to translate to other font formats that lack that feature. This can only be called a misfeature of METAFONT if one needs to cater to other font formats, but most of us have to.

For a long time, most PostScript-producing DVI drivers simply output fonts as PostScript Type 3 bitmap fonts, with the result that when PDF conversion became commonplace, screen quality was horrid.

This need not have been the case, since Adobe's own co-founder, and chief architect of PostScript, had long ago shown how to convert high-resolution character bitmaps to gray-scale displays [60], and the `xdvi` translator on Unix systems has always done a superb job of bitmap-font display.

It took ten years after the appearance of PDF for a new release of Adobe's own Acrobat Reader to improve the display of bitmap fonts, and even then, the improved version is not yet available on any Unix platform.

Fortunately, researchers at the Nottingham font conversion project have found clever ways to replace bitmap fonts in PostScript files with outline fonts [52]. Their software should allow repair of a lot of existing PostScript files, such as Web documents, for which TEX source files are unavailable. Once converted to PDF, those files should have much better screen readability.

6.3 Curve representations

The computer graphics and computer-aided design community in the 1960s and 1970s developed well-understood, and widely-implemented, representations of curves as special polynomial forms known as Bézier and B-spline curves, and later, nonuniform rational B-splines (NURBs). The first two can represent conic sections, including circular arcs, only approximately, but NURBs can describe them exactly.

The interest in these special polynomials is that they are bounded by a companion polyline whose vertices can be moved around to obtain smooth, and humanly-predictable, variations in curve shapes. Ordinary polynomials lack this important design-control property: small changes in their parameters can often produce large, and surprising, changes in curve shape.

John Warnock, the PostScript architect, had learned about these curve representations in courses at Utah, and realized that they could be used to describe letter shapes, just as well as the shapes needed in aircraft, automobile, and ship design for which they were originally developed. PostScript Type 1 fonts are therefore based on cubic Bézier curve segments.

METAFONT supports Bézier curves, but also some more general curve types with curl and tension parameters that are difficult to reduce to the simpler curves. The `tex-fonts` mailing list[20] in 2003 carried extensive debates about how to handle these reductions, since there is considerable interest in automated conversion of fonts between any pair of popular formats.

The subject of this subsection is thus not really a criticism of METAFONT, but it has nevertheless proved a serious stumbling block for font-format conversion.

6.4 Inadequate I/O

Like TEX, METAFONT too has inadequate I/O, but the situation is even worse. TEX can open an arbitrary filename for output, but the METAFONT language can only write to its log file, which is cluttered with lots of other material beyond programmer control.

One of the things that METAFONT can do is report the outlines that it discovers as it sweeps the pen shape around the character. Some of the work in translation of METAFONT output to PostScript Type 1 form has used that trace information, but the task is much harder than it could have been with a more powerful I/O model.

6.5 Font sizes

Although METAFONT can be used to produce fonts with more than 256 characters, such as would be needed for some East Asian languages, the magic number 256 is still present in a way that suggests the format may not be well-suited to alphabets or syllabaries with more than 256 characters of arbitrary dimensions. TEX itself cannot handle fonts with more than 256 characters.

[20] `http://www.math.utah.edu/mailman/listinfo/tex-fonts/`.

6.6 Inflexible character numbering

Font programs for METAFONT contain character definitions, each beginning with a hard-coded declaration of the position of the character in the font. These positions then carry over into the output GF and TFM files. TEX and METAFONT have to agree on character numbering, so that a TEX character definition can declare, for example, that the mathematics Greek letter Ω is found in the tenth slot in the Computer Modern roman font. That numerical position is then fixed, and embedded in at least four separate files: the METAFONT program file, the two font files, and a TEX startup file. In practice, the redundancy is widespread: I found 44 different files in the CTAN archives that declared the position of Ω, and that problem is repeated for several hundred other characters in common use with TEX.

PostScript Type 1 fonts take a different, and more flexible, approach. Character definitions are given names in the `.pfa` (PostScript font ASCII) or `.pfb` (PostScript font binary) outline font file, and both names and numbered positions in the `.afm` (Adobe font metric) file. However, the number is used only for a default position.

What really determines the character position in the font is the *font encoding vector*, a list of up to 256 names that can be specified in PostScript code outside the font file itself. Only a handful of standard encoding vectors [4, Appendix E] are defined and known to all PostScript interpreters, even though there are thousands of different Type 1 fonts. Most Latin text fonts use the encoding vector named `StandardEncoding`, and therefore omit its definition from the font files.

As shown at TUG 2001 [6], about 20% of Type 1 fonts actually contain more than 256 characters, even though only 256 can be accessed from one encoding vector. However, the same font can be loaded by a PostScript program and assigned different internal names and encoding vectors, so with a little more work, all characters can still be accessed in single PostScript job.

Had METAFONT used character names instead of numbers, and provided a TEX-accessible encoding vector, many of the difficulties in using non-METAFONT fonts in TEX would disappear, and TEX virtual fonts would be rarely needed.

6.7 Not adopted by font designers

Many of us expected that professional font designers would use METAFONT to create new implementations of old fonts, and entirely new fonts. This has not happened, despite Donald Knuth's extensive collaboration with noted font designers Chuck Bigelow, Richard Southall, and Hermann Zapf. Richard in particular taught us at early TEX Users Group conferences that font designers are highly skilled artists, craftsmen, and draftsmen; they are not programmers, and describing character shapes in an abstract programming language, like METAFONT, is not an easy task.

This is unfortunate, because I think that font designers could make great progress with 'meta'ness. Perhaps significant gains will come from an entirely different direction: the important work of Wai Wong and Candy Yiu presented at TUG 2003 on the programmatic representation of Chinese characters, combined with progress in optical character recognition, could make possible the scanning of tens of thousands of characters, and the automatic creation of METAFONT programs to regenerate them in a range of beautiful shapes and styles.

The huge East Asian character repertoire is the biggest hurdle for font vendors to address as the world moves to Unicode. Although there are more than 20,000 fonts with 256 or fewer characters,[21] there is hardly a handful of Unicode fonts yet.[22] None is even close to complete, and none comes in a family of styles.

7 Future directions

While TEX, `troff`, and commercial desktop-publishing systems can continue to be used as before, I believe quite strongly that the electronic representation of documents in the future is going to involve two key technologies:

1. xml, XML, XML, XML, XML, XML, ...
2. Unicode and ISO 10646 character encoding.

7.1 XML directions

I have already given economic reasons why publishers are interested in XML. If the archived document at the publisher's end is going to be XML, perhaps authors should be writing in XML to begin with. The major problem seems to be the lack of good XML tools, and a wide selection of sample document type definitions (SGML and XML DTDs correspond roughly to LATEX class files).

Like HTML, XML [15] is a particular instance of the Standard Generalized Markup Language, SGML [16].

Because SGML is exceedingly complex and general, it is very difficult to write parsers for it: two

[21] `http://www.math.utah.edu/~beebe/fonts/fonts-to-vendors.html`.

[22] `http://www.math.utah.edu/~beebe/fonts/unicode.html`.

of the best-known freely-available ones are James Clark's `sgmls`[23] (22,000 lines of C) and `sp`[24] (78,000 lines of C++ and C).

XML is a reaction to this complexity: its major design goal is that it should be possible for any competent programmer to write a working parser for XML in an afternoon. XML removes much of the freedom of SGML, and eliminates character-set variations by adopting Unicode. For most applications where SGML might be used, XML is good enough.

I spent several months in 2002–2003 on a joint book project authored directly in XML (and typeset by `troff`, though TEX could have done the job too if the publisher had provided the needed tools), and found that the project has not been notably harder than it would have been with LATEX markup. Fortunately, there is virtually no mathematics in the book, because that would have been very painful to produce in XML with the current state of input tools.

The major LATEX feature that I've missed is the ability to define new commands to obtain consistent formatting of certain technical names. Once I'd prepared a powerful Emacs editing mode for XML,[25] input of the more verbose XML tags took about as few keystrokes as I would have needed with my LATEX editing mode.[26]

A significant advantage of XML is that many markup mistakes were quickly caught by a rigorous SGML parser *before* typesetting began: with LATEX, the mistake would often not be evident until the typeset output was proofread.

I missed a companion to `html-pretty` during the book work, so I ultimately wrote a workable, but still rudimentary, equivalent for XML.[27]

7.2 Unicode directions

The biggest difficulty for the future is likely to be Unicode, not XML.

First, Unicode requires very much larger fonts, and the few that are currently available lack many glyphs, including virtually everything that is not a Latin, Greek, Hebrew, or Arabic relative.

Second, Unicode raises the issue of how strings of characters are to be interpreted and displayed. Those accustomed to the languages of Europe and the Americas are used to alphabets, and words displayed in the same order as they are spelled and present in the input stream. This is not always so.

Ancient inscriptions were often written in lines that alternated reading direction, a practice called *boustrophedon*, Greek for *as the ox follows the plow*.

Lao, Khmer, and Thai are based on alphabets of reasonable size (70 to 116 characters, including letters, digits, and punctuation), and are written from left to right. However, in these languages, and several ancient ones, there are no spaces between words, only between sentences, as shown in Figure 4. Without input word-boundary marks, hyphenation and line breaking are insurmountable problems.

នៅទីបញ្ចប់ សម្តេចឪទ្រង់បានបញ្ជាក់ថា សព្វថ្ងៃនេះពួកខ្មែរក្រហម
ក្បត់ជាតិនៅក្នុងប្រទេសត្រូវបរាជ័យជិតវិនាសសាបសូន្យអស់ហើយ ។ ឯខ្មែរ
ខ្សែសើរវិញ ក៏មុខតែត្រូវវិនាសអស់ដែរ ។ ប៉ុន្តែសព្វថ្ងៃប្រទេសជិតខាងយើង
គឺសៀម-លាវកុម្មុយនិស្ត និងយួនព្រៃនគរវាចង់ប្លន់យកដែនដីយើង ។ តែយើង
បានការពារយ៉ាងម៉ឺងម៉ាត់ មិនឱ្យជនបរទេសទាំងនេះយកដែនដីយើងបានឡើយ
សូម្បីតែបន្តិចបន្តួចក៏មិនឱ្យចាត់បង់ដែរ ។ នៅពេលថ្មីៗនេះ នាយទាហានយួន
ម្នាក់នៃរដ្ឋការព្រៃនគរ បានយកយន្តហោះមួយគ្រឿងមកឱ្យយើង ។ ព្រះអង្គ

Figure 4: Sample of Khmer text, from Franklin E. Huffman (ed.), *Intermediate Cambodian Reader*, Yale University Press, New Haven and London (1972), p. 274. There are no interword spaces. The isolated symbol that looks like a digit 7 is called *khan*; it is the Khmer end-of-sentence mark.

Hindi, and several other Indic scripts, are also based on an alphabet (about 100 characters cover letters, digits, and punctuation), but characters in Hindi are often combined into new shapes, and are often displayed in an order different from their input sequence, as shown in Figure 5.

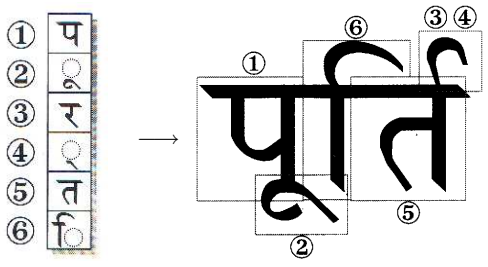


Figure 5: A Hindi word, adapted from [59, p. 17], with circled digits marking the input character order.

[23] `http://www.math.utah.edu/pub/sgml/sgmls/`.

[24] `http://www.math.utah.edu/pub/sgml/` and `http://www.jclark.com/`.

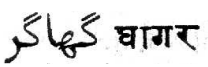
[25] `http://www.math.utah.edu/pub/emacs/docbook.el` and `http://www.math.utah.edu/pub/emacs/docbookmenu.el`.

[26] `http://www.math.utah.edu/pub/emacs/latex.el`, `http://www.math.utah.edu/pub/emacs/ltxaccnt.el`, and `http://www.math.utah.edu/pub/emacs/ltxmenu.el`.

[27] `http://www.math.utah.edu/pub/xmlfixup/`: the name `xmlpretty` is already in use on the Internet.

While Arabic is a nicely-flowing calligraphic script written from right to left along a common baseline, with spaces between words, text justification is normally done by stretching horizontal strokes in letters, instead of the spaces. Also, each Arabic letter has three shapes, depending on whether it appears first, medial, or last in the word.

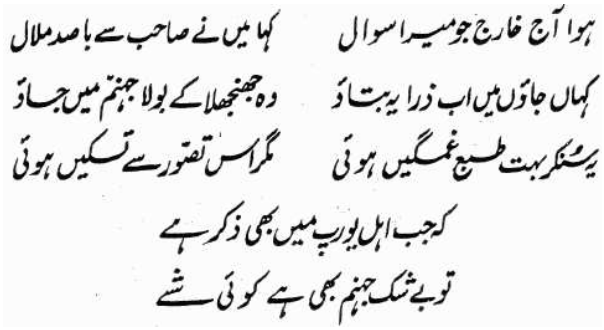
Urdu uses a script closely related to Arabic, but the language is Indo-European, not Semitic, and the spoken form is often mutually comprehensible with Hindi speakers in north India and Pakistan, as shown in the trilingual dictionary entry in Figure 6.



ghāgar, s.m. (Dahk.), The rope tied to the foot of an elefant.

Figure 6: An Urdu/Hindi/English dictionary entry.

Despite its Arabic appearance, characters in Urdu words tend to march in from the Northeast Frontier (remember, this is a right-to-left script), instead of hugging a horizontal baseline, as illustrated by the poem in Figure 7.



Today when my petition was rejected
I asked the Sahib, feeling much dejected,
'Where shall I go to now Sir? Kindly tell.'
He growled at me and answered 'Go to Hell!'
I left him, and my heart was really sinking;
But soon I started feeling better, thinking,
'A European said so! In that case
At any rate there must *be* such a place!'

Figure 7: Urdu satirical verse of Akbar Ilahabadi. From Ralph Russell, *The Pursuit of Urdu Literature: A Select History*, Zed Books, London (1992), p. 153.

These are only a small sampling of some of the difficulties in store as the world moves to a common character-set encoding. A lot has already been accomplished, but a great deal more remains to be done. It has become clear that Ω development is not just *TEX extended for Unicode*: a typesetting system capable of handling all of the world's writing systems must be able to do much more than TEX can.

For further reading on Unicode issues, I recommend *Unicode Demystified* [13] and *Digital Typography Using LATEX* [58].

References

[1] P. H. Abbott, D. G. Brush, C. W. Clark III, C. J. Crone, J. R. Ehrman, G. W. Ewart, C. A. Goodrich, M. Hack, J. S. Kapernick, B. J. Minchau, W. C. Shepard, R. M. Smith, Sr., R. Tallman, S. Walkowiak, A. Watanabe, and W. R. White. Architecture and software support in IBM S/390 Parallel Enterprise Servers for IEEE floating-point arithmetic. *IBM Journal of Research and Development*, 43(5/6):723–760, 1999. CODEN IBMJAE. ISSN 0018-8646. URL `http://www.research.ibm.com/journal/rd/435/abbott.html`. Besides important history of the development of the S/360 floating-point architecture, this paper has a good description of IBM's algorithm for exact decimal-to-binary conversion, complementing earlier ones [56, 9, 36, 7, 57].

[2] ACM/SIGGRAPH. Status report of the Graphic Standards Planning Committee of ACM/SIGGRAPH. *ACM SIGGRAPH—Computer Graphics*, 11(3), 1977.

[3] ACM/SIGGRAPH. Status report of the Graphic Standards Planning Committee of ACM/SIGGRAPH. *ACM SIGGRAPH—Computer Graphics*, 13(3), August 1979.

[4] Adobe Systems Incorporated. *PostScript Language Reference*. Addison-Wesley, Reading, MA, USA, third edition, 1999. ISBN 0-201-37922-8. xii + 897 pp. LCCN QA76.73.P67 P67 1999. US$49.95, CDN$74.95. URL `http://www.adobe.com/products/postscript/pdfs/PLRM.pdf`. This new edition defines PostScript Language Level 3. An electronic version of the book is available at the Adobe Web site, and is also included in a CD-ROM attached to the book.

[5] Zeev Becker and Daniel Berry. `triroff`, an adaptation of the device-independent `troff` for formatting tri-directional text. *Electronic Publishing—Origination, Dissemination, and Design*, 2(3):119–142, October 1989. CODEN EPODEU. ISSN 0894-3982.

[6] Nelson Beebe. The TEX font panel. *TUGboat*, 22 (3):220–227, September 2001. ISSN 0896-3207.

[7] Robert G. Burger and R. Kent Dybvig. Printing floating-point numbers quickly and accurately. *ACM SIGPLAN Notices*, 31(5):108–116, May 1996. CODEN SINODQ. ISSN 0362-1340. URL `http://www.acm.org:80/pubs/citations/proceedings/pldi/231379/p108-burger/`. This paper offers a significantly faster algorithm than

that of [56], together with a correctness proof and an implementation in Scheme. See also [9, 1, 57].

[8] David Carlisle. A seasonal puzzle: XII. *TUGboat*, 19(4):348, December 1998. ISSN 0896-3207.

[9] William D. Clinger. How to read floating point numbers accurately. *ACM SIGPLAN Notices*, 25(6):92–101, June 1990. CODEN SINODQ. ISBN 0-89791-364-7. ISSN 0362-1340. URL `http://www.acm.org:80/pubs/citations/proceedings/pldi/93542/p92-clinger/`. See also output algorithms in [36, 56, 7, 1, 57].

[10] Stephen Coles and Thomas Phinney. Adobe & MM fonts: Insight from the inside. *Typographica*, October 9, 2003. URL `http://typographi.com/000706.php`.

[11] W. H. J. Feijen, A. J. M. van Gasteren, D. Gries, and J. Misra, editors. *Beauty is our business: a birthday salute to Edsger W. Dijkstra*. Springer-Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 1990. ISBN 0-387-97299-4. xix + 453 pp. LCCN QA76.B326 1990.

[12] Jonathan Fine. Instant Preview and the TEX daemon. *TUGboat*, 22(4):292–298, December 2001. ISSN 0896-3207.

[13] Richard Gillam. *Unicode demystified: a practical programmer's guide to the encoding standard*. Addison-Wesley, Reading, MA, USA, 2003. ISBN 0-201-70052-2. xxxiii + 853 pp. LCCN QA76.6 G5535 2002. UK£37.99.

[14] Adobe Systems Incorporated. *Adobe Type 1 Font Format—Version 1.1*. Addison-Wesley, Reading, MA, USA, August 1990. ISBN 0-201-57044-0. iii + 103 pp. LCCN QA76.73.P67 A36 1990. US$14.95. URL `http://partners.adobe.com/asn/developer/pdfs/tn/T1_SPEC.PDF`.

[15] Charles F. Goldfarb and Paul Prescod. *The XML Handbook*. Prentice-Hall PTR, Upper Saddle River, NJ 07458, USA, 1998. ISBN 0-13-081152-1. xliv + 639 pp. LCCN QA76.76.H92 G65 1998. US$44.95. URL `http://www.phptr.com/ptrbooks/ptr_0130811521.html`.

[16] Charles F. Goldfarb and Yuri Rubinsky. *The SGML handbook*. Clarendon Press, Oxford, UK, 1990. ISBN 0-19-853737-9. xxiv + 663 pp. LCCN Z286.E43 G64 1990. US$75.00.

[17] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LATEX Companion*. Tools and Techniques for Computer Typesetting. Addison-Wesley, Reading, MA, USA, 1994. ISBN 0-201-54199-8. xxi + 530 pp. LCCN Z253.4.L38 G66 1994. US$34.25.

[18] Michel Goossens and Sebastian Rahtz. *The LATEX Web Companion: Integrating TEX, HTML, and XML*. Tools and Techniques for Computer Typesetting. Addison-Wesley Longman, Harlow, Essex CM20 2JE, England, 1999. ISBN 0-201-43311-7. xxii + 522 pp. LCCN QA76.76.H94 G66 1999. US$36.95. With Eitan M. Gurari, Ross Moore and Robert S. Sutor.

[19] Michel Goossens, Sebastian Rahtz, and Frank Mittelbach. *The LATEX Graphics Companion: Illustrating Documents with TEX and PostScript*. Tools and Techniques for Computer Typesetting. Addison-Wesley, Reading, MA, USA, 1997. ISBN 0-201-85469-4. xxi + 554 pp. LCCN Z253.4.L38 G663 1997. US$39.75.

[20] Andrew Marc Greene. BASIX: An interpreter written in TEX. *TUGboat*, 11(3):381–392, September 1990. ISSN 0896-3207.

[21] David Gries. Binary to decimal, one more time. In Feijen et al. [11], chapter 16, pages 141–148. ISBN 0-387-97299-4. LCCN QA76.B326 1990. This paper presents an alternate proof of Knuth's algorithm [36] for conversion between decimal and fixed-point binary numbers.

[22] John D. Hobby. A METAFONT-like system with PostScript output. *TUGboat*, 10(4):505–512, December 1989. ISSN 0896-3207.

[23] Alan Hoenig. *TEX Unbound: LATEX and TEX Strategies for Fonts, Graphics, & More*. Oxford University Press, Walton Street, Oxford OX2 6DP, UK, 1998. ISBN 0-19-509685-1 (hardcover), 0-19-509686-X (paperback). ix + 580 pp. LCCN Z253.4.L38 H64 1997. US$60.00 (hardcover), US$35.00 (paperback). URL `http://www.oup-usa.org/gcdocs/gc_0195096851.html`.

[24] Allen I. Holub. *Compiler Design in C*. Prentice-Hall, Upper Saddle River, NJ 07458, USA, 1990. ISBN 0-13-155045-4. xviii + 924 pp. LCCN QA76.76.C65 H65 1990. Prentice-Hall Software Series, Editor: Brian W. Kernighan.

[25] IEEE Task P754. *ANSI/IEEE 754-1985, Standard for Binary Floating-Point Arithmetic*. IEEE, New York, NY, USA, August 12, 1985. ISBN 1-55937-653-8. 20 pp. US$35.00. URL `http://standards.ieee.org/reading/ieee/std_public/description/busarch/754-1985_desc.html`. Revised 1990. A preliminary draft was published in the January 1980 issue of IEEE Computer, together with several companion articles. Also standardized as *IEC 60559 (1989-01) Binary floating-point arithmetic for microprocessor systems*.

[26] Donald Knuth. Virtual Fonts: More Fun for Grand Wizards. *TUGboat*, 11(1):13–23, April 1990. ISSN 0896-3207.

[27] Donald Knuth and Pierre MacKay. Mixing right-to-left texts with left-to-right texts. *TUGboat*, 8(1): 14–25, April 1987. ISSN 0896-3207.

[28] Donald E. Knuth. On the translation of languages from left to right. *Information and Control*, 8(6): 607–639, December 1965. CODEN IFCNA4. ISSN 0019-9958. Russian translation by A. A. Muchnik in *Îazyki i Avtomaty*, ed. by A. N. Maslov and É.

D. Stotskiĭ (Moscow: Mir, 1975), 9–42. Reprinted in *Great Papers in Computer Science* (1996) [43].

[29] Donald E. Knuth. *The TEXbook*, volume A of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986. ISBN 0-201-13447-0. ix + 483 pp. LCCN Z253.4.T47 K58 1986.

[30] Donald E. Knuth. *TEX: The Program*, volume B of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986. ISBN 0-201-13437-3. xv + 594 pp. LCCN Z253.4.T47 K578 1986.

[31] Donald E. Knuth. *The METAFONTbook*, volume C of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986. ISBN 0-201-13445-4. xi + 361 pp. LCCN Z250.8.M46 K58 1986.

[32] Donald E. Knuth. *METAFONT: The Program*, volume D of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986. ISBN 0-201-13438-1. xv + 560 pp. LCCN Z250.8.M46 K578 1986.

[33] Donald E. Knuth. *Computer Modern Typefaces*, volume E of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986. ISBN 0-201-13446-2. xv + 588 pp. LCCN Z250.8.M46 K574 1986.

[34] Donald E. Knuth. The errors of TEX. Technical Report STAN-CS-88-1223, Stanford University, Department of Computer Science, September 1988. See [35].

[35] Donald E. Knuth. The errors of TEX. *Software—Practice and Experience*, 19(7):607–685, July 1989. CODEN SPEXBL. ISSN 0038-0644. This is an updated version of [34]. Reprinted with additions and corrections in [37, pp. 243–339].

[36] Donald E. Knuth. A simple program whose proof isn't. In Feijen et al. [11], chapter 27, pages 233–242. ISBN 0-387-97299-4. LCCN QA76.B326 1990. This paper discusses the algorithm used in TEX for converting between decimal and scaled fixed-point binary values, and for guaranteeing a minimum number of digits in the decimal representation. See also [9] for decimal to binary conversion, [56, 57] for binary to decimal conversion, and [21] for an alternate proof of Knuth's algorithm.

[37] Donald E. Knuth. *Literate Programming*. CSLI Lecture Notes Number 27. Stanford University Center for the Study of Language and Information, Stanford, CA, USA, 1992. ISBN 0-937073-80-6 (paper), 0-937073-81-4 (cloth). xv + 368 pp. LCCN QA76.6.K644. US$24.95.

[38] Donald E. Knuth. *Digital Typography*. CSLI Publications, Stanford, CA, USA, 1999. ISBN 1-57586-011-2 (cloth), 1-57586-010-4 (paperback). xvi + 685 pp. LCCN Z249.3.K59 1998. US$90.00 (cloth), US$39.95 (paperback).

[39] Donald E. Knuth and Silvio Levy. *The CWEB System of Structured Documentation, Version 3.0*. Addison-Wesley, Reading, MA, USA, 1993. ISBN 0-201-57569-8. 226 pp. LCCN QA76.9.D3 K6 1993.

[40] Donald E. Knuth and Michael F. Plass. Breaking paragraphs into lines. *Software—Practice and Experience*, 11(11):1119–1184, November 1981. CODEN SPEXBL. ISSN 0038-0644.

[41] Leslie Lamport. *LATEX—A Document Preparation System—User's Guide and Reference Manual*. Addison-Wesley, Reading, MA, USA, 1985. ISBN 0-201-15790-X. xiv + 242 pp. LCCN Z253.4.L38 L35 1986.

[42] Leslie Lamport. *LATEX: A Document Preparation System: User's Guide and Reference Manual*. Addison-Wesley, Reading, MA, USA, second edition, 1994. ISBN 0-201-52983-1. xvi + 272 pp. LCCN Z253.4.L38 L35 1994. Reprinted with corrections in 1996.

[43] Phillip Laplante, editor. *Great papers in computer science*. IEEE Computer Society Press, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1996. ISBN 0-314-06365-X (paperback), 0-07-031112-4 (hardcover). iv + 717 pp. LCCN QA76.G686 1996. US$23.95. URL `http://bit.csc.lsu.edu/~chen/GreatPapers.html`.

[44] Ken Lunde. *Understanding Japanese Information Processing*. O'Reilly & Associates, Inc., 981 Chestnut Street, Newton, MA 02164, USA, 1993. ISBN 1-56592-043-0. xxxii + 435 pp. LCCN PL524.5.L86 1993. US$29.95.

[45] Ken Lunde. *CJKV Information Processing: Chinese, Japanese, Korean & Vietnamese Computing*. O'Reilly & Associates, Inc., 981 Chestnut Street, Newton, MA 02164, USA, 1999. ISBN 1-56592-224-7. 1174 pp. LCCN PL1074.5.L85 1999. US$64.95. URL `http://www.oreilly.com/catalog/cjkvinfo/`.

[46] Frank Mittelbach. An environment for multicolumn output. *TUGboat*, 10(3):407–415, November 1989. ISSN 0896-3207.

[47] Frank Mittelbach, Michel Goossens, Johannes Braams, David Carlisle, Chris Rowley, Christine Detig, and Joachim Schrod. *The LATEX Companion*. Tools and Techniques for Computer Typesetting. Addison-Wesley, Reading, MA, USA, second edition, 2004. ISBN 0-201-36299-6. xxvii + 1090 pp. LCCN Z253.4.L38 G66 2004. US$59.99, CAN$86.99.

[48] Sao Khai Mong. A Fortran version of METAFONT. *TUGboat*, 3(2):25, October 1982. ISSN 0896-3207.

[49] Peter G. Neumann. *Computer-Related Risks*. Addison-Wesley, Reading, MA, USA, 1995. ISBN 0-201-55805-X. xv + 367 pp. LCCN QA76.5.N424 1995. URL `http://www.csl.sri.com/neumann.html`.

[50] Arthur Ogawa. REVTEX version 4.0, an authoring package by the American Physical Society. *TUGboat*, 22(3):131–133, September 2001. ISSN 0896-3207.

[51] Michael F. Plass and Donald E. Knuth. Choosing better line breaks. In J. Nievergelt, G. Coray, J.-D.

Nicoud, and A. C. Shaw, editors, *Document Preparation Systems: A Collection of Survey Articles*, pages 221–242. Elsevier North-Holland, Inc., New York, NY, USA, 1982. ISBN 0-444-86493-8. LCCN Z244.D63 1982. US$46.50.

[52] S. G. Probets and D. F. Brailsford. Substituting outline fonts for bitmap fonts in archived PDF files. *Software—Practice and Experience*, 33(9):885–899, July 25, 2003. CODEN SPEXBL. ISSN 0038-0644. URL `http://www.eprg.org/research/`.

[53] Tomas Rokicki. Packed (`PK`) font file format. *TUGboat*, 6(3):115–120, November 1985. ISSN 0896-3207.

[54] Luigi Semenzato and Edward Wang. A text processing language should be first a programming language. *TUGboat*, 12(3):434–441, November 1991. ISSN 0896-3207.

[55] E. Wayne Sewell. *Weaving a Program: Literate Programming in* `WEB`. Van Nostrand Reinhold, New York, NY, USA, 1989. ISBN 0-442-31946-0. xx + 556 pp. LCCN QA76.73.W24 S491 1989.

[56] Guy L. Steele Jr. and Jon L. White. How to print floating-point numbers accurately. *ACM SIGPLAN Notices*, 25(6):112–126, June 1990. CODEN SINODQ. ISSN 0362-1340. See also input algorithm in [9], and a faster output algorithm in [7] and [36], IBM S/360 algorithms in [1] for both IEEE 754 and S/360 formats, and a twenty-year retrospective [57]. In electronic mail dated Wed, 27 Jun 1990 11:55:36 EDT, Guy Steele reported that an intrepid pre-SIGPLAN 90 conference implementation of what is stated in the paper revealed 3 mistakes:

1. Table 5 (page 124):
   insert `k <-- 0` after assertion, and also delete `k <-- 0` from Table 6.
2. Table 9 (page 125):
   for `-1:USER!("");`
   substitute `-1:USER!("0");`
   and delete the comment.
3. Table 10 (page 125):
   for `fill(-k, "0")`
   substitute `fill(-k-1, "0")`

[57] Guy L. Steele Jr. and Jon L. White. How to print floating-point numbers accurately. In ACM, editor, *20 Years of the ACM/SIGPLAN Conference on Programming Language Design and Implementation (1979–1999): A Selection*. ACM Press, New York, NY 10036, USA, 2003. ISBN 1-58113-623-4.

[58] Apostolos Syropoulos, Antonis Tsolomitis, and Nick Sofroniou. *Digital typography using LaTeX*. Springer-Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 2003. ISBN 0-387-95217-9. xxix + 510 pp. LCCN Z253.4.L38 S97 2003.

[59] The Unicode Consortium. *The Unicode Standard, Version 4.0*. Addison-Wesley, Reading, MA, USA, 2003. ISBN 0-321-18578-1. xxxviii + 1462 pp. LCCN QA268.U545 2004. URL `http://www.unicode.org/versions/Unicode4.0.0/`. Includes CD-ROM.

[60] J. E. Warnock. The display of characters using gray level sample arrays. *Computer Graphics*, 14 (3):302–307, July 1980. CODEN CGRADI. ISSN 0097-8930.

[61] Stefan Wohlfeil. *On the Pagination of Complex, Book-Like Documents*. Shaker Verlag, Aachen and Maastricht, The Netherlands, 1998. ISBN 3-8265-3304-6. 224 pp. DM 98.00. URL `http://www.shaker.de/Online-Gesamtkatalog/Details.idc?ID=24201&CC=59311&IDSRC=1&ISBN=3-8265-3304-6&Reihe=15`.

TEX and the Interfaces

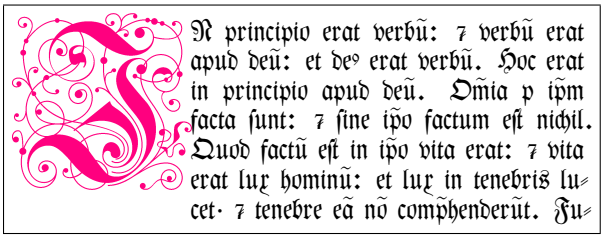
Peter Flynn
Electronic Publishing Unit, University College, Cork, Ireland
pflynn@ucc.ie
http://imbolc.ucc.ie/~pflynn

Abstract

TEX systems have been a cornerstone of research and academic publishing for a long time. Development of the interfaces with different classes of user or potential user, however, has been uneven. Recent developments in other areas of text processing are opening up new opportunities for TEX-based systems. Should TEX development become involved in these areas, or should it be restricted to those areas where it has traditionally been a strong player? This is a summary of my keynote presentation to the Practical TEX 2004 conference in San Francisco.

The cornerstone

Gutenberg's inventions were not Open Source: he worked on them alone, in relative secrecy, for many years before starting to print.

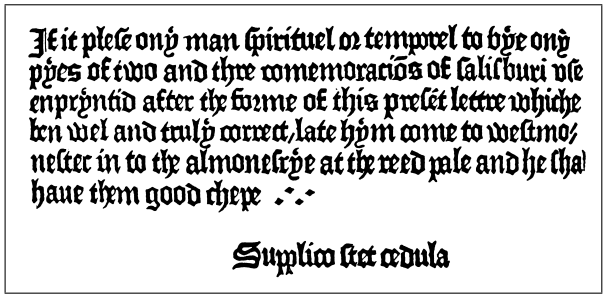

In principio erat verbū: ⁊ verbū erat apud deū: et deꝰ erat verbū. Hoc erat in principio apud deū. Oīa p ipm facta sunt: ⁊ sine ipo factum est nichil. Quod factū est in ipo vita erat: ⁊ vita erat lux hominū: et lux in tenebris lucet· ⁊ tenebre eā nō comphenderūt. Fu-

- He had to get the blessing of the Church, his principal customer, and that meant keeping stumm.
- But he did have partners, and they had to know what he was doing and how he did it.
- He finally screwed up (or was screwed over) and sold out to his backers.
- Eventually someone had to teach the next generation of printers, and the 'secret' was out.

How different from the origins of TEX, where Knuth placed the whole system at the disposal of the world virtually from the start.

Development of the interfaces

If it hadn't been for the spread of the knowledge, Caxton would never have been able to bring the idea of printing from movable type to England, nor Ben Franklin have been able to print in America.


If it plese ony man spirituel or temporel to bye ony pyes of two and thre comemoracions of Salisburi vse enprynted after the forme of this prelet lettre whiche ben wel and truly correct, late hym come to westmo; nester in to the almonesrye at the reed pale and he sha haue them good chepe

Supplico stet cedula

The same holds true for most printing and publishing inventions down the ages — some they tried to keep secret, but in general you can't keep technological inventions from a technically literate and mobile workforce (printers).

In these old printed documents we see the first signs of an interface: between printer, reader, and publisher. Not a technical interface but a moral, social, and business interface.

TEX was explicitly freed from the normal commercial restrictions on software by its author. This was an unusual move in 1978. This was a major contributing factor to its initial success in research labs and academia (no money needed, no license to prevent copying), and also to its successful commercialization.

Printing equips your paragraphs of text with certain features and facilities: dissemination (you can make multiple identical copies), usefulness (people can use your text in different ways), education (literacy and the spread of printing have a well-

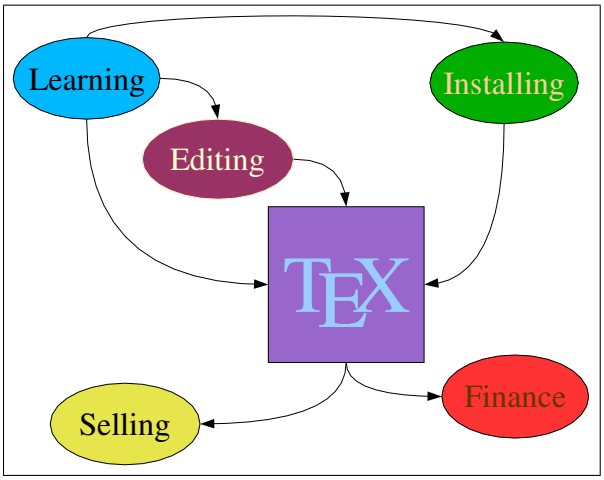
established association), reusability (especially in involving photocopying, scanning, and reprinting!) — and there are many others. The use of an openly-accessible system like TEX equips your pars with other associated benefits:

Extensibility
Quality
Usability
I ndependence
Portability
Persistence
Accuracy
Robustness
Speed

Success is what has made TEX a cornerstone. Much of its success is due to the fact that it keeps on producing the goods, especially when other systems fail.

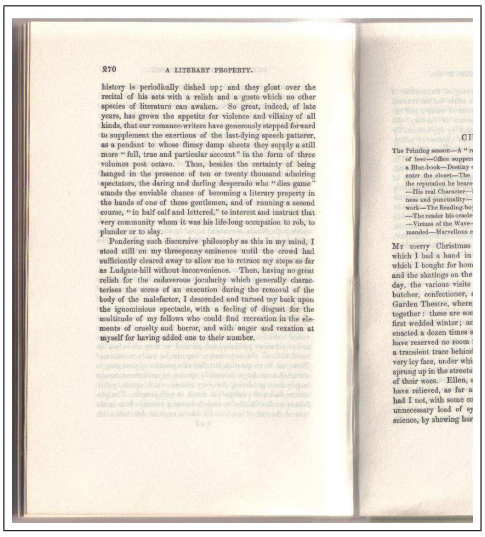
Differences between the interfaces

However, if you show TEX being edited to a publisher, or a business person, or a non-Computer Science, non-Physics, non-Math student, or Marketing, or Sales, or even Management, they'll take one look and laugh.

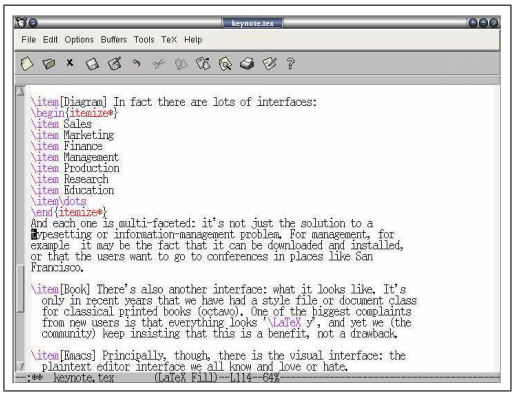


All they see is the physical (editing) interface: and we don't sell TEX as a solution to their problems, we sell it as something else, something akin to a religion.

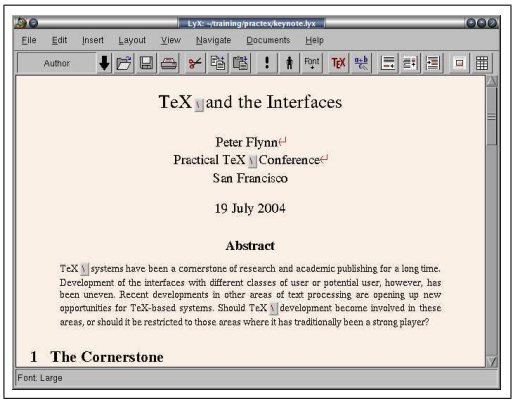
In fact, there are lots of interfaces: Sales, Marketing, Finance, Management, Research, Production, Education ... and each is multi-faceted: it's not just the solution to a typesetting or information-management problem. For management, for example it may be the fact that it can be downloaded and installed without signing a contract.



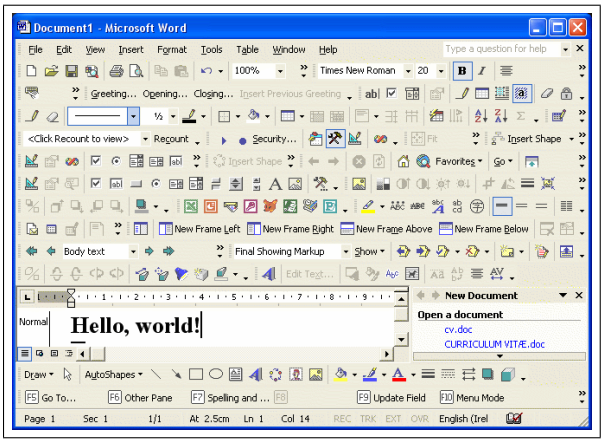
There's also another interface: what it looks like. It's only in recent years that we have had a style file or document class for classical printed books (`octavo`, [3]). One of the biggest complaints from new users is that everything looks 'LATEXy', and yet we (the community) keep insisting that this is a benefit, not a drawback. Principally, though, there is the visual interface: the plain text editor interface we all know and love or hate.



But there are lots of others, including synchronous typographic interfaces like LyX. It's not WYSIWYG but What You See Is What You Meant.



And of course other interfaces we know and hate or love. If you turn on all the facilities LATEX has:



Dissonance

So why are there all these differences? There is a considerable degree of disparity and dissonance between those who want TEX to stay plain text and therefore psychologically inacessible to the user—whose mind-set has largely been conditioned by synchronous typographical interfaces; and those who want a more approachable interface.

There is the 'user-seductive' interface (Microsoft Word, for example) in all its forms (almost any 'office productivity' package, for example) which lets the user 'draw a document instead of writing it' [2].

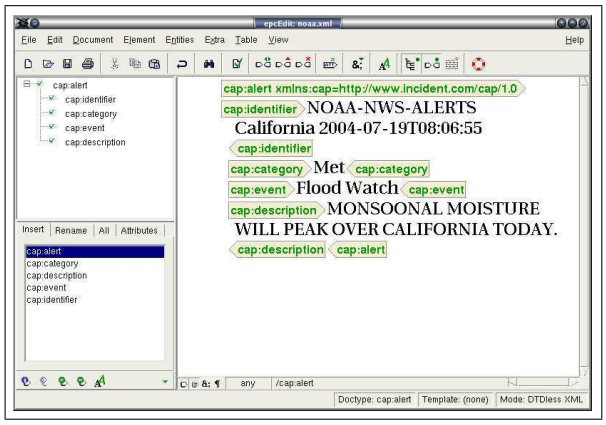
There is the learning interface which I mentioned a few years ago at the TUG meeting in Delaware [1], which ranged from 'sitting by Nellie' to a full-scale two-week training course tailored for your organization.

There is the support interface — fixed by commercial versions, but the free versions still use TUG and `comp.text.tex`, for obvious reasons.

And there is also another kind of interface growing, used for XML. Because of its ubiquity in business and publishing, there is a huge amount of software, and it has many of the features we know from LATEX.

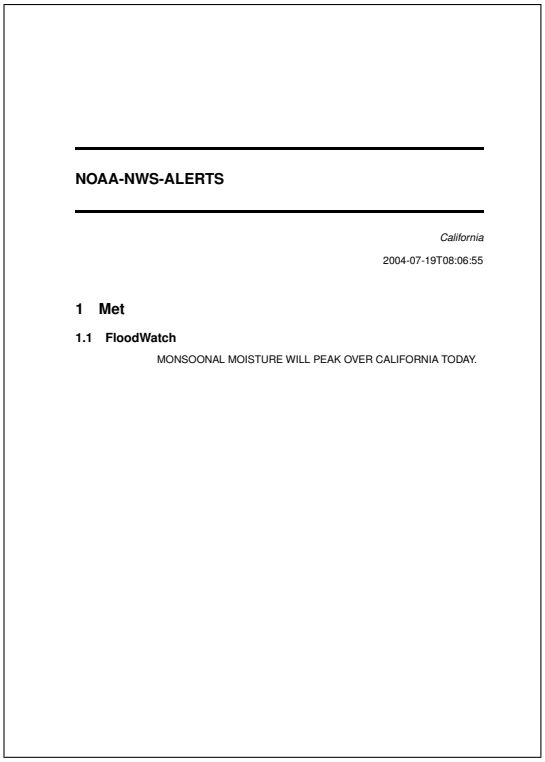
```
<?xml version='1.0' encoding='UTF-8'?>
<cap:alert xmlns:cap=
 'http://www.incident.com/cap/1.0'>
  <cap:identifier>NOAA-NWS-ALERTS
    California 2004-07-19T08:06:55
  </cap:identifier>
  <cap:category>Met</cap:category>
  <cap:event>Flood Watch</cap:event>
  <cap:description>MONSOONAL MOISTURE
    WILL PEAK OVER CALIFORNIA TODAY.
  </cap:description>
</cap:alert>
```

XML has synchronous typographical interfaces too:



I'm reminded of a paper presented at a TUG conference very many years ago, entitled something like 'TEX versus PostScript', as if PostScript were some kind of competitor. There are of course areas where LATEX and XML compete, and probably none more so than in the interface, but it's extremely easy to convert XML to LATEX for output using XSLT. The XSL-FO path to PDF means reinventing the wheel multiple times, whereas LATEX has everything already built in.

The following output was produced from the XML above, using XSLT into LATEX. (The source files `noaa.xml` and `noaa.xsl` are available at `http://silmaril.ie/xml` if you want to try it for yourself.)



Paying attention

So why isn't everyone is paying attention to the interface? They're certainly not ... at least not for documents. Unless we are very careful, we run the risk of turning our primary asset into its primary liability. (I may be preaching to the converted here, but it is the current users who form the interface between the potential user and TEX.) We need more development. If you examine the interfaces in more detail, and start asking questions about new users' expectations, you find some surprising difficulties:

- What do you expect to happen when you press the Enter key?
- Can the B, I, and U buttons capture the reason why you want bold, italics, or underlining?
- Can the font style and size drop-downs be used to capture the reason why you want big bold type at this point?

As I noted earlier, I deliberately authored this in LATEX: I missed the rigour of using XML, but I still haven't found anything to beat TEX and LATEX for formatting. We just need to tell people.

References

[1] Peter Flynn. TEX—A mass-market product? Or just an image in need of a makeover? *TUGboat*, 22(3):137–139, Sept 2001.

[2] Anthony Goreham. Re: Installing a new font: PFM, PFB. `comp.text.tex`, (`m3r8qj42o3.fsf@micawber.queens.ox.ac.uk`), 28 November 2001.

[3] Stefan A. Revets. The octavo package. `http://www.ctan.org/tex-archive/macros/latex/contrib/octavo`.

Note: I am grateful to Prof. Knuth for permission to reproduce the mock-woodcut of a printer's shop experiencing the arrival of TEX, which he used in *Digital Typography*.

Micro-typographic Extensions of pdfTEX in Practice

Hàn Thế Thành
University of Education, Ho Chi Minh City, Vietnam
hanthethanh@myrealbox.com

Abstract

pdfTEX provides two micro-typographic extensions: margin kerning (also known as character protrusion) and font expansion. While they have been available around for quite a long time and the samples showed interesting results, these features have not been much used in practice. The reason is that pdfTEX only provides very low-level support, so in order to use these features, a high degree of TEX knowledge was required. In this article I would like to share some experiences in using these extensions from the user's point of view. Thus, I will not go into any technical detail — there are already papers that have done that job. My wish is that after reading this article, anyone can start using these extensions in practice without much difficulty. Therefore, I will try to look at these extensions from a practical point of view, and focus on things most useful for newcomers.

Introduction

We briefly describe here the concepts of margin kerning and font expansion, collectively called the *hz* extensions, and what purpose they serve.

Margin kerning Margin kerning is the term used for slight shifting of certain characters at the margins so the margins *look* smooth. This technique is sometimes called *hanging punctuation*, as it is useful mostly for punctuation marks such as comma, period and the like. However, margin kerning is a more general concept, as it can be usefully applied to certain letters as well.

Margin kerning in principle is quite similar to general kerning. Kerning is the adjustment of space between certain letters to make the text look good, while margin kerning deals with space between letters and the margins of text. Hence, similar to kerning, margin kerning is also a question of taste: to one person margin kerning makes things look better, to another, it makes no sense. Still, this technique used to be quite common in traditional typography, and its disappearance probably has more to do with its difficult deployment in DTP systems than any change of taste.

Margin kerning is not something completely unknown to TEX users. It is possible to have hanging punctuation (margin kerning applied to punctuation only) in TEX using macros. However, there are certain limitations and problems; for instance, it requires all hanging punctuation marks to be (in TEX terms) active characters, and it doesn't work for the hyphen character. To "hang" the hyphen, a font with a special hyphen character is required. pdfTEX makes use of margin kerning much easier and better.

A sample text with and without margin kerning is shown in figure 1.

Font expansion Font expansion is the technique of expanding or shrinking a font very, very, slightly, in order to break a paragraph into lines in a better way. Of course a font must not be expanded or shrunk too much, otherwise the effect caused by font distortion will spoil everything. Using font expansion can lead to line breaking with:

1. fewer hyphenations,
2. fewer overfull and underfull boxes,
3. more nearly uniform interword spacing (fewer "rivers").

Therefore, font expansion is useful when one wishes to get a more even color of page, or just to reduce the number of hyphenation or overfull/underfull boxes. Such needs are quite common in narrow-column typesetting. When it comes to automated typesetting, such small improvements can significantly reduce the manual work required to correct "problematic" cases.

A sample text with and without margin kerning is shown in figure 2. This entire article is also typeset with margin kerning enabled.

Usability

The micro-typographic extensions of pdfTEX were originally developed for experimental purposes. As a result, the underlying concepts were designed to be general and flexible, so we could examine the effect

A father had two sons, of whom the eldest was clever and bright, and always knew what he was about; but the youngest was stupid, and couldn't learn or understand anything. So much so that those who saw him exclaimed: "What a burden he'll be to his father!" Now when there was anything to be done, the eldest had always to do it; but if something was required later or in the nighttime, and the way led through the churchyard or some such ghostly place, he always replied: "Oh! no, father: nothing will induce me to go there, it makes me shudder!" for

A father had two sons, of whom the eldest was clever and bright, and always knew what he was about; but the youngest was stupid, and couldn't learn or understand anything. So much so that those who saw him exclaimed: "What a burden he'll be to his father!" Now when there was anything to be done, the eldest had always to do it; but if something was required later or in the nighttime, and the way led through the churchyard or some such ghostly place, he always replied: "Oh! no, father: nothing will induce me to go there, it makes me shudder!" for

Figure 1: Text without (left column) and with (right column) margin kerning

of those extensions in many contexts. The drawback, however, is that in order to make use of those extensions, a certain degree of knowledge of TEX programming and font-related issues is required. This is especially true for font expansion, as it required the user to be able to generate so-called expanded TFM files. The lack of an easy user interface also discouraged the average user from trying margin kerning, although this is much easier to use than font expansion.

However, enough people have been interested in testing and using those new features, and hence there has been also some progress on the user interface as well as the implemention:

1. An important step for LATEX users was the package pdfcprot by Carsten Schurig, allowing activation of margin kerning in an easy way.
2. In summer 2004 I added a feature called "auto expansion" for easy use of font expansion to pdfTEX. Now generation of actual expanded TFMs is no longer required, as pdfTEX can expand required TFM's on-the-fly in memory.
3. Not very long after the version with auto expansion had been released, a LATEX package called microtype was created by Robert Schlicht, allowing easy access to both margin kerning and font expansion. Furthermore, this package contains a rich collection of predefined settings of margin kerning for various fonts.

With the microtype package and an up-to-date enough version of pdfTEX, using micro-typographic extensions has become accessible to the average user, without having to deal with low-level commands and messy font issues.

How to begin?

In order to make use of what will be described along, we need two things:

1. pdfTEX version at least 1.20a;[1]
2. the LATEX package microtype, which is available from CTAN.

Instructions on how to upgrade pdfTEX or install a LATEX package onto your system are system-specific and are not covered here. The best place to look or ask for them is probably a mailing list or a forum dedicated to the specific TEX system you are using.

[1] At least pdfTEX 1.20b is the recommended version at the time of writing this article; 1.20a still had some problems with *hz* extensions.

A father had two sons, of whom the eldest was clever and bright, and always knew what he was about; but the youngest was stupid, and couldn't learn or understand anything. So much so that those who saw him exclaimed: "What a burden he'll be to his father!" Now when there was anything to be done, the eldest had always to do it; but if something was required later or in the night-time, and the way led through the churchyard or some such ghostly place, he always replied: "Oh! no, father: nothing will induce me to go there, it makes me shudder!" for

A father had two sons, of whom the eldest was clever and bright, and always knew what he was about; but the youngest was stupid, and couldn't learn or understand anything. So much so that those who saw him exclaimed: "What a burden he'll be to his father!" Now when there was anything to be done, the eldest had always to do it; but if something was required later or in the night-time, and the way led through the churchyard or some such ghostly place, he always replied: "Oh! no, father: nothing will induce me to go there, it makes me shudder!" for he was afraid.

Figure 2: Text without (left column) and with (right column) font expansion

Given that the two above requirements are met, we can already do something practical. A quick test can be done by making a copy of the standard file `sample2e.tex` and insert into the preamble one line, namely:

```
\usepackage{microtype}
```

Let's call the resulting document, with microtype loaded, `sample2ex.tex`. When you run pdfLaTeX on the document, you get a PDF file `sample2ex.pdf` that is *almost* the same as `sample2e.pdf` (the PDF you get without loading microtype). Let us focus on the differences.

1. First, the two files are quite different in size: `sample2ex.pdf` is much larger, because of the font expansion. You can check this if you open it in Acrobat Reader, for instance, and check for embedded fonts, you will see many instances of the same font with the `_Extend` tag appended.[2]
2. In `sample2ex.pdf`, certain characters slightly "protrude" out when at the margins, like the period, comma or double quotes (we don't have any hyphenation in this document to see how the hyphen char would protrude). The effect of such "protrusions" is to achieve the visual effect that the margins look smooth. This is, in short, what margin kerning brings to you.
3. The two files have slightly different line breaks! The line breaking in `sample2ex.pdf` can be considered better, as there are fewer hyphenations than in `sample2e.pdf`. This is, in short, what font expansion brings. The effect of font expansion is more visible when applied to narrow column typesetting; in that case, typesetting without font expansion there results in problems of frequent hyphenations, overfull boxes or rivers.

And that's the essential part of what pdfTEX and microtype offers. If you like it, you can experiment more by loading microtype into some of your own documents, and maybe try a few options of the microtype package. Don't neglect the documentation, as microtype has very nice documentation, with good advice for new users.

How to learn more?

The default settings of the microtype package are reasonable and safe for typical cases (and taste). However, the time may come when you wish to control

[2] Or you can use the tool pdffonts coming from the XPDF distribution. This tool lists embedded fonts in a PDF file.

the *hz* extensions according to your own taste. To this end, it may be useful to:

1. read the microtype documentation and try the options it offers;
2. read about *hz* extensions in the pdfTEX manual;
3. ask on the pdfTEX mailing list for advice.

What about ConTEXt?

I asked Hans Hagen for a short introdution to the ConTEXt interface to *hz* extensions. The following text in this section comes from him.

In ConTEXt, margin kerning as well as font expansion are hooked into the font handling mechanism. This permits users to apply these features to any font and on each abstraction level of the font mechanism. We provide a few examples, to give you an idea of how it's done.

```
\setupfontsynonym [Serif]     [handling=hz]
\setupfontsynonym [SerifBold] [handling=pure]
```

This marks all Serif fonts as candidates for expansion, and all bold serifs for protruding. We can now load the Palatino typeface combination, using a predefined typescript:

```
\usetypescript[palatino][\defaultencoding]
```

These Palatino fonts are enabled by for instance:

```
\setupbodyfont[palatino,10pt]
```

Both mechanisms will only be available when they are turned on:

```
\setupalign[hz,hanging]
```

This demonstrates that both features are also hooked into the alignment handler. They can be disables by 'nohz' and 'nohanging'.

You can finetune expansion with:

```
\setupfonthandling [hz] [min=80,max=80,step=5]
```

In a similar fashion, one can finetune protruding, for instance for specific font shapes or for classes of glyphs. In the previous example we used the protruding alternative tagged as 'pure' but there are more variants.

A quick and dirty approach to enabling both features for all fonts is for instance:

```
\definefonthandling[default][hz,pure]
\usetypescript[palatino][\defaultencoding]
\setupbodyfont[palatino,10pt]
\setupalign[hanging,hz]
```

In the ConTEXt file 'hand-def' you can see what combinations are defined and what parameters can be set. Both features work for all font encodings supported by ConTEXt; defining your own preferences is not that hard and involves no TEX coding. Overloading and inheritance of features is provided.

The most important thing you need to keep in mind is that the font handling you wish to apply must be known to the font before the font is first used. This is a result of the way pdfTEX implements this feature.

How to contribute?

If you use the microtype package and have determined your own settings for a particular font, please send your settings to microtype's author. As well as the easy user interface, the package also offers a collection of settings for various typefaces. The more feedback the author gets, the richer the collection will be and the more pleasurable it will be to use.

Likewise, if you use ConTEXt, please send your feedbacks or suggestions to ConTEXt's author, so other ConTEXt users can share your experiences as well.

TEX4ht: HTML Production

Eitan M. Gurari
Ohio State University
USA
gurari@cse.ohio-state.edu
http://www.cse.ohio-state.edu/~gurari

Abstract

TEX4ht is a highly configurable system for producing hypertext from TEX-based sources. The system is distributed with a large set of configuration files. The most commonly used configurations are those supporting LATEX inputs and HTML, MathML, OpenOffice, and DocBook targets. The first part of the presentation will describe how the system can be used for different applications.

ConTEXt is a new addition to the style files being supported by TEX4ht. The second part of the presentation will describe the work done to provide TEX4ht configurations for ConTEXt, with the objective of offering an insight into the inner working of TEX4ht.

1 From LATEX to Hypertext

Reports authored in LATEX may be converted into hypertext through the TEX4ht system [1]. The system offers an assortment of basic commands for invoking translations to different target mark-up languages, provides switches for requesting predefined variations to the default configurations, and lets the users tailor configurations of their own.

command	output	comment
`htlatex abc`	`abc.html`	HTML, bitmap math
`xhlatex abc`	`abc.html`	XHTML, bitmap math
`mzlatex abc`	`abc.xml`	XHTML, MathML math
`oolatex abc`	`abc.sxw`	OpenOffice XML (uses MathML math)
`dbmlatex abc`	`abc.xml`	DocBook, MathML math

Figure 1: Requests to compile `abc.tex`.

1.1 Basic Translations

To activate a translation relying on a default configuration, one needs just to invoke an appropriate command and provide it with the LATEX file name. Figure 1 lists a few examples. Most users of the TEX4ht system are probably familiar just with the `htlatex` option. However, the `mzlatex` option seems also to be quite popular.

From the perspective of a user, the process is similar to that employed in requesting a standard translation to DVI or PDF. In such cases, typically the translations are requested through a command named `latex` or `pdflatex`, respectively.

HTML devotes very little support to mathematics, providing only simple superscript and subscript elements. Bitmap representations are offered for mathematical expressions to try to address this shortcoming. Such representations are commonly employed as most users are able to view them in their browsers. Yet, bitmap representations are visually inferior with respect to their surrounding text, as they do not scale in size. In addition, non-visual applications can make little use of these representations.

MathML introduces a markup language for expressing mathematics, in a manner compatible with HTML support of regular text. Currently, not many browsers come with built-in support for MathML. Mozilla is an example of a browser which supports MathML. For Microsoft Internet Explorer, an easily installed plug-in program named MathPlayer offers similar capabilities [2]. Stylesheets are also available to render MathML through XSLT and CSS code [3].

1.2 Available Adjustments

The distribution of TEX4ht provides configurations for default behavior, as well as configurations for achieving alternative outcomes. The latter configurations can be requested by referring to their named options through generalized invocation commands of the following form:

This material is based upon work supported by the National Science Foundation under Award No. IIS-0312487. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the National Science Foundation.

```
\documentclass{article}
   \usepackage{makeidx}
   \makeindex
   \title{A Title}
   \author{An Author}
   \date{July 19, 2004}
\begin{document}
   \maketitle  \tableofcontents

   \section{First Section}
      Some text.

   \section{Second Section}
   \subsection{A Subsection}
         Put \index{this}this
         and \cite{bib-1}.
   \subsection{Another Subsection}
         Put \index{this}this
         and \index{that}that
         and \index{one}one,
         \index{two}two,
         \index{three}three.

   \begin{thebibliography}{99}
     \bibitem{bib-1}
        A bib entry.
     \bibitem{bib-2}
        Another bib entry.
   \end{thebibliography}
   \printindex
\end{document}
```

(a)

A Title

An Author
July 19, 2004

Contents

1 First Section
2 Second Section
  2.1 A Subsection
  2.2 Another Subsection

1 First Section

Some text.

2 Second Section

2.1 A Subsection

Put this and [1].

2.2 Another Subsection

Put this and that and one, two, three.

References

[1] A bib entry.
[2] Another bib entry.

Index

one, 1

that, 2
this, 3, 4
three, 5
two, 6

(b)

Figure 2: (a) A LaTeX file `source.tex`. (b) A view of the HTML outcome of '`htlatex source`'.

command-name file-name `"html,`*options*`"`

Figure 2(a) lists an example source LaTeX file `source.tex` which requests standard logical structures, including a title segment, sectioning blocks, table of contents, bibliography, and index. A compilation of this file with the command

```
htlatex source
```

produces the default outcome for HTML code. Figure 2(b) shows a possible rendering of this outcome.

A compilation of the same LaTeX file with

```
htlatex source "html,index=2,3"
```

sets the index in two columns, and partitions the document into web pages based on the sectioning units to a depth of three levels. Figure 3 shows a possible rendering of the different web pages and their hierarchy in a tree structure. The tables of contents enable navigation down the tree levels, and the '`up`' buttons enable navigation in the opposite direction. Navigation between siblings is possible through '`next`' and '`prev`' buttons. For instance, the '`next`' button on the web page of the *Second Section* leads to the web page of the *References*.

A somewhat similar organization of content can be achieved with

```
htlatex source "html,index=2,3,next"
```

Figure 4 shows the result. Here, however, due to the '`next`' option, the '`next`' and '`prev`' navigation buttons assume a different ordering of pages in which the document content is visited sequentially. For instance, under this option the '`next`' button of the root web page leads to the web page of the table of contents. Similarly, the '`next`' button of the web page of *Second Section* leads to the web page of subsection 2.1. On the other hand, the '`next`' button of the web page of subsection 2.2 leads to the web page of the *References*.

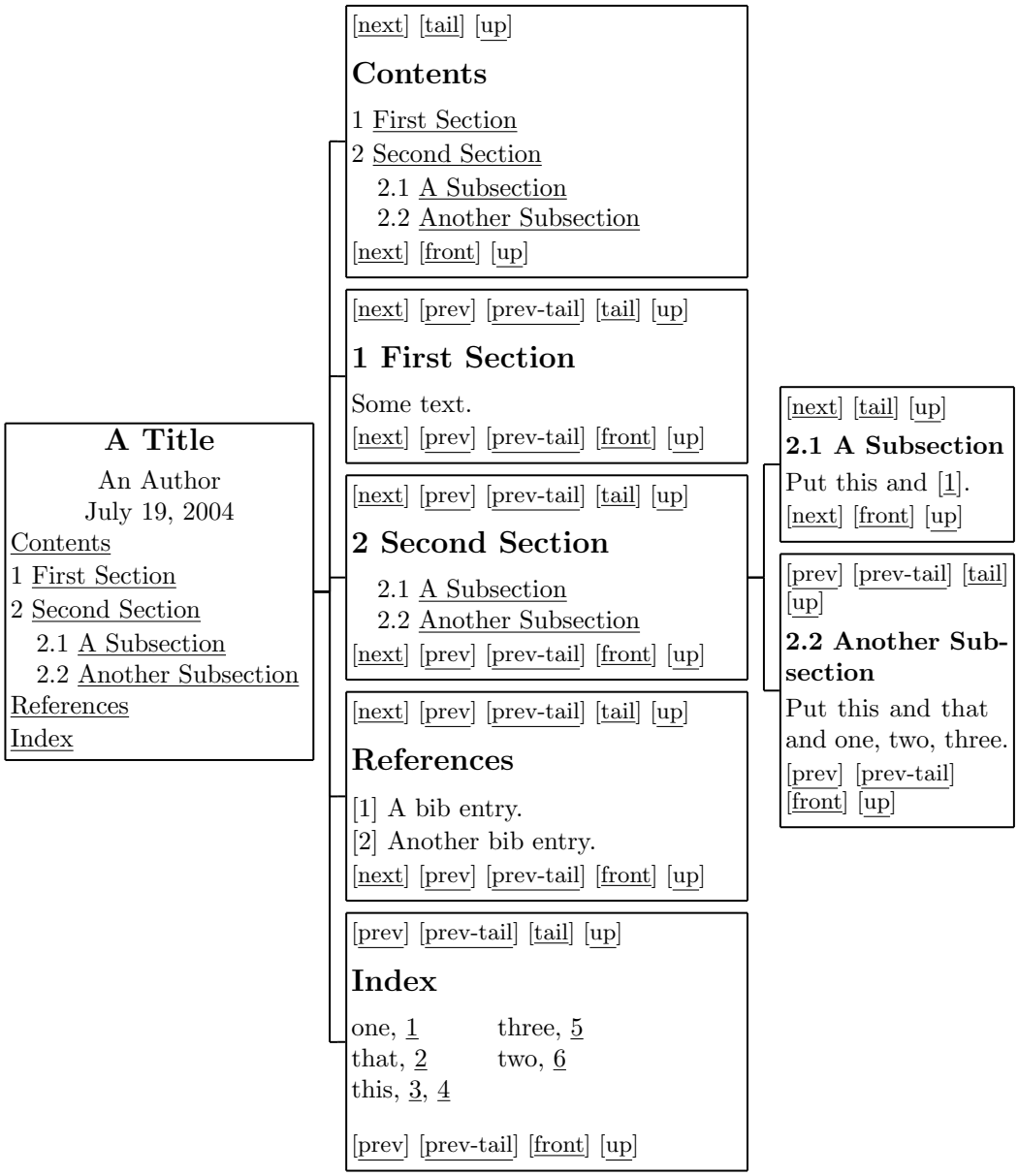


Figure 3: A view of the HTML outcome of '`htlatex source "html,index=2,3"`'. This produces the index in two columns, and separates sections to the third level into their own files.

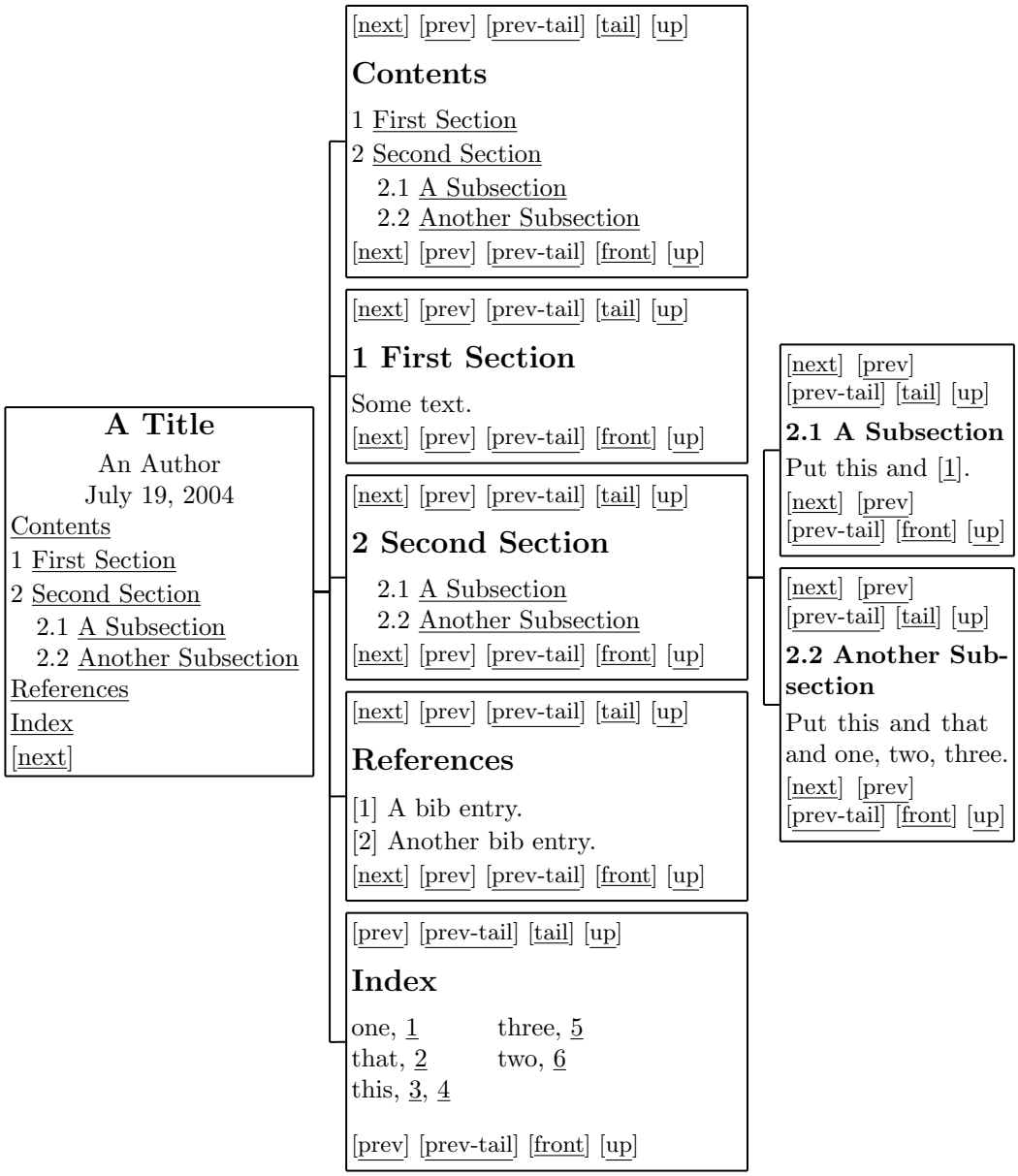


Figure 4: A view of the HTML outcome of '`htlatex source "html,index=2,3,next"`'. Similar to the previous figure, but with sequential navigation, due to the `next` option.

```
/.../texmf/tex/generic/tex4ht/tex4ht.sty
version 2004-05-26-22:19

-- Note -- for automatic sectioning pagination, use
           the command line option '1', '2', or '3'

-- Note -- for i-columns index, use the command line
           option 'index=i' (e.g., index=2)

-- Note -- for linear crosslinks of pages, use the
           command line option 'next'

-- Note -- for inline footnotes use
           command line option 'fn-in'

-- Note -- for content and toc in 2 frames,
           use the command line option 'frames'

-- Note -- For multi-platform MathML through
           stylesheet transforms, use the command
           line option 'pmathml'. If css rendering
           is preferred, use 'pmathml-css'.

-- \TeX4ht{} warning -- If not done so, the index is
to be processed by
  tex '\def\filename{{source}{idx}{4dx}{ind}}
       \input  idxmake.4ht'
  makeindex -o source.ind source.4dx
instead of
  makeindex -o source.ind source.idx
```

Figure 5: TEX4ht messages recorded in the log.

A few other selected options:

- The '`frames`' option may be used to incorporate a table of contents as a navigation bar for the web pages.
- The '`fn-in`' option asks for footnotes at the end of the web pages, instead of being placed at separate pages.
- The '`mouseover`' option requests pop up messages showing content associated with pointers to footnotes and bibliography entries.

1.3 Log Files: A Source of Information

A compilation of a LATEX file `source.tex` produces messages that are recorded in a `source.log` file. Some of these messages, though not all of them, are also listed on the user's terminal. The messages depend on the TEX4ht configurations being activated, and contain useful hints, including the available command line options, version indicators, warnings about possible problems, and information about errors encountered. Figure 5 lists a few examples of the messages obtained in compiling the file of Figure 2(a) with the `mzlatex` command.

The listed command line options '`3`', '`index=2`', '`next`', '`fn-in`', and '`frames`' were considered earlier. The '`pmathml`' and '`pmathml-css`' options refer to the stylesheets of [3]. The warning message indicates how indexes are to be compiled.

```
\documentclass{article}
 \def\greeting{Hi}
\begin{document}
 \greeting{} from \LaTeX{}!
\end{document}
```

(a)

```
\Preamble{html}
\begin{document}
 \def\greeting{Hello}
 \def\LaTeX{}{\TeX4ht{}}
\EndPreamble
```

(b)

Figure 6: (a) A LATEX file `src.tex`. (b) A configuration file `cf.cfg`, changing macros.

LATEX is a system comprised of a very large set of style files, with new styles being added and old ones being modified periodically. Furthermore, there are numerous ways to represent in hypertext the special properties of the style files. The TEX4ht system is quite often updated to address changes in the LATEX environment, users' requests for new features, and errors in the implementation.

1.4 User Configurations

A single LATEX file might be employed by different commands to create a document in an assortment of formats, such as PDF and HTML. Consequently, it is undesirable to explicitly include TEX4ht code in LATEX sources. Commands of the following form can be used to indirectly load configuration files into compilations (the *cfg-file* is the new piece):

command-name file-name "*cfg-file*`,`*options*"

An extension `.cfg` is assumed for a configuration file specified without an extension. The configuration file is loaded into the compilation when the start of the LATEX source body is reached; that is, at '`\begin{document}`'. The configuration file must have a structure compatible with the following template:

`\Preamble{`*options*`}`
configurations before the HTML header
`\begin{document}`
configurations within the HTML header
`\EndPreamble`

For instance, Figure 6(a) lists a LATEX source file whose body is intended to produce the content "Hi from LATEX!". Yet, when compiled with the command

```
htlatex src "cf"
```

```
\begin{itemize}
  \item First item
  \item Second item
\end{itemize}
```

(a)

```
<ul class="itemize1">
   <li class="itemize">
      First item
   </li>
   <li class="itemize">
      Second item
   </li>
</ul>
```

(b)

```
\Preamble{html}
\begin{document}
  \Css { ul.itemize1 {
    color : red ;
    background-color : yellow;
    font-weight: bold ;
    font-size : 150\%
  }}
  \Css { li {
    border : black 1px solid;
    margin : 2em ;
    text-align : center
  }}
\EndPreamble
```

(c)

Figure 7: (a) LaTeX fragment. (b) Corresponding HTML code. (c) Possible CSS configurations.

the outcome is "Hello from TeX4ht!" given the configuration file `cf.cfg` listed in Figure 6(b). This example illustrates, rather dramatically, the idea and potential of configuration files. However, configuration files are typically used to tailor mark-up for the content, not to actually *change* the content!

1.5 Touch-Up With CSS

According to its definition, "Cascading Style Sheets (CSS) is a simple mechanism for adding style (e.g., fonts, colors, spacing) to Web documents" [4]. Accordingly, TeX4ht provides a `\Css{...}` instruction for incorporating CSS code into target files.

Figure 7(a) lists a sample LaTeX source fragment. When compiled for HTML output, the code produced is as listed in Figure 7(b). The configuration file of Figure 7(c) can be introduced into the compilation to associate the given CSS decorations with the HTML code.

1.6 Changing HTML Configurations

TeX4ht indirectly seeds hooks within the LaTeX constructs and associates default configurations with the constructs through the hooks. Users can change these configurations, but typically should do this with a good understanding of LaTeX programming, HTML, and TeX4ht.

Hints as to how the default configurations can be modified may be seen through the '`info`' command line option. The hints are recorded within the log files of the compilations. For instance, the command

```
htlatex source "html,info"
```

shows hints like this within `source.log`:

- Configure environments `\begin{`*name*`}` ... `\end{`*name*`}` with `\ConfigureEnv{`*name*`}{...}{...}{...}{...}`
- Configure lists `\begin{`*name*`}\item` ... `\item` ... `\end{`*name*`}` with `\ConfigureList{`*name*`}{...}{...}{...}{...}`

Figure 8(a) exhibits a possible use of the above instructions for configuring typical LaTeX sources, as shown in Figure 8(b). The outcome is listed in Figure 8(c).

1.7 Beware of Errors: Validate

LaTeX is forgiving of different kinds of misuses of the language. In addition, TeX4ht is not configured for all features of LaTeX and their possible interactions. In contrast, hypertext markup languages impose strict requirements on their use. Consequently, translations are not immune to errors and users are therefore encouraged to validate the output files.

Validators can be invoked via constructs similar to '`.html` *utility* `%%1.html`' in the system environment file `tex4ht.env`.

2 Configuring TeX4ht for ConTeXt

ConTeXt is a macro package offering high-level constructs for expressing logical units of documents [5]. The remainder of this report describes what it took to introduce support for ConTeXt in TeX4ht. The underlying ideas are similar to those employed to support LaTeX.

2.1 Getting Background Information

TeX4ht processes a source file by indirectly modifying the style files in use, invoking the native compiler to translate the source file into DVI code and then processing the DVI output into hypertext markup. In the case of ConTeXt, '`texexec` *filename*' is the basic command to output DVI code.

```
\ConfigureEnv{titlepage}
  {\ifvmode \IgnorePar\fi
   \EndP
   \HCode{<h1>}\IgnorePar }
  {\ifvmode \IgnorePar\fi
   \EndP \HCode{</h1>}}
  {} {}
\ConfigureList{enumerate}
  {\HCode{<div>}}
  {\HCode{</div>}}
  {\HCode{<span
          class="mark">}}
  {\HCode{</span>} }
```

(a)

```
\begin{titlepage}
  Some Title
\end{titlepage}
\begin{enumerate}
  \item First item
  \item Second item
\end{enumerate}
```

(b)

```
<h1> Some Title </h1>
<div><span class="mark">
1. </span> First item
   <span class="mark">
2. </span> Second item </div>
```

(c)

Figure 8: (a) TEX4ht configurations changing the HTML output for the `titlepage` and `enumerate` environments. (b) LATEX source. (c) HTML outcome.

The modification of the style files consisted of indirectly seeding hooks into the files and providing default configurations to the seeds. To achieve this end, simple sample files were needed for experimenting with the features under consideration and learning the issues involved. The files had to be minimal in size and address the different issues in isolation.

Berend de Boer offers a rich assortment of simple source ConTEXt files [6]. These files turned out to be very helpful in the development of TEX4ht support for ConTEXt.

2.2 Proof of Concept

To provide support for a new package, TEX4ht must find a way to indirectly access the different features introduced by the package. The first challenge was to determine whether TEX4ht can deal with the simplest ConTEXt source files.

```
\starttext
Hello world.
\stoptext
```

(a)

```
\input tex4ht.sty
\Preamble{xhtml}
\EndPreamble
\starttext
Hello world.
\stoptext
```

(b)

Figure 9: (a) The simplest ConTEXt file. (b) Explicit request for TEX4ht configuration within the ConTEXt file.

To answer this question, the `hello.tex` source of Figure 9(a) was compiled for DVI output with the command '`texexec hello`'. The successful compilation ensured that ConTEXt installed correctly and that the source file was correct. The next stage consisted of creating a similar file `hello4ht.tex` that explicitly loaded the core TEX4ht configurations into the compilation. This modified file is shown in Figure 9(b).

The modified file was similarly compiled with the command '`texexec hello4ht`' to produce DVI output. Then the sequence of commands '`tex4ht hello4ht`' and '`t4ht hello4ht`' post-processed the DVI output into HTML format. The compilation into the DVI target complained along the way about a few errors. Similarly, the post-processing created an imperfect HTML file, with extra text scattered around.

The above problems called for a few corrections to the core TEX4ht configurations. In addition, they required the tailoring of a nucleus of a TEX4ht configuration file `context.4ht` for ConTEXt.

The configuration file incrementally grew in size as the different features of ConTEXt were treated for TEX4ht support. Eventually, all the HTML code was transferred into a configuration file `html4.4ht` dedicated to handling HTML code, and `context.4ht` contained just the code for seeding ConTEXt hooks.

2.3 Setting an Invocation Script

A desirable objective of TEX4ht is to leave the user source file and the ConTEXt style files untouched. In the case of ConTEXt, a new `htcontext` command was introduced to invoke the following script.

```
\def\complexstartsmaller[#1]%
  {\par \bgroup ...
   \advance\leftskip ...
   \advance\rightskip ...}
\def\stopsmaller{\par \egroup}
```

(a)

```
\let\o:complexstartsmaller: =
          \complexstartsmaller
\def\complexstartsmaller[#1]{%
   \o:complexstartsmaller:[#1]%
   \a:narrower\bgroup
   \aftergroup\b:narrower
   \aftergroup\egroup }
\NewConfigure{narrower}{2}
```

(b)

```
\Configure{narrower}
   {\ifvmode\IgnorePar\fi \EndP \HCode{<div class="narrower">}}
   {\ifvmode\IgnorePar\fi \EndP \HCode{</div>}}
\Css{div.narrower {margin-left:2em; margin-right:2em;}}
```

(c)

Figure 10: (a) ConTEXt's `\complexstartsmaller` macro. (b) TEX4ht hooks. (c) HTML configuration.

```
texexec \
  --arg="opt-arg=configuration-options" \
  --use=tex4ht ConTEXt-options filename
tex4ht filename tex4ht-options
t4ht filename t4ht-options
```

The `texexec` command line loads the TEX4ht configurations, including the file `context.4ht`, at the `\starttext` instruction of the source file. The `\starttext` instruction marks the end of the preamble of the document and the start of the body. Consequently, the TEX4ht configurations have the last word on how the environment will look for the compilations into DVI.

2.4 Planting and Configuring Hooks

Planting hooks indirectly into a package's macros requires a deep understanding of the implementation of the macros. For many features, acquiring such knowledge is not an easy task. Experiments with simple source files that use these features can provide very helpful hints. Still, the job is often tedious and time consuming.

Figure 10(b) illustrates how TEX4ht hooks are indirectly introduced, within the `context.4ht` file, into the ConTEXt macro `\complexstartsmaller`. This macro is defined in the style file `core-spa.tex` of ConTEXt — its outline is shown in Figure 10(a). The implementation takes advantage of having the `context.4ht` file loaded at the `\starttext` instruction, while `core-spa.tex` is loaded earlier.

Figure 10(c) shows the HTML configurations to be associated with the hooks in the default setting.

2.5 Observations

ConTEXt is a TEX environment very rich in features. The work described in this report relates to the core ConTEXt features discussed in [6]. Additional configurations will be provided in response to requests from users of the system.

The following are a few of the hardships encountered in preparing TEX4ht configurations for ConTEXt.

- ConTEXt is written in Dutch. Not knowing the language makes it difficult to follow the meaning of commands.
- Having a limited understanding of ConTEXt, too much time was spent on brute force experimentations and tracing of computations.
- General purpose environments such as '`\begin{`*env*`} ...\end{`*env*`}`' in LATEX are very rewarding. They require very few hooks and cover large sets of commands. ConTEXt offers similar environments through hidden definitions to macros of the form '`\??`*env*'.
- Lack of a clear semantics makes it difficult to provide intelligent configurations (this seemed to be the case for enumerated versus description lists).
- Hooks at different levels of grouping make it difficult to communicate information between the hooks. For instance, the `\@@somedefinitie` macro apparently forces this type of approach.

The ConTEXt system has been created to help produce good looking documents with well-specified page formats, often in PDF format. In this respect it has achieved outstanding results. Hypertext seems

to offer a large array of additional opportunities for this system.

2.6 Acknowledgment

I am grateful to Bob Kerstetter for initially requesting TEX4ht configurations for ConTEXt and for arranging help to get me started with ConTEXt. I am indebted to Patrick Gundlach for his considerable effort to install ConTEXt on my platform. I would like to thank Hans Hagen for his input, and Karl Berry for editing the report.

References

[1] Eitan M. Gurari, *TEX4ht: LATEX and TEX for Hypertext*, http://www.cse.ohio-state.edu/~gurari/TeX4ht/.

[2] Design Science, *MathPlayer*, http://www.dessci.com/en/products/mathplayer/default.htm.

[3] David Carlisle, *XSLT stylesheets for MathML*, http://www.w3.org/Math/XSL/Overview-tech.html.

[4] *Cascading Style Sheets (CSS)*, http://www.w3.org/Style/CSS/.

[5] Ton Otten and Hans Hagen, *ConTEXt: An excursion*, Pragma Ade, http://pragma-ade.nl/general/manuals/mp-cb-en.pdf.

[6] Berend de Boer, *LATEX in proper ConTEXt*, http://www.berenddeboer.net/tex/LaTeX2ConTeXt.pdf, July 2003.

The State of ConTEXt

Hans Hagen
Pragma ADE, The Netherlands
pragma@wxs.nl

Abstract

In this article I will describe the current state of the ConTEXt macro package and the forces that play a role in its evolution. I will also indicate the directions in which we look for further developments.

1 ConTEXt developments

The public part of the ConTEXt story started around 1995. If we summarize the main developments in this macro package we can roughly identify the following points of focus:

- a configurable environment where users can define styles, using an interface in a language of choice; the multilingual interface was first needed when the chemical package PPCHTEX was adapted to English and made generic
- support for document collections such as we find in educational settings, with a focus on re-usability; multilingual support, selective processing and dedicated modules for chemical formulas and consistent usage of physical units evolved from there
- features aimed at highly interactive documents, optimized for reading on computer screens; support for one source, multiple output was part of that
- extensive support for grid snapping combined with advanced multi-column typesetting
- typescripts as a means of building font collections and combining typefaces in many (possibly weird) ways
- all kinds of fuzzy configuration options needed in order to mimic the behavior of desktop publishing applications
- integrated support for processing XML documents and using XML databases

Extending and improving ConTEXt have never been related to strong versioning or promises for successors. Part of the game is that we try to remain downward compatible. And so, officially, we still have ConTEXt version 1. Successive releases are tagged by date.

The most recent change was not so much related to new features but more to the machinery behind the screens. Those who have looked into the source code probably have noticed that for reasons we will not discuss now, keywords and variable names look rather Dutch, that is, until recently. Around August 2004 we made the move to a low level English interface. Although we had some help from a Perl script that had been written for this purpose years ago, still quite some manual checking had to be done.

This does not mean that ConTEXt is completely clean under the hood. When we started developing the system, TEX's were small, and so we ended up with quite some dirty (not that verbose) code. One can easily recognize the older code, but we hope to weed out the ugly bits in due time.

There is good reason to qualify the current version as ConTEXt version 2. The reason for this is that users who use low level Dutch keyword constants (prefixed by `\v!` and `\c!`) in their non-Dutch styles, now need to translate these into English. A bonus is that third party extensions will be easier to implement. Such developments will further be stimulated by Taco Hoekwater's ConTEXt API project and Patrick Gundlach's ConTEXt interface description project hosted at `contextgarden.net`. I must admit that the decision to go low-level-English now and not later, was triggered by their initiatives.

Of course one can legitimately ask whether there is still need for further developments in TEX macro packages like ConTEXt. At Pragma ADE we deal with documents coded in TEX as well as the more avant-garde XML format. It cannot be denied that XML coding makes documents much less error-prone: it's much simpler to check the syntax

Editor's Note: This article is reprinted from MAPS 31 by kind permission of the author and editor. The author made several related presentations at the Practical TEX 2004 conference.

of an XML file than of a TEX file. However, it can also not be denied that the loss of typographic detail (or more precisely: the means of authors to improve the look and feel of the final result) is a high price to pay. Of course there are also document types that cannot easily be covered by a manageable set of XML elements. Just think of highly complex math, physics or chemistry, or manuals that use a wide range of visualisations. One easily ends up with something that, although coded without backslashes, looks rather familiar to TEX users.

An even more important observation is that whatever means of going from document source to typeset product we choose, the visualisation problem will not change. No matter how many tools (or macros) one writes, differences in designs (and not seldom inconsistencies in designs) demand unique solutions. Although one can easily become overwhelmed by the possibilities that today's publishing tools provide, there is still a place for the proven TEX technology.

2 ConTEXt and browsers

Recently we redesigned the Pragma ADE web-site, a site that is mostly dedicated to ConTEXt. The HTML pages are generated from XML sources using XSLTPROC. Some of the PDF documents are generated from the same XML code. In addition, we generate templates and interfaces for the eXAMPLE framework, of which we now run an instance on the web site. This framework is a shell around TEX and friends, and provides features like page imposition and font tests, wrapped in an interface, but very TEXish underneath.

When rebuilding the web site, it was enlightening to find out that standards like CSS were not always precisely supported, even after being around for many years. Firefox (Mozilla Gecko engine) does a decent job. But for Internet Explorer, we have to cheat dimensions and use dirty tricks to get the alignment right. Opera was not that bad, but could not handle relative dimensions well. In the end we had to follow yet another approach to make Apple's Safari Browser (based on the KDE engine) happy as well. One lesson that I learned here was that even an abundance of implementations (or renderers) and tons of documentation (it's easy to find info in the web on CSS and HTML) makes defining a simple layout a painful and time consuming process. It's also interesting to see that the amount of XSLT code needed is not necessarily smaller than the ConTEXt code needed to generate similar output in PDF. Although the TEX community is under pressure of evolving techniques, it should also realize that its huge repository of tools and macros is not that bad after all.

Interestingly, browsers can handle complex operations, like displaying Arab or Chinese and handling widgets and JavaScript quite well, but setting up a simple geometry based on fractions (percentages of the screen size) goes beyond their capacity. Something similar can be observed with the XML related CSS cousin XSL-FO: I still have to run into a nicely typeset book done that way with a better than mediocre design. Again the focus seems to be more on the machinery around it, than on the creation of masterpieces. But then, this may well be beyond its purpose. Whatever a TEX user may think of CSS compared to his or her favorite macro package, its influence is undeniable. The evolution of the Mozilla platform demonstrates this: it provides a user interface builder based on CSS and XHTML called XUL. When PDF came around, I made some documents that could be considered to be programs. It looks like in the end typesetting and user interfacing finally meet each other.

3 Future developments

The majority of documents is a collection of paragraphs of running text, itemized lists, a few graphics here and there, and a couple of tables. TEX and TEX-related packages can handle such documents with ease. However, it seems that even in automated work-flows, where most of the interface can be hidden, TEX is seldom considered to be an option. But, when no other alternative is available, or when other applications failed to perform, this 25 year old program can come to help. It's interesting to observe that the TEX community can still attract new users who don't consider the user interface too much of a problem. So it definitely makes sense to continue development, if only because there is still a large group of documents that demand such tools and typographical detail. As long as TEX can keep up, the ConTEXt story will continue and we will see version 4 (extremely modularized), version 8, 16 and maybe 32 some time in the future. In the end it may be that properly typeset documents where time and effort is put in the look and feel, become a niche, and make way for documents with a minimum of design that can be generated each time they are updated, using the user's preferences.

What is currently happening at the ConTEXt frontier? ConTEXt has been ε-TEX aware for a long

time, and the PDFTEX engine is supported quite well. The good news is that PDFTEX is still under active development. For instance large parts of the font handling were redesigned, and paragraph optimization (PDFTEX implements a font expansion algorithm akin to the HZ micro-typography algorithm by Prof. Hermann Zapf) as well as protruding (hanging punctuation) have become more user friendly. CONTEXT supports both mechanisms quite well.

With the arrival of Aleph, the stable descendant of Omega, support for this extension will become more visible than it was so far. Although UTF is supported, as well as some specialized Chinese encodings, using Aleph will bring Unicode support in the broadest sense, given that adequate fonts and hyphenation patterns become available. In many aspects Omega is not as multilingual as advertised, and certainly not by nature. Omega and therefore Aleph provide some mechanisms, but one still needs macros on top of these to tie the directional typesetting to actual languages and layout. Taking fonts as a starting point, the Mac OS X specific unicoded TEX variant XETEX also looks promising. According to one of the CONTEXT Mac OS X experts, Adam Lindsay, hardly any extensions to CONTEXT are needed in order to get documents typeset in virtually every script.

Other developments that may become of interest are Taco Hoekwater's merge of TEX and METAPOST. There CONTEXT will not only benefit from a speedup due to more efficient inter-process communication, but it may also open new worlds. The average user will probably not use CONTEXT the way we do, for instance to create DTP like output from XML sources, which often means multiple calls to METAPOST per page. Think of documents with 250–500 pages, hundreds of (possibly run time manipulated) graphics, thousands of calls to METAPOST, with an occasional size of over 500 megabytes, and you can imagine that any speed improvement counts. Most features that we use in projects end up in the kernel, and so many users may profit from an efficient integration.

I already mentioned XML. In the next couple of years, more CONTEXT subsystems will use this format in one way or another. If you take a closer look at the distribution, you will notice that quite some XML objects are present already, like in the figure database mechanism and other tools. New is FOXET, yet another XSL-FO engine. Formatting Objects (FO's) are a kind of building block to be handled by a typesetting engine.

Although FOXET ended up on our agenda due to some vague promises made long ago, the actual development of FOXET was triggered by the observation that the CONTEXT MathML engine is being used to fill in the gap in commercial engines. Why bother making small bitmaps (or PDF snippets) of formulas while TEX can do the whole thing? It is interesting to notice that most of the documents that this applies to are rather trivial to typeset with either CONTEXT built-in XML features or by using XSLT to generate intermediate TEX code. It is also interesting to observe that there are CONTEXT users who use XML documents with CONTEXT as a backend, thereby hiding TEX completely.

The magic sound of XSL-FO occasionally makes our customers express the wish for an engine that can handle them (even if their designs are not that well suited for it). Somehow the magic obscures the fact that it's a relatively slow process, that it may take longer to implement (as said before: the problem does not change), does not necessarily lead to well typeset documents, et cetera. If one knows that something is possible (and with TEX much is possible) the demands of designers are seldom adapted. When something is not possible at all (and this occurs with XSL-FO) my guess is that the demands will be dropped. Float handling and marginal notes are examples of areas where TEX is hard to beat.

4 Paragraph building

So what about TEX's superior paragraph builder? Unfortunately most of the documents that we have to typeset professionally are designed by those who use DTP systems with poor quality paragraph builders. This means that they simply cannot believe that there are programs that can do a decent job. As a result we end up with colorful and abundantly illustrated documents that have rather complex layouts (especially if you take into account that they are typeset automatically) but with poorly typeset paragraphs, and that is what they recognize. It is hard to explain that by setting all TEX's penalties to their maximum, the solution space becomes pretty small. Even the somehow always demanded ragged right justification then looks plain bad. The problem for the TEX community is that alternatives for TEX don't have to provide TEX quality paragraph routines. As long as they can get the layout done, they win the game. CONTEXT users who like to look into the source will have noticed that quite some control was added in order to meet these demands, even to the extent that it may lower the quality.

So what good is it for TEX users? As with many things, it's no bad idea to take the best of all worlds. There is nothing wrong with DTP, and for many applications, an Office Suite does well. And for a certain range of documents XSL-FO is a good choice. Of course it remains puzzling why some of today's publishing on demand workflows are presented as something new, while in practice it already could have be done that way for decades using SGML and TEX, at far lower costs too. In some sense TEX was simply too far ahead.

One can mix those techniques. Just as one makes a graphic in a drawing program, one can imagine embedding a one page document coded in XSL-FO as a graphic in a TEX document. In this way we get a kind of 'placed XML'. And ConTEXt already can happily combine TEX and XML in such ways. Also, it's more convenient to store information in a standardized (XML) format, than to invent some syntax for each situation and develop different tools for each of them. For instance, if we want file information in our documents, we use `xmltools` to generate a directory database (this can be done at document processing time by using a system call) and we then let TEX filter the information from that database. Another example is OpenOffice. Anyone who has taken a closer look at this program will probably have recognized similarities with XSL-FO related developments. Seeing TEX as an alternative back-end for texts edited in that environment is not such a bad idea.

All these worlds can meet each other in ConTEXt. In ConTEXt, TEX and XML come together not only in foXet, but also in what we've called 'The Example Framework'. The eXaMpLe logo has the x, m, and l hidden inside, but the actual purpose of this project is to hide TEX from users. On our web site you can play with some of these framework features.

It will be clear that the future of ConTEXt is to some extent related to the advance of XML, although the pure TEX approach will not be neglected. For many documents the TEX syntax (or in our case, the ConTEXt one) is quite well suited and efficient. Although I nowadays code most database related documents in XML (like the PDF showcase document interfaces) I have no plans to abandon TEX. Even thinking of coding a manual like the one about MetaFun in XML already gives reasons for nightmares. And so... plenty of ConTEXt ahead.

A Simple Book Design in ConTEXt

Steve Grathwohl
Duke University Press, Durham, NC, USA
sgrathwohl@dukeupress.edu

Abstract

As a test of book design implementation in ConTEXt, I report on a design for *A Voyage to Arcturus* by David Lindsay, including page setups, chapter heads, headers, and typescripts for fonts.

1 Motivation

Whenever I learn a new TEX system, I try to implement a design for a significant number of pages. Many years ago, when I was learning LATEX (2.09), I wrote a `rawls.sty` to mimic Harvard Press's design for John Rawls's *A Theory of Justice* [3], a design that featured a number of interesting features: not only were there parts, chapters, and sections, but the sections were numbered consecutively throughout the book, orthogonally to the chapters. So the first section of Chapter 3, say, might be numbered section 16. Page headers had rules under the header texts. I mention all this because in those pre-LATEX2$_\varepsilon$ days, it was far from trivial to make substantive changes to the default styles. I remember studying Don Knuth's (plain) code for his Computer Journal article on Literate Programming and thinking what a nightmare it would be to implement in LATEX; but that was 1988. Matters have certainly improved since then.

When I first encountered ConTEXt I was immediately impressed by the `setups` mechanism of key/value pairs approach to a design interface. I began using ConTEXt for typesetting internal documentation here at the Duke Press (coded in DocBook XML and processed using Simon Pepping's *Docbook In ConTeXt* [2]). But I had in mind all along trying out ConTEXt in a larger project. I wanted to see how easy it would be to render a book design compared to LATEX. I suspected it would be much easier; I was right.

2 The Text

When I discovered that a very strange book I had first read as a youth, *A Voyage to Arcturus* by David Lindsay [1], had been deposited in Project Gutenberg, I knew I had my text. In the event, the OCR text was quite corrupt, and it took a while to make the necessary edits to bring it to an acceptable standard.

The design I had in mind for the book was based on a mathematics text I read in college. The unifying theme was a vertical rule separating visual elements of the chapter headings and page headers.

3 Fonts

I decided to use a Bembo clone (called Bergamo) for the text and an Optima clone (called Opus) for the chapter headings and header texts. Both are from the FontSite 500 collection [4]. To use these fonts with ConTEXt, I write some typescripts.

```
\starttypescript [serif] [bergamo] [ec]
  \definefontsynonym [Bergamo-Roman]
      [5borjx8t] [encoding=ec]
  \definefontsynonym [Bergamo-Bold]
      [5bobjx8t] [encoding=ec]
  \definefontsynonym [Bergamo-Italic]
      [5borix8t] [encoding=ec]
  \definefontsynonym [Bergamo-Bold-Italic]
      [5bobix8t] [encoding=ec]
  \definefontsynonym [Bergamo-Caps]
      [5borcj8t] [encoding=ec]
  \definefontsynonym [Bergamo-Bold-Caps]
      [5bobcj8t] [encoding=ec]
\stoptypescript
```

Observant readers who know the Berry naming conventions will see that Bergamo contains both full 'f' ligatures and old-style numerals.

In the following I declare that maths be in scaled Palatino (even though in this project there are no maths). I find that Palatino for maths blends well with Bergamo, and I wanted to go ahead and set this up for future projects.

```
\starttypescript [Bergamo]
  \definetypeface [Bergamo] [rm] [serif]
      [bergamo] [default] [encoding=ec]
  \definetypeface [Bergamo] [ss] [sans]
      [opus]    [default] [encoding=ec]
  \definetypeface [Bergamo] [tt] [mono]
      [modern]   [default]
  \definetypeface [Bergamo] [mm] [math]
      [palatino] [default] [encoding=ec,
           rscale=.90]
\stoptypescript
```

The code for Opus is similar. I store these typescripts in `type-fontsite.tex` and invoke them. Note that I use hanging punctuation and open up the lines to improve readability.

```
% Set up hanging punctuation, pure style;
% Declare Berry naming conventions, ec
% encoding

\usetypescript[serif] [hanging] [pure]
\usetypescript[berry] [ec]

% Load Bergamo and Opus fonts,

% declare sizes and leading.
% Looks better with lines opened a bit.

\usetypescriptfile[type-fontsite]
\usetypescript[Bergamo]
\setupbodyfont[Bergamo,10pt]
\setupinterlinespace[line=1.35em]

\setupalign[hanging]
```

4 Chapter Heads, Page Headers and Footers

I set up the heads with these options

```
\setuphead
  [chapter]
  [page=yes,
   before={\blank[force,4*line]},
   after={\blank[4*line]},
   command=\mychap]
```

Note the `command` option. This allows me to design my own chapter head appearance. `\mychap` looks like this (`#1` refers to the chapter number, and `#2` refers to the chapter title):

```
% the % after ] and } suppresses space

\def\mychap#1#2%
  {\hbox to \hsize \bgroup
     \hfill
     \setupframed
       [offset=0.5em,frame=off]%
     \tbox
       {\framed
         [width=2cm,align=left]
         {\ss #1}}%
     % now instructions for #2,
     % anything but ragged right with
     % no hyphenation looks bad
     \tbox
       {\framed
          [width=.5\textwidth,
           align=flushright,
           leftframe=on]
          {\hyphenpenalty 10000 \ss #2}}%
   \egroup}
```

I want dropped caps for my chapter openers, and small caps afterwards for a certain number of words that I choose. (It is also possible to set this up so the entire first line is in small caps automatically; but I prefer to choose my own breaks.) The dropped cap will be in Opus, be 3 `\baselineskip`s tall, be dropped one line, and have 2 points of padding.

```
\def\Drop {\DroppedCaps
  {} {Sans} {3\baselineskip}
  {2pt} {1\baselineskip} {2}}

\def\chap#1/#2/{\Drop #1{\sc#2}}
```

so I can say

```
\chapter{The S\'eance}

\chap O/n a march evening/, at eight
    o'clock, Backhouse, the
```

You can see the result in Figure 1.

To unify the design, I make the headlines mirror the chapter openers, with a vertical rule separating verso the page number and book title and recto the chapter title and page number, all in Opus. First I declare doublesided pages and turn off auto page-number placement. Then I specify a different scheme for chapter opening pages.

```
% Remove auto page numbering placement;
% I'll do it manually.

\setuppagenumbering
  [alternative=doublesided,
   location=]

% Set up header texts, recto and verso

\setupheadertexts
  [] [\setups{text:header:1}]
  [\setups{text:header:2}] []

\startsetups text:header:1
  \getmarking[chapter][current]
  \quad\vrule\quad
  \pagenumber
\stopsetups

\startsetups text:header:2
  \pagenumber
  \quad\vrule\quad
  A Voyage to Arcturus
\stopsetups

% Define heads for chapter opening pages

\definetext
  [chapterstart]
  [footer]
  [pagenumber]

\setuphead
  [chapter]
  [header=empty,
   footer=chapterstart]

\setupheader
  [style=\ss]
```

A page spread can be seen in Figure 2.

Now I specify the Table of Contents:

```
% Set up table of contents format.
% Move whole operation to the right
% to better center the TOC, and make
% sure chap numbers align properly
% (flushright) in their own box

\definelist
  [chapter]

\setuplist
  [chapter]
  [alternative=a,
   margin=.2\textwidth,
   numbercommand=\NumCom]

\def\NumCom#1{\hbox to 2em{\hfill #1}}
```

5 Setting Up the Pages

Last (actually first) I set up the pages and a switch for page imposition. Pay attention to the commented lines for crop marks, etc.

```
% Set the sizes.

\definepapersize
  [arc]
  [height=220mm,
   width=145mm]

\setuppapersize
  [arc]
  [letter]

% Set up arrangements for printing as
% booklet. Toggle as needed.

% \setuparranging[2UP,rotated,doublesided]

\setuplayout
  [margin=0pt,
   width=middle]

\setuplayout
  [topspace=2\baselineskip,
   height=middle]

% Layout modifications to headers, etc

\setuplayout
  [header=2\baselineskip,
   footer=2\baselineskip,
   location=middle]

% Crop marks.
%
% \setuplayout[marking=on]

\setupindenting
  [medium]
```

```
% I guess Bush would call this
% 'freedomspacing'

\setuplanguage
  [en]
  [spacing=broad] % french spacing
```

Finally, for output targeted for a computer screen instead of print, I can say

```
\setuppapersize[S6][S6]
\setupinteraction[state=start]
```

I can't argue emphatically enough for this approach to books and articles destined for a computer screen. The advantages to making one's way through the text by just the touch of the space bar are, to me, self evident.

6 Future Work

Clearly, implementing a simple design in ConTEXt is quite straightforward. In fact, the advantages of using ConTEXt become more obvious the more complicated the document design. I hope that this article might motivate others to give ConTEXt a try for their own typesetting projects.

Eventually I plan to code the book in XML along with supporting files for browser display and direct typesetting with ConTEXt. For the moment, I will post the screen version at `http://www.duke.edu/~grath/arcS6.pdf`, after a friend designs a suitable cover page for it. Other versions will follow when ready.

But be warned—many have found Lindsay's philosophy detestable (a worship of suffering is one characteristic of it). The English writer C. S. Lewis certainly found it so, even if the book did influence his wonderful space novels.

Acknowledgments. Thanks are due to Hans Hagen for improving my humble code in places and for writing the ConTEXt *TUGboat* style.

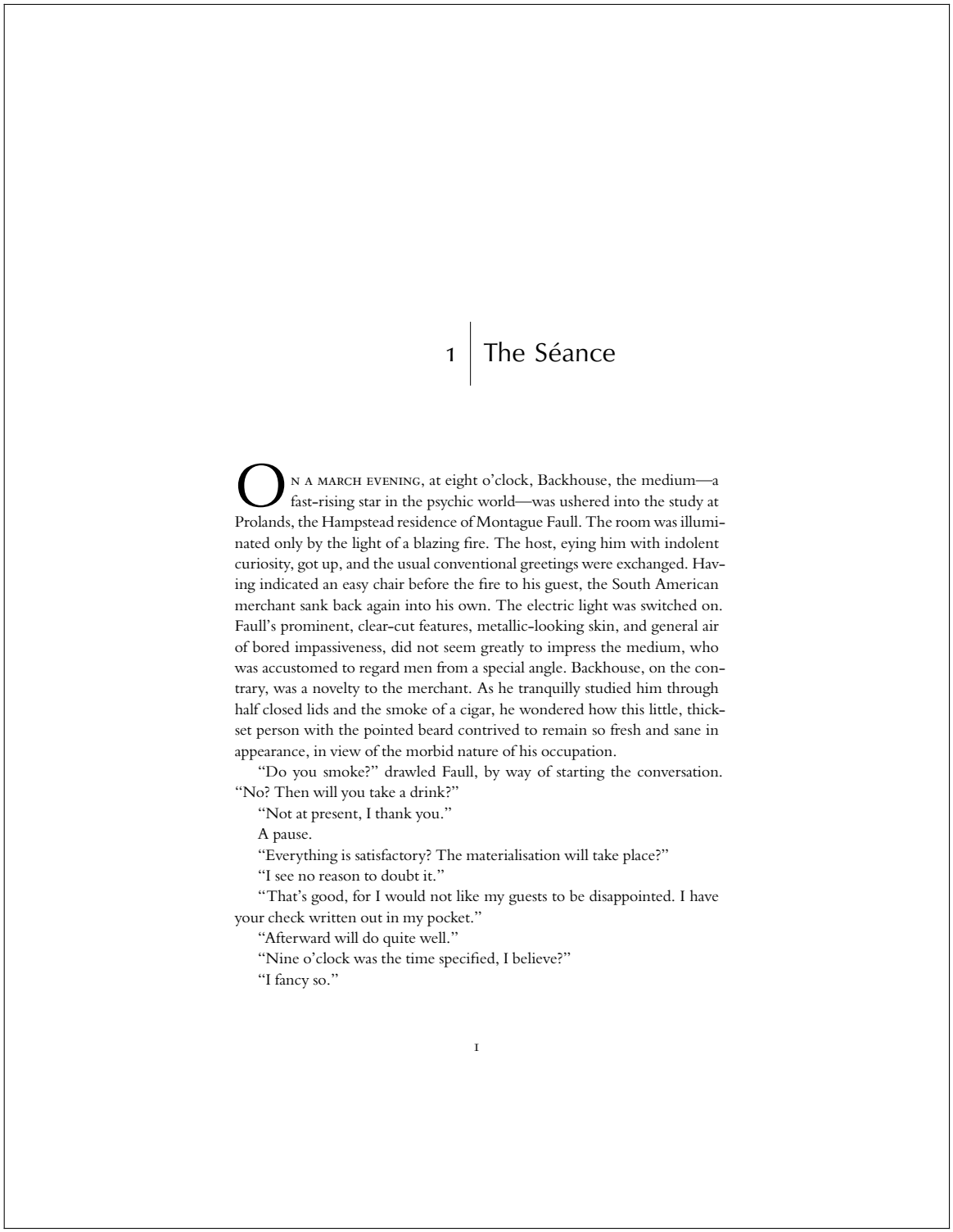
References

[1] Lindsay, David, *A Voyage to Arcturus*, London, Methuen, 1920. Text available at Project Gutenberg: `http://www.gutenberg.net/etext/1329`. Other, corrupt editions can be found on `amazon.com`.

[2] Pepping, Simon, *DocbookInConTeXt*, available at `http://www.leverkruid.nl/context/index.html`.

[3] Rawls, John, *A Theory of Justice*, Revised edition, Belknap Press, Cambridge, MA, 1999.

[4] FontSite. `http://www.fontsite.com`. TEX font metrics and LATEX support files by Christopher League are available at `http://contrapunctus.net/league/haques/fs500tex/`.



1 The Séance

On a march evening, at eight o'clock, Backhouse, the medium—a fast-rising star in the psychic world—was ushered into the study at Prolands, the Hampstead residence of Montague Faull. The room was illuminated only by the light of a blazing fire. The host, eying him with indolent curiosity, got up, and the usual conventional greetings were exchanged. Having indicated an easy chair before the fire to his guest, the South American merchant sank back again into his own. The electric light was switched on. Faull's prominent, clear-cut features, metallic-looking skin, and general air of bored impassiveness, did not seem greatly to impress the medium, who was accustomed to regard men from a special angle. Backhouse, on the contrary, was a novelty to the merchant. As he tranquilly studied him through half closed lids and the smoke of a cigar, he wondered how this little, thickset person with the pointed beard contrived to remain so fresh and sane in appearance, in view of the morbid nature of his occupation.

"Do you smoke?" drawled Faull, by way of starting the conversation. "No? Then will you take a drink?"

"Not at present, I thank you."

A pause.

"Everything is satisfactory? The materialisation will take place?"

"I see no reason to doubt it."

"That's good, for I would not like my guests to be disappointed. I have your check written out in my pocket."

"Afterward will do quite well."

"Nine o'clock was the time specified, I believe?"

"I fancy so."

1

Figure 1 A chapter opening page

"It is Shaping stripped of illusion."

"How comes this horrible world to exist?"

Catice did not answer.

"Who is Surtur?"

"You will get nearer to him tomorrow; but not here."

"I am wading through too much blood," said Maskull. "Nothing good can come of it."

"Do not fear change and destruction; but laughter and joy."

Maskull meditated.

"Tell me, Catice. If I had elected to follow Spadevil, would you really have accepted his faith?"

"He was a great-souled man," replied Catice. "I see that the pride of our men is only another sprouting—out of pleasure. Tomorrow I too shall leave Sant, to reflect on all this."

Maskull shuddered. "Then these two deaths were not a necessity, but a crime!"

"His part was played and henceforward the woman would have dragged down his ideas, with her soft love and loyalty. Regret nothing, stranger, but go away at once out of the land."

"Tonight? Where shall I go?"

"To Wombflash, where you will meet the deepest minds. I will put you on the way."

He linked his arm in Maskull's, and they walked away into the night. For a mile or more they skirted the edge of the precipice. The wind was searching, and drove grit into their faces. Through the rifts of the clouds, stars, faint and brilliant, appeared. Maskull saw no familiar constellations. He wondered if the sun of earth was visible, and if so which one it was.

They came to the head of a rough staircase, leading down the cliffside. It resembled the one by which he had come up; but this descended to the Wombflash Forest.

"That is your path ," said Catice, "and I shall not come any farther."

Maskull detained him. "Say just this, before we part company—why does pleasure appear so shameful to us?"

"Because in feeling pleasure, we forget our home."

"And that is...?"

"Muspel," answered Catice.

Having made this reply, he disengaged himself, and, turning his back, disappeared into the darkness.

Maskull stumbled down the staircase as best he could. He was tired, but contemptuous of his pains. His uninjured probe began to discharge matter. He lowered himself from step to step during what seemed an interminable time. The rustling and sighing of the trees grew louder as he approached the bottom; the air became still and warm.

He at last reached level ground. Still attempting to proceed, he began to trip over roots, and to collide with tree trunks. After this had happened a few times, he determined to go no farther that night. He heaped together some dry leaves for a pillow, and immediately flung himself down to sleep. Deep and heavy unconsciousness seized him almost instantly.

Figure 2 A page spread as arranged in signatures

TEX and Linguistics

Steve Peter
Beech Stave Press
310 Hana Road
Edison, NJ 08817, USA
speter@beechstave.com

Abstract

TEX has long been associated with mathematics and "hard" sciences such as physics. But even during the early days of TEX, linguists were attracted to the system, and today a growing number of them are turning to TEX (LATEX, ConTEXt). Aside from the general advantages of TEX for producing academic papers, it offers linguists largely intuitive means for dealing with often complex notational issues. In this paper, an abbreviated version of my Practical TEX 2004 talk, I show some notational issues and their solutions in TEX.

Why TEX?

As a linguist and an avid user of TEX, I'm frequently asked why linguists would want to use TEX, as opposed to a word processor, to write their papers. Of course, there are the general reasons why any academic would benefit from TEX, such as easy handling of numbered examples, footnotes that make sense, bibliographic management via BIBTEX, etc.

For me, the main reason to use TEX to typeset linguistic papers and books is due to the complex, but often mathematically-inspired, notational systems used in the various subfields of linguistics. In fact, there are some cases[1] where the ability of TEX to format certain constructs aided their adoption by the field.

In this paper, I discuss various aspects of using TEX and LATEX to typeset linguistics. One thing you will find largely absent here is a discussion of Omega, which should offer great hope for linguists using TEX. Until its development is further along, discussion of it only presents users with a utopian taste of what might be. I would dearly love for Omega to advance, but the wait (for me) has been painful. Your mileage may of course vary. *Digital Typography Using LATEX* [1] has a good overview of language support in Omega.

I should note that the union of TEX and linguistics goes back quite far, even here in the pages of *TUGboat* [2, 3]. Donald E. Knuth[2] notes that linguists were among the first outside of mathematics to embrace TEX.

[1] Such as Attribute Value Matrices in Head-driven Phrase Structure Grammar.

[2] Personal communication at the Practical TEX banquet.

The field of linguistics

Linguistics is a large field that stands at the crossroads of several other disciplines, but is united in dealing with the scientific study of language. As you might expect from a large field, linguistics is commonly subdivided into various disciplines, each of which has various notational traditions and goals.

What is important for TEXnicians is that the notational issues presented here fall essentially into the general categories of "special symbols" or "special layouts". In order to partition the field into units we can deal with here, though, let us instead adopt a fairly traditional division that linguists often use.

1. philology
2. phonology
3. phonetics
4. syntax
5. semantics
6. "hyphenated"
   (a) psycholinguistics
   (b) sociolinguistics
   (c) biolinguistics
   (d) discourse analysis
   (e) ...

Fortunately for us, the "hyphenated" subdivisions largely make do with notation from other subfields, so they need not concern us further here.

In the remainder of the paper, I will take up each subfield in turn and discuss notational issues and how they may be solved with TEX. Most of the discussion deals with various LATEX packages, which reflects the "market share" of LATEX. Some of what

follows can be done with more or less difficulty in Plain TEX or ConTEXt.[3]

Due to the complexity of syntactic notation and the generality of application of tree structures, I will postpone discussion of Syntax until installment two of this paper.

Philology

Philology was once the general term for linguistic science, but is now more commonly used to refer to textual (rhetoric, poetics, textual criticism) and historical/diachronic linguistics. Most of the notational issues here deal either with different writing systems or with modifications of Latin.

The latter is usually quite straightforward in TEX, such as *ā, ē, ī, ō, ū, š, ṣ*, etc., obtained by `\=a`, `\=e`, `\=\i`, `\=o`, `\=u`, `\v{s}`, `\d{s}`. For example, Figure 1 is an example of something you might encounter in a paper on Indo-European.[4]

to me (although the suggestion of Kuryłowicz, *Apophonie* 170, that the ablaut *CeHi* : *CHī* is paralleled by a type *CeRi* : *CRī* seems worth considering).

2.2.4. When the laryngeal followed **r̥, *l̥, *m̥*, or **n̥*, we expect the resonants to become *αρ, αλ, αμ, αν*, with the same *α* that appears outside laryngeal environments, and this is what we often get, e.g. *ἔβαλον* 'they threw' < **ɛǧʷl̥E-*, *ἔκαμον* 'they toiled' < **ɛḱm̥A-*. But where the following laryngeal was *O*, we generally get *ορ, ολ* : *ἔμολον* 'they went', *ἔθορον* 'they leapt', *ἔτορε* 'pierced', *ἔπορον* 'they granted' to the presents *βλώσκω, θρώ(ι)σκω, τιτρώσκω* and the perfect *πέπρωται*. An ablaut *ρω* : *αρ* or *λω* : *αλ* seems not to occur in Greek. F. B. J. Kuiper has suggested, *India Antiqua* 199, that the development to *ορ* and *ολ* here was phonetically regular, the laryngeal influencing the vocalization of the resonant. I doubt that this is correct, because I believe that Dor. *πρᾶτος* represents a direct outcome of **pr̥O-tos* (cf. §2.3.2). If preconsonantal *O* failed to color the

Figure 1: What an Indo-Europeanist reads at the breakfast table

Paleography, for example, often uses a dot below a letter to indicate an obscured reading, which is quite easy to do in TEX (via `\d{}`), but in Word requires a special font, and a utility to access the needed characters.[5] A few other symbols require the use of the TIPA package, which we will discuss below in the section on Phonology.

A major undertaking in philology is the production of critical editions (see Figure 2). The requirements of line numbers, cross-references lemmata, layer upon layer of notes, marginalia, etc. can bring a typesetter to the brink of madness, but for the edmac (Plain TEX), ledmac and ednotes (LATEX) packages. Unfortunately, none of these packages offers a complete solution, so you will need to select one based on your specific goals and circumstances. Uwe Lück offers a critical overview of these critical edition packages in [4]. Recently, David Kastrup has created the bigfoot package [5], but I have yet to try it, so I cannot offer an opinion.

[3] Hans Hagen is aware of many of these issues, and we are working on adding more support for linguistics to ConTEXt.

[4] From *The Collected Writings of Warren Cowgill*, Beech Stave Press, 2005.

[5] I once tried to explain to my advisor how to get at some of these characters on a Mac. He shook his head and told me, "If I learn how to do that, I'll forget Hittite."

1 A dhuine gan chéill do mhaisligh an chléir
b is tharcaisnigh naomhscruipt na bhfáige,
c na haitheanta réab 's an t-aifreann thréig
d re taithneamh do chlaonchreideamh Mhártain,
e cá rachair 'od dhíon ar Íosa Nasardha
f nuair chaithfimid cruinn bheith ar mhaoileann Josepha?
g Ní caraid Mac Crae chuim t'anama ' phlé
h ná Calvin bhiais taobh ris an lá sin.

2 Nách damanta an scéal don chreachaire chlaon
b ghlac baiste na cléire 'na pháiste
c 's do glanadh mar ghréin ón bpeaca ró-dhaor
d trí ainibhfios Éva rinn Ádam,
e tuitim arís fé chuing na haicme sin
f tug atharrach brí don scríbhinn bheannaithe,
g d'aistrigh béasa agus reachta na cléire
h 's nách tugann aon ghéilleadh don Phápa?

3 Gach scolaire baoth, ní mholaim a cheird
b 'tá ag obair le géilleadh dá tháille
c don doirbhchoin chlaon dá ngorthar Mac Crae,
d deisceabal straeigh as an gcolláiste.
e Tá adaithe thíos in íochtar ifrinn,
f gan solas gan soilse i dtíorthaibh dorcha,
g tuigsint an léinn, gach cuirpeacht déin
h is Lucifer aosta 'na mháistir.

22 *Teideal*: Dhuinnluinng T, Seághan Mac Domhnaill cct B
1.a dhuinne T 1.a mhaslaidh T, mhaslaig B 1.c raob T 1.d le B 1.e dod B 1.f chaithfamíd T 1.f maoilinn B 1.g phleidh T 1.h bhíos B 1.h leis B 2.a claon B 2.c glannuig T 2.d ainnibhfios T, ainnbhfios B 2.d Éabha B 2.g is B 2.h tuigionn T 3.a sgollaire T 3.a mholluim T 3.b 'tág ccobar T 3.b re B 3.c dorbhchon daor B 3.d straodhaig T 3.e fhadoghthe tsíos T 3.e fadaighthe B 3.f sollus T 3.g cuirripeacht T 3.h Luicifer T, Lúcifer B 3.h mhaighistir T

Figure 2: A critical edition

Typesetting Greek critical editions presents the same problems as above, plus the need for good Greek fonts. Claudio Beccari [6] has extended babel to produce a remarkable facsimile of the famous Teubner editions. It still lacks some refinement for producing the critical apparatus, but the package is under active development, and the results thus far are quite pleasing.

But Greek fonts aren't an issue just when doing Greek critical editions. Due to whatever historical accident, Greek examples in philology are usually typeset in Greek, even while other languages that don't use the Latin alphabet (such as Sanskrit, Russian, Armenian, Tocharian, etc.) are transliterated. Fortunately there are several options for getting and entering Greek examples. The Beccari Greek fonts are excellent, and there is also the PSGreek package [7], which bundles Greek PostScript fonts and a style file to make accessing them easier, by hiding some

of the horrors of encoding vectors. The quality of the PS fonts bundled is somewhat uneven, and installing new fonts for use in the same manner is not easy. To do so requires the grkfinst fontinst plugin [8] and some time configuring. I wish it were a bit easier, since the PSGreek interface is one I find quite comfortable to use, and it has proven to be a lifesaver for switching Greek fonts.

The Greek in Figure 1 was produced with PSGreek. For example, to get ἔμολον 'they went', you enter `\textgreek{>'emolon}`. I am now quite used to entering Greek in this manner, and therefore I can do it quite rapidly. However, you may be more comfortable entering Greek in Unicode, given an appropriate text editor. For that, put the following in your preamble:

```
\usepackage{ucs}
\usepackage[utf8]{inputinc}
\usepackage[polutonikogreek,english]{babel}
```

There are two general contexts for typesetting languages in alphabets other than Latin. First, of course there are times when you need to typeset a single language solely for speakers of that language, such as setting a Russian text in Cyrillic for a Russian reader. On the other hand, at times it is necessary to mix two or more languages, such as in dictionaries or instructional material.

Both scenarios are supported in TEX, although dealing with encoding vectors can cause a headache or two.[6] Since I can't detail all possible language packages, let me limit myself here to a couple of packages I've found to be useful.

Underpinning nearly all multilingual endeavors in LATEX is babel by Johannes Braams [9]. It is included in (I believe) all TEX distributions, the manual is comprehensive and well written, and you should spend some time familiarizing yourself with it if you plan to do multilingual typesetting.

For Russian and the other Cyrillic-alphabet languages, there is the default Computer Modern Cyrillic font, which matches the standard Soviet look nicely.[7] At some point, though, you'll no doubt want a change of pace. The pscyr package [10] contains a number of serif, sans serif, and a couple of display faces.

[6] For Mac OS X users, the XETEX system frees you from many of these problems. `http://scripts.sil.org/cms/scripts/page.php?site_id=nrsi&item_id=xetex`. However, you cannot exchange source files with colleagues who use other operating systems.

[7] The Soviets heavily standardized book typefaces at a time when "modern" fonts were popular. There were some fantastic Russian typefaces developed during the 1920s that were neglected for decades.

Languages that use Indic scripts, such as Devānagarī, have a complication that not all graphemes occur in the same order as they are pronounced, plus there are many, many di- and trigraphs. The devnag package [11] provides a preprocessor to take care of these complexities, plus good fonts and macros for both Plain TEX and LATEX. Using devnag makes it possible to typeset a bilingual critical edition with essentially the same input for both the Devānagarī and the transliterated text. Figure 3 shows the vowels of Marāṭhī, typeset with the devnag package.

अ	आ	इ	ई	उ	ऊ
*a*bout	c*a*r	s*i*t	s*ea*t	p*u*t	r*oo*t
ऋ	ऌ	ए	ऐ	ओ	औ
und*er*	bott*le*	s*ay*	b*y*	r*oa*d	l*ou*d
अं	अः				

Figure 3: The vowels of Marāṭhī

For languages written in the Arabic alphabet (such as Arabic, Persian, Pashto, etc.), Klaus Lagally's ArabTEX is a must. The system is by now quite stable, and the output is very good. Several people are working on various extensions, especially for typesetting Arabic mathematics. See for example, Lazrek et al. [12, 13]. While it is possible to typeset Hebrew using ArabTEX, Alan Hoenig's Makor [14] is worth every penny.[8]

Typesetting Chinese using TEX is possible with the CJK [15] package (which provides for much more than just Chinese, Japanese, and Korean support). However, I prefer ConTEXt, due to its support of visual debugging via `\tracechinesetrue`. Numbering can be toggled between Chinese and western styles via `[conversion=chinese]` or `[conversion=numbers]`. More traditional vertical typesetting is possible essentially by flowing the text into narrow columns.

Semantics

Semantics is the study of meaning, and the notation used is tied closely to formal logic. Thus it is very straightforward to typeset with TEX. So the function of the set of things similar to houses is denoted by $\lambda x Similar\_to(x, houses)$. The TEX to get this is `$\lambda x \textit{Similar\_to}(x, \textit{houses})$`. We had to wrap the 'English' inside the function with `\textit` to prevent TEX

[8] Yes, it is free software, and yes, I am making an exception to not discussing Omega software.

from interpreting the words as a series of variables. In some cases `\mbox` will work, and note that sometimes spaces inside the `\mbox`es are important. So a possible interpretation for the sentence *I have told one friend of mine all those stories*[9] is given as $\exists x[\forall y[(x \in \text{friends of mine} \wedge y \in \text{those stories}) \rightarrow \text{I have told } y \text{ to } x]]$, or in TEX terms

```
$\exists x [\forall y[
   (x \in \mbox{friends of mine}
    \wedge y \in \mbox{those stories})
   \rightarrow \mbox{I have told }
    y \mbox{ to } x]]$
```

Double brackets (representing semantic evaluation) are provided by the stmaryrd package [16]. With that loaded, typing `$\llbracket(MN)\rrbracket^{ \mathcal{M}}$` yields $[\![(MN)]\!]^{\mathcal{M}}$. You may also need to load the latexsym package for an occasional symbol.

Phonetics and Phonology

Phonetics is a branch of acoustics that deals with speech sounds and their production and perception. The notation used is a combination of transcription symbols (as covered below) and diagrams representing articulatory spaces. For example, Figure 4 shows a typical representation of a vowel system.[10]

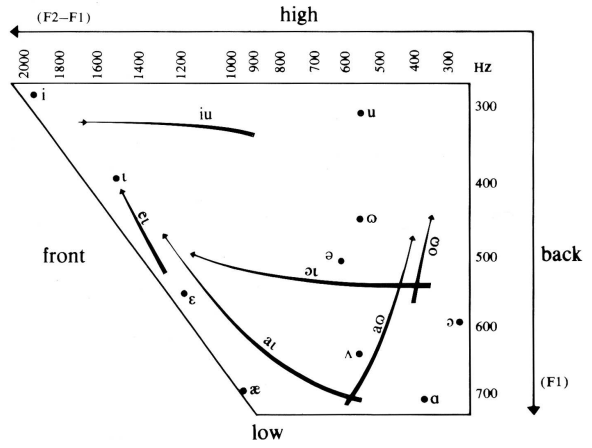


Figure 4: Some American English vowels

Also frequent are diagrams of the human vocal tract. Unfortunately there is no easy way to handle automatic generation of notation of this type. A typical way to handle them is to create the illustration in a vector program (such as Inkscape or Illustrator) and then to import it into TEX. Given the complexity of creating them, publishers (and therefore also authors) have been reluctant to use them aside from in very specialized books.[11]

The International Phonetic Association came into existence in 1886 with a goal of promoting phonetics in education and the creation of an international phonetic alphabet (now known as the IPA) for the universal transcription of languages.[12] A separate tradition of transcription developed among anthropological linguists in America. Both systems of transcription [18, 19] are supported via the TIPA package [17].

While there are numerous fonts that provide IPA symbols that more or less match existing typefaces, there are to my knowledge still only a small number of type families that have complete sets of corresponding IPA symbols: Computer Modern, Lucida, Times, Le Monde, Gentium, Garamond, and Stone — and two of them are provided for by TIPA. To wit, the Computer Modern IPA symbols work best with Computer Modern,[13] but they will fit in reasonably with other vertical-stress typefaces. The Times IPA symbols, again, work best with Times, but in a pinch, they fit in with other oblique-stress typefaces. In a sans-serif environment, TIPA provides a Helvetica-like symbol set.

In addition to the IPA fonts and the interface (more on which below), TIPA provides a style file to produce simple vowel diagrams (simpler than the one shown in Figure 4). I could conceive — given enough labor — of creating the more complex charts like Figure 4, with arrows and swooshes, programmatically via the tools provided by PSTricks [20] or MetaFun [21].

TIPA provides for a couple of different ways to enter phonetic notation. There are long forms that have generally mnemonic names, so I can write [əhˈa] as `[\textschwa h\textprimstress a]` if my paper uses a limited set of symbols, and I don't want to learn the more involved transcription.

On the other hand, if you need to input larger amounts of transcription, it is useful to enter the IPA environment via `\textipa{}`, `{\tipaencoding }`, or `\begin{IPA}` and `\end{IPA}`. So, if we enter the IPA environment and type

```
D@ "nO;T "wInd @nd D@ "s2n w@ dIs"pju;tIN
wItS w@z D@ "str6Ng5, wEn @ "tr\ae vl5
keIm @"l6N "r\ae pt In @ "wO:m "kloUk. DeI
```

[9] From Ray Jackendoff, *Semantic Interpretation in Generative Grammar*, Cambridge: MIT Press, 1972, p. 308.

[10] From Peter Ladefoged, *A Course in Phonetics*, San Diego: Harcourt Brace Jovanovich, 1982, p. 198.

[11] They can hardly be avoided in an introduction to phonetics, for example.

[12] Remember that there were no tape recorders in 1886!

[13] And variants such as Latin Modern.

```
@"gri:d D@t D@ "w2n hu; f3;st s@k"si;dId
In "meIkiN D@ "tr\ae vl5  teIk hIz "kloUk
6f SUd bI k@n"sId@d "str6Ng@ D@n DI "2D@.
```

this will be rendered into IPA as follows:[14]

ðə ˈnɔˌθ ˈwɪnd ənd ðə ˈsʌn wə dɪsˈpjuˌtɪŋ wɪtʃ wəz ðə ˈstrɒŋgɐ, wɛn ə ˈtrævlɐ keɪm əˈlɒŋ ˈræpt ɪn ə ˈwɔːm ˈkloʊk. ðeɪ əˈgriːd ðət ðə ˈwʌn huˌ fɜˌst səkˈsiˌdɪd ɪn ˈmeɪkɪŋ ðə ˈtrævlɐ teɪk hɪz ˈkloʊk ɒf ʃʊd bɪ kənˈsɪdəd ˈstrɒŋgə ðən ðɪ ˈʌðə.

TIPA allows you to enter tonal specifications and has many other nice features to explore. I heartily recommend reading the excellent manual included in the package.

As I mentioned earlier in the paper, the TIPA package also allows you to enter Indo-European reconstructed forms. For example, the work for '100' is reconstructed as $*\hat{k}\underset{\circ}{m}tóm$, which can be entered as `\textroundcap{k}\textsubring{m}t\'om` or as `\textipa{*\|c{k}\r*mt\'om}`.

A new notational twist entered both phonology and syntax within the past decade as Optimality Theory grew in popularity. The central part of its notation are the so-called optimality tableaux. There are a number of ways to enter them, but I've settled on using PSTricks together with colortab [22]. Here is source and output using some totally nonsensical data.

```
\begin{tabular}[t]{r|c|c|c|}
\cline{2-4}
 & /ba/  & \textipa{\!@} & b\textturna\\
\LCC
 & & & \lightgray \\ \cline{2-4}
\w   & [ba] & & * \\ \cline{2-4}
 & [*ba] & *! & \\ \cline{2-4}
\ECC
\end{tabular}
```

/ba/	ɓə	bɐ
[ba]		*
[*ba]	*!	

One other subdivision of phonology, computational phonology, uses a mixture of standard phonological notation plus tree structures. As such it will be covered in the second installment of this paper.

Next time

The subfield of linguistics with perhaps the widest variety of notations is syntax. I will postpone until installment two of this paper a discussion of the trees, matrices, and derivations, as I wish to cover them in greater detail than space or time allows at present. In particular, the macros for drawing trees have far wider application than just linguistics.[15]

[14] *The north wind and the sun were disputing which was the stronger, when a traveler came along wrapped in a warm cloak. They agreed that the one who first succeeded in making the traveler take his cloak off should be considered stronger than the other.* The text comes from the International Phonetic Association.

[15] As Nelson Beebe remarked at the PracTEX conference. Installment two will, I hope, serve as his requested paper.

References

[1] Apostolos Syropoulos, Antonis Tsolomitis, and Nick Sofroniou, *Digital Typography Using LaTeX*, New York: Springer, 2003.

[2] Christina Thiele, "TEX and Linguistics", *TUGboat* **16**(1), 42–44.

[3] Christina Thiele, "TEX and the Humanities", *TUGboat* **17**(4), 388–393.

[4] Uwe Lück, "ednotes—critical edition typesetting with LaTeX", *TUGboat* **24**(2), 224–236.

[5] David Kastrup, "The bigfoot bundle for critical editions", Preprints for the 2004 Annual TUG Meeting, 105–110.

[6] CTAN/macros/latex/contrib/teubner

[7] CTAN/fonts/greek/psgreek

[8] CTAN/fonts/utilities/fontinst-contrib/grkfinst

[9] http://www.braams.cistron.nl/babel/

[10] http://www.opennet.ru/prog/info/1117.shtml

[11] http://devnag.sarovar.org/

[12] Mustapha Eddahibi, Azzeddine Lazrek, and Khalid Sami, "Arabic mathematical e-documents", Preprints for the 2004 Annual TUG Meeting, 42–47.

[13] Azzeddine Lazrek, "CurExt, typesetting variable-sized curved symbols", *TUGboat* **24**(3), 323–327.

[14] CTAN/language/hebrew/makor

[15] CTAN/language/chinese/CJK

[16] CTAN/fonts/stmaryrd

[17] CTAN/fonts/tipa

[18] Geoff Pullum and William Ladusaw, *Phonetic Symbol Guide*, Chicago: University of Chicago Press, 1996.

[19] *Handbook of the International Phonetic Association*, Cambridge: Cambridge University Press, 1999.

[20] http://tug.org/applications/PSTricks

[21] http://contextgarden.net/MetaFun

[22] CTAN/macros/generic/colortab

MetaPlot, MetaContour, and Other Collaborations with METAPOST

Brooks Moses
Mechanical Engineering
Building 520
Stanford University
Stanford, CA 94305
USA
bmoses@stanford.edu

Abstract

Most methods of creating plots in METAPOST work by doing all of their calculations in METAPOST, or by doing all of their calculations in a preprocessing program. There are advantages to dividing the work more equitably, by doing the mathematical and data-visualization calculations in a preprocessing program and doing the graphical and layout calculations in METAPOST. The MetaPlot package provides a standard, flexible, interface for accomplishing such a collaboration between programs, and includes a general-purpose set of formatting macros that are applicable to a wide range of plot types. Examples are shown of linear plots with idiosyncratic annotation and two-dimensional contour plots with lines and filled contours on a non-Cartesian mesh.

1 Introduction

One of the challenges of scientific writing in TEX (or in LATEX) is producing figures that are of comparable quality to the typesetting. These figures often include plots and graphs that represent mathematically-intense visualization of large data files, implying that some form of specialized program must be used to create them. They also typically contain labels, notes, and other text that should be typeset in a manner consistent with the rest of the document, which requires using TEX's typesetting engine.

Traditionally, programs that meet these goals have taken one of two approaches. The first approach, used by programs such as ePiX [1] and Gnuplot [2], is to implement the program in a "traditional" programming language such as C++ or Fortran, and produce the complete figure as output in TEX/eepic or METAPOST code, which is then post-processed. The other approach, taken by METAPOST's `graph` package and m3D [3], is to implement the program directly in METAPOST's macro language.

There are advantages and tradeoffs to both of these approaches. Programming in METAPOST allows one to work directly with the language features such as declarative equations and ability to measure the size of typeset text, and thus to specify the figure layout in an intuitive, simple, and flexible manner. On the other hand, programming in a traditional language allows one to write mathematically-intensive programs that use floating-point numbers and can be compiled rather than run slowly through an interpreter; in addition, it may allow one to take advantage of existing visualization libraries, or to provide an interactive user interface.

This paper describes an intermediate approach, which combines the benefits of METAPOST and traditional-language programs. The initial data processing is done with a program written in a traditional language, which produces a METAPOST source file containing the processed data in an encapsulated form. This processed data is then fed into a set of METAPOST formatting macros, and the scaling, drawing, and annotation of the plots is all done by user-written commands within METAPOST.

Creating plots in two steps in this manner has several advantages:

- The initial data visualization can be done in a special-purpose program that uses a programming language and code libraries intended for substantial computations, with no need to implement more than a very simple output routine.
- The METAPOST macros for formatting plots and arranging them within a figure are largely independent of the details of the plots they are working with, and can be written in a generic manner suitable for widespread distribution.
- The layout of any given figure can be done using the same processes as a native-METAPOST

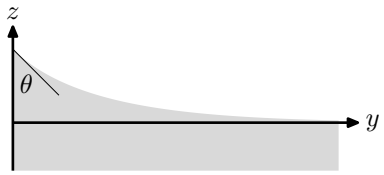


Figure 1: A capillary surface on a liquid touching a solid wall, after Batchelor [4].

drawing.

2 A simple example

Consider, by way of example, a plot of the shape of a meniscus formed by a liquid surface meeting a solid wall, as shown in Figure 1. The surface curve is given by a somewhat complicated expression involving inverse hyperbolic cosines,[1] and is representative of calculations that would be easier to do in a traditional programming language.

The C++ program to produce this curve in a METAPOST format is straightforward. The most complicated part is the function to generate a string containing a METAPOST representation of a point, which we accomplish using the <*sstream*> standard library.

```
string mpoint(double x, double y) {
      ostringstream pointstring;
  pointstring.setf(ios_base::fixed, ios_base::
      floatfield);
  pointstring.precision(5); pointstring << '(' << x
      << ',' << y <<
  ')'; return pointstring.str(); }
```

The **setf** and **precision** commands set the numeric format for the stream (fixed-precision, five decimal places), and then the coordinates are fed into the **ostringstream** with the appropriate punctuation, producing a result like `(0.01556,0.75006)`.

Given this and a *capillary*() function that computes the equation for the surface, creating the METAPOST command for the curve is simply a matter of looping through the points and dumping them to the standard output, with appropriate text before and after the loop to define the picture variable and close the curve into a cyclic path.

```
int main() { double theta = pi/4.0; double d =
      1.0; double h =
  sqrt(2.0 * d*d * (1.0 - sin(theta))); double y, z;

  cout << "picture capillary;\n"; cout << "
      capillary :=
  nullpicture;\n"; cout << "addto capillary
      contour " << mpoint(0.0,
  h); for(int i = 99; i > 2; i-- ) { z = (i/100.0) * h;
      y =
  capillary(z,h,d); cout << " .. " << mpoint(y, z); }
      cout << " -- "
  << mpoint(y, -0.5); cout << " -- " << mpoint
      (0.0, -0.5); cout << "
  -- cycle;\n"; }
```

This produces the following METAPOST code as output:

```
picture capillary; capillary :=
    nullpicture; addtocapillary contour
    (0.00000,0.76537) .. (0.00772,0.75771)..
    (0.01556,0.75006)
        % [ ... and so forth ... ]
.. (3.39322,0.02296) --
    (3.39322,-0.50000) --(0.00000,-0.50000)
    -- cycle;
```

We can then follow this with additional METAPOST commands to scale the figure to an appropriate size for printing on the page, and draw axes and labels, in order to produce the plot shown in Figure 1.

```
beginfig(1) draw (capillary scaled 0.5in) withcolor
  0.85white; linecap := butt; pickup pencircle scaled
      1pt; drawarrow
  (0,-0.25in) -- (0, 0.5in); label.top(btex $z$ etex
      ,(0,
  0.5in)); x1 := (xpart(lrcorner capillary) * 0.5in, 0)
      + (0.1in, 0);
  drawarrow (0,0) -- x1; label.rt(btex $y$ etex, x
      1);
  pickup pencircle scaled 0.25pt; x2 := ulcorner
      capillary scaled
  0.5in; draw ((0,0) -- (0.24in, -0.24in)) shifted x2;
      label(btex
  $\theta$ etex, x2 + (0.07in, -0.18in)));
  endfig; end
```

Although this example produces a perfectly serviceable result, it has some noteworthy drawbacks. The scale factor of 0.5 in does not have a clear relationship to the size of the plot, and producing a plot of a particular size would require measurement of the *capillary* picture and explicit computation of the scale factor. The locations of the annotations

[1] For those who are curious, the equation (from [4]) is

$$\frac{y}{d} = \cosh^{-1}\frac{2d}{z} - \cosh^{-1}\frac{2d}{h} + \left(4 - \frac{h^2}{d^2}\right)^{\frac{1}{2}} - \left(4 - \frac{z^2}{d^2}\right)^{\frac{1}{2}},$$

where $h^2 = 2d^2(1 - \sin\theta)$ is the height of the meniscus, θ is the contact angle, and d is a scaling parameter related to the surface tension and liquid density.

are likewise determined by explicit measurement, or by being typed in directly. If we were to change one of the parameters in the C++ program and re-run it, many of the values in the METAPOST code would need to be changed as well.

3 A more general example: The MetaPlot package

The MetaPlot package is designed to address many of the shortcomings of the example given in Section 2. It provides a consistent way of transferring the plot commands and associated metadata from the generating program into METAPOST, and direct handles for manipulating the plots within METAPOST using its normal idiom of declarative equations rather than procedural assignments.

To accomplish this in a general manner, we define two types of METAPOST data structures: *plot objects* and *plot instances*. A plot object is a plot "in the abstract", containing paths, filled contours, and metadata that make up the plot (or a set of related plots), represented in a manner that is independent of the details of how the plot is positioned. By contrast, a plot instance is a plot "on the page", containing parameters for the scaling and positioning of a given plot, and a reference to a parent plot object that gives the actual pictures to be drawn.

A typical preamble for a figure using MetaPlot will consist of an **input** `metaplot` command to load the MetaPlot macros, an **input** command to load the METAPOST file that contains the plot objects (typically an output file from the preprocessing program), and calls to the MetaPlot macros to generate plot instances from the plot objects.

3.1 The concept of a plot-object

Suffix arguments and multi-token variable names in METAPOST allow us to define data structures that approximate structures or objects in more traditional programming. The correspondence is not exact; in particular, there is no data type associated with the overall object. METAPOST is simply passing around a fragment of a variable name and constructing complete variable names from it, so any arbitrary element can be added to the class without changing its type. Thus, the MetaPlot macros can deal with arbitrary types of plots in a generic manner, so long as they meet a few minimal requirements that allow them to be scaled and positioned.

The paths and contours that make up a plot object are not defined in terms of the native data coordinates, but are rescaled to fit within a unit box (that is, extending from 0 to 1 in both coordinate directions), which is treated as the bounding box of the plot for purposes of scaling and positioning.[2] As a result, the possibility of coordinates too small or too large for METAPOST's fixed-point number representation is avoided; in addition, positioning the plot on the page is a simple matter of scaling by the final width and height and shifting by the final position of the lower-left corner. The original data scales are stored in four numeric components that record the values corresponding to the extents of the bounding box, and can be used later to rescale the plot to an appropriate size and aspect ratio.[3]

The remaining details of the format can be shown by rearranging the example from Section 2 into a plot object, as follows. For purposes of later examples, we will presume that this has been saved as `capillary.mp`.

```
% Definition of capillary plot-object
% Picture components
picture capillary.fplot; capillary.fplot := nullpicture;
      addto
  capillary.fplot contour (0.00000,1.00000) ..
      (0.00227,0.99395)
  .. (0.00459,0.98790)
      % [ ... and so forth ... ]
  .. (1.00000,0.41329) -- (1.00000, 0.00000) --
      (0.00000, 0.00000) --
  cycle; picture capillary.lplot; capillary.lplot :=
      nullpicture;
  addto capillary.lplot doublepath
      (0.00000,1.00000)
  .. (0.00227,0.99395)
      % [ ... and so forth ... ]
  .. (1.00000,0.41329);

% Required metadata
numeric capillary.xleft; capillary.xleft = 0.0;
      numeric
capillary.xright; capillary.xright = 3.39322; numeric
      capillary.ybot;
capillary.ybot = -0.5; numeric capillary.ytop;
      capillary.ytop =
0.76537;

% Plot-specific metadata
pair capillary.contactpoint; capillary.contactpoint =
      (0.0, 1.0);
```

[2] Using a box from −4096 to +4096 would make better use of METAPOST's fixed-point number range, but even on a unit box expanded to a 2-meter-wide poster, the granularity is only 0.03 mm — which is better than most printers. A larger box is probably not worth the inconvenience.

[3] Although these variables are represented here as numerics and thus are still vulnerable to under- or overflow, it would be a simple matter to replace them with string-represented numbers from the `sarith` package.

```
  numeric capillary.contactangle; capillary.
        contactangle = 45.0;
```

In this case, I have also added an additional component: this version of *capillary* contains a path for the liquid surface line (*capillary.lplot*), as well as the original filled contour (now *capillary.fplot*); the decision about which of them to draw can be made later. A plot object can contain any number of these pictures (even zero), with arbitrary names.

The four required scale variables are *capillary.xleft*, *.xright*, *.ybot*, and *.ytop*; these, for purposes of the MetaPlot macros, must be named thus.

Finally, there are two metadata variables for this particular plot, *capillary.contactpoint* and *capillary.contactangle*, which will be useful in drawing the annotations. Any number of additional variables can be present, with arbitrary names. Of note is that *.contactpoint* is given in the same unit-box coordinate system that the paths and contours are in, allowing it to be positioned by the same macros that scale and position the picture components.

3.2 Creation of a plot instance

The next step after creating plot objects is manipulating them on the page by means of plot instances. A plot instance thus needs to contain three sets of components: coordinates and dimensions of the plot as shown on the page, a representation of the plot's internal scale for use in alignment and producing axes, and a means of accessing picture components from its parent plot object. These are created by the *plot_instantiate*() macro, which is part of MetaPlot; the version below is simplified somewhat.

```
% Args: inst is the new plot instance.
% plot_object is the parent plot object.
def plot_instantiate(suffix inst)(suffix plot_object) =

% Define (unknown) parameters for plot-instance
% location on page
  numeric inst.pagewidth, inst.pageheight; numeric
        inst.pageleft,
  inst.pageright, inst.pagetop, inst.pagebottom; inst.
        pageleft +
  inst.pagewidth = inst.pageright; inst.pagebottom +
        inst.pageheight =
  inst.pagetop;

% Define (known) parameters for plot's scaling
  numeric inst.scaleleft, inst.scaleright, inst.scaletop,
        inst.scalebottom; inst.scaleleft :=
              plot_object.xleft;
        inst.scaleright := plot_object.xright; inst.
              scalebottom :=
              plot_object.ybottom; inst.scaletop :=
              plot_object.ytop;

% Pointer-function to plot_object's plots, scaled
% and positioned.
  vardef inst.plot(suffix name) = plot_object.name
        xscaled
    inst.pagewidth yscaled inst.pageheight shifted (
          inst.pageleft,
    inst.pagebottom) enddef; enddef;
```

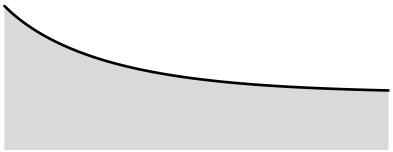


Figure 2: The capillary surface, in its unadorned form as plot object elements scaled to 2.0in by 0.75in.

Note that, immediately after a plot instance is created, the page information is unknown while the scale information is known.

We can now start putting plot objects on the page in a limited fashion, by assigning known values to the unknown page information, and then drawing the scaled picture elements.

```
input metaplot     % MetaPlot macros
input capillary    % capillary plot object

plot_instantiate(plotA, capillary); plotA.pageleft =
      0.0;
plotA.pagebottom = 0.0; plotA.pagewidth = 2.0in;
      plotA.pageheight =
0.75in; beginfig(2) draw plotA.plot(fplot)
      withcolor
0.85white; draw plotA.plot(lplot) withpen pencircle
       scaled 1pt;
endfig; end
```

The result of this is shown in Figure 2. Note that the color of the filled plot and the line size for the line plot are specified in the draw command, rather than in the plot object.

3.3 Manipulation of plot-objects

The bare plot instances are of little use without a set of macros for manipulating them. We start with a macro to set the x-axis and y-axis scales to equal values:

```
def plot_setequalaxes(suffix inst) = inst.pagewidth =
      inst.pageheight
```

```
∗ ((inst.scaleright − inst.scaleleft) / (inst.scaletop
      −
inst.scalebottom)); enddef;
```

This is written so that the page-related variables do not appear in the denominator of fractions, because either one (or both) of them may be unknown when the macro is called, and METAPOST can only solve linear equations.

There is also a set of macros for converting between locations expressed in the plot's coordinates and locations on the page. For example,

```
def plot_xpageloc(suffix inst)(expr scalex) = inst.
      pageleft + (scalex
− inst.scaleleft) ∗ (inst.pagewidth / (inst.scaleright
      −
inst.scaleleft)); enddef;
```

The additional macros in this series are *ypageloc*, *zpageloc* (which takes an x and a y coordinate as input, and returns a point), and *xscaleloc* and *yscaleloc* for the reverse direction of converting from a page location to a plot coordinate.

With these, we have most of what we need to manipulate plots in an intuitive way. For instance, consider the figure from Section 2, which can now (with some small changes) be written in a much more general way as

```
input metaplot     % MetaPlot macros
input capillary    % capillary plot object

plot_instantiate(plotB, capillary);
plot_setequalaxes(plotB); plotB.pageleft = 0.0; plotB.
      pagebottom =
0.0; plotB.pageheight = 0.75in; beginfig(3) draw
plotB.plot(fplot) withcolor 0.85white; linecap :=
      butt; pickup
pencircle scaled 1pt;
  % z-axis (vertical)
  z1 = (plotB.pageleft, plotB.pagebottom); z2 = (
      plotB.pageleft,
  plotB.pagetop + 0.1in);
  % y-axis (horizontal)
  z3 = (plotB.pageleft, plot_ypageloc(plotB,0.0)); z4
      =
  (plotB.pageright + 0.1in, plot_ypageloc(plotB,0.0));
      drawarrow z1 --
  z2; label.top(btex $z$ etex, z2); drawarrow z3
      -- z4;
  label.rt(btex $y$ etex, z4); pickup pencircle
      scaled
  0.25pt;
  % Label for contact angle
```

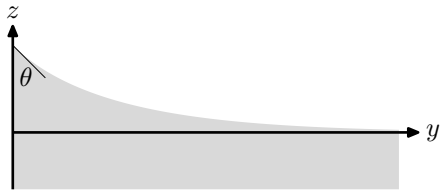


Figure 3: The capillary surface, with equal y and z scales, a page height of 0.75in, and appropriate annotations.

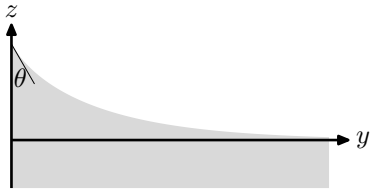


Figure 4: The capillary surface with parameters and page height as in Figure 3, but with $\theta = \pi/6$.

```
  z5 = plotB.plot(contactpoint); z6 = z5 + 0.24in ∗
      dir(−90 +
  capillary.contactangle); z7 = z5 + 0.18in ∗ dir(−90
      +
  0.5∗capillary.contactangle); draw z5 -- z6; label(
      btex
  $\theta$ etex, z7); endfig; end
```

The result of this is shown in Figure 3. We can demonstrate that this is flexible by changing the value of θ to $\pi/6$ rather than $\pi/4$, and recreating the figure using exactly the same files; the result is shown in Figure 4. Note that changing the contact angle raises the contact point, making the plot taller in scale coordinates; thus, it is drawn at a smaller scale to maintain the 0.75-inch page height.

Having two figures in this way is not the clearest way to compare the two plots, particularly with the differences in scale. A better approach is to overlay them at the same scale, making use of the existence of the filled plot from one plot object and the line plot from the other to provide a visually clear result. A simple way of placing both plots on the same coordinate axes is to require that their (0,0) and (1,1) points coincide on the page, which we do by means of the *plot_zpageloc* command; the remainder of the file is as much in the previous plots, although there is a little additional code to make certain that the axis-arrows cover both plots.

```
input metaplot     % MetaPlot macros
input capillary    % capillary plot object
input capillary2   % capillaryb plot object
```

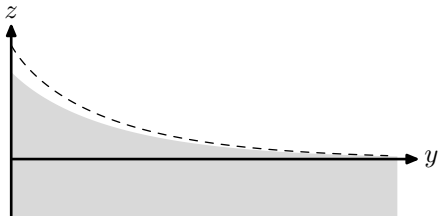


Figure 5: Two capillary surfaces, as in Figure 3 and Figure 4, showing the difference in the curves as a result of varying θ.

```
plot_instantiate(plotB, capillary);
plot_setequalaxes(plotB); plotB.pageleft = 0.0; plotB.
      pagebottom =
0.0; plotB.pageheight = 0.75in;

plot_instantiate(plotC, capillaryb); plot_zpageloc(plotB
      , 0.0, 0.0) =
plot_zpageloc(plotC, 0.0, 0.0); plot_zpageloc(plotB, 1.0,
      1.0) =
plot_zpageloc(plotC, 1.0, 1.0);

beginfig(5) linecap := butt; pickup pencircle scaled
  1pt; draw plotB.plot(fplot) withcolor 0.85white;
      draw
  plotC.plot(lplot) dashed evenly withpen pencircle
      scaled 0.5pt;
  % z-axis (vertical)
  z1 = (plotB.pageleft, plotB.pagebottom); x2 = plotB
      .pageleft; y2 =
  max(plotB.pagetop, plotC.pagetop) + 0.1in;
  % y-axis (horizontal)
  z3 = (plotB.pageleft, plot_ypageloc(plotB,0.0)); x4
      =
  max(plotB.pageright,plotC.pageright) + 0.1in; y4 =
  plot_ypageloc(plotB,0.0); drawarrow z1 -- z2;
      label.top(btex
  $z$ etex, z2); drawarrow z3 -- z4; label.rt(
      btex
  $y$ etex, z4); endfig; end
```

The result of this is shown in Figure 5.

3.4 Creation of axes

Any quantitative graph is meaningless without grid-labels for the coordinate axes, and so MetaPlot includes macros to create them. Unlike METAPOST's `graph.mp` package, MetaPlot's axis-drawing functionality requires that the user specify most of the details of the formatting, with the benefit of having a much more flexible implementation.[4]

The core of the axis-drawing functionality is a set of macros for creating generic tickmarks, labeled tickmarks, rows of tickmarks, and so forth, which are included with MetaPlot in a `axes.mp` file (and thus, for consistency, are prefaced with *axes_* rather than *plot_*). These are interfaced to the plot object coordinates by the *plot_xtickscale* and *plot_ytickscale* macros.

```
def plot_xtickscale (suffix inst) (expr startpoint,
      endpoint,
  ticklength , tickspace , tickdir , tickzero , tickstep
      , ticklabelformat )
  =
  axes_tickscale (
    startpoint , % First endpoint of the tickrow
    endpoint,    % Second endpoint of the tickrow
    ticklength , % Length of tickmarks
    tickspace , % Space between tickmark and label
    tickdir ,    % Tickmark direction
    plot_xscaleloc (inst)(xpart(startpoint)),
                 % Coordinate value at first endpoint
    plot_xscaleloc (inst)(xpart(endpoint)),
                 % Coordinate value at second
                     endpoint
    tickzero ,   % Coordinate value for a known
                 %  tick location
    tickstep ,   % Coordinate space between ticks
    ticklabelformat
                 % Format for tick labels
                 %  (syntax from format.mp package)
                 %  (use "" for no tick labels)
  ) enddef;
```

The *plot_ytickscale* definition is nearly identical. Note that these macros do not actually draw the tickmarks; they return a picture object, which can then be explicitly drawn or otherwise manipulated.

A simple way of adding grid labels to the previous example would be the following:

```
beginfig(6)
  % [ ... repeat of definitions from fig(4) ... ]
  x5 = plotB.pageleft; x6 = x4; y5 = y6 = plotB.
      pagebottom; draw
  plot_xtickscale(plotB)(z5, z6,
    0.08in, 0.06in, down, 0.0, 1.0, "%3f")
    withpen pencircle scaled 0.5pt; y7 = plotB.
        pagebottom; y8 = y2; x7
  = x8 = plotB.pageleft; draw plot_ytickscale(plotB)(
      z7, z8,
```

[4] There is, of course, no need for flexible implementations and simple interfaces to be mutually exclusive, and functions for more automated axes may be included in MetaPlot as it continues to be developed.

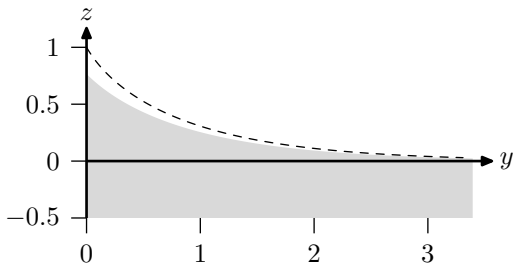


Figure 6: A repeat of Figure 5, with simple grid labels added.

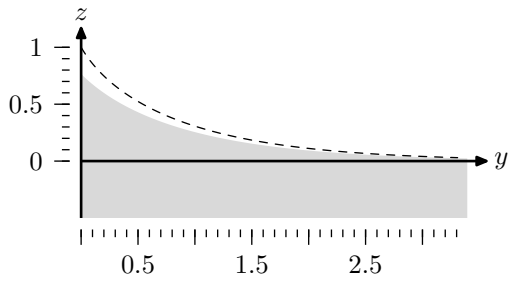


Figure 7: A repeat of Figure 5 again, with more advanced grid labels.

```
    0.08in, 0.06in, left, 0.0, 0.5, "%3f")
    withpen pencircle scaled 0.5pt; endfig;
```

The results of this are shown in Figure 6. As can be seen with the x-axis, the *tickscale* macros do not include the axis-lines themselves, thus allowing the user to draw them with a different line style than that used for the ticks, or to leave them off entirely.

For a more polished look, we can move the grid ticks a small distance away from the plot, limit the y-axis range to the region that has meaningful significance, and add intermediate ticks without labels. In addition, this example illustrates the use of the *tickzero* parameter to start the labeled x-axis ticks at 0.5 rather than zero.

```
beginfig(7)
  % [ ... repeat of definitions from fig(4) ... ]

  x5 = plotB.pageleft; x6 = x4 − 0.1in; y5 = y6 =
      plotB.pagebottom −
  0.06in; draw plot_xtickscale(plotB)(z5, z6,
    0.08in, 0.06in, down, 0.5, 1.0, "%3f")
    withpen pencircle scaled 0.5pt; draw
      plot_xtickscale(plotB)(z5,
  z6, 0.08in, 0.06in, down, 0.0, 1.0, "") withpen
      pencircle scaled
  0.5pt; draw plot_xtickscale(plotB)(z5, z6, 0.04in,
      0.06in, down,
  0.0, 0.1, "") withpen pencircle scaled 0.5pt; y7 =
      y4; y8 = y2 −
  0.1in; x7 = x8 = plotB.pageleft − 0.06in; draw
  plot_ytickscale(plotB)(z7, z8,
     0.08in, 0.06in, left, 0.0, 0.5, "%3f")
    withpen pencircle scaled 0.5pt; draw
      plot_ytickscale(plotB)(z7,
  z8, 0.04in, 0.06in, left, 0.0, 0.1, "") withpen
      pencircle scaled
  0.5pt; endfig;
```

The result is shown in Figure 7.

3.5 MetaContour: A C++ program for contour plots

Now that the METAPOST side of the collaboration has been described in some detail, we return to the matter of programs that generate plot objects as output. One of the particular reasons for developing MetaPlot was to have a way of producing contour plots, and so the MetaPlot package comes with a C++ program, MetaContour, for creating them.

The internals of MetaContour are beyond the scope of this paper, but it does make use of one additional capability of plot objects that is worth noting — the ability to include color information. The plot object is defined with commands like the following, with color directives.

```
picture contplotA.LinePlot; contplotA.LinePlot :=
      nullpicture; addto
contplotA.LinePlot doublepath (0.48075,0.50000)
      -- (0.48163,0.50597)
withcolor contourcolor27; addto contplotA.LinePlot
       doublepath
(0.48420,0.50000)-- (0.48492,0.50490) withcolor
      contourcolor28; addto
contplotA.LinePlot doublepath (0.45994,0.50000)
      -- (0.46169,0.51245)
withcolor contourcolor23;
     % [ ... and so forth ... ]
```

Then, before the plot object file is read into the main METAPOST file, the *contourcolor* array is defined as desired.

```
% Contour colors for grayscale scheme
color contourcolor[ ]; contourcolor0 = 1white;
      contourcolor1 =
0.98white;
     % [ ... and so forth ... ]
contourcolor30 = 0.4white;
```

Thus, each line of the contour plot is associated with a color, and it will be drawn in that color unless it is overridden by another color directive; for instance,

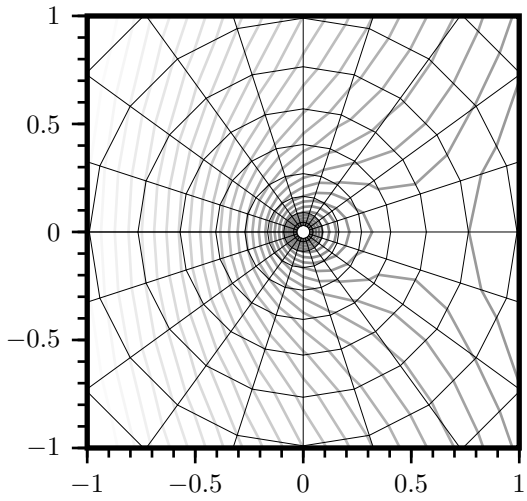


Figure 8: Sample graph created by MetaContour and MetaPlot, showing potential lines for a combination of a linear gradient and a point source, plotted on a polar grid.

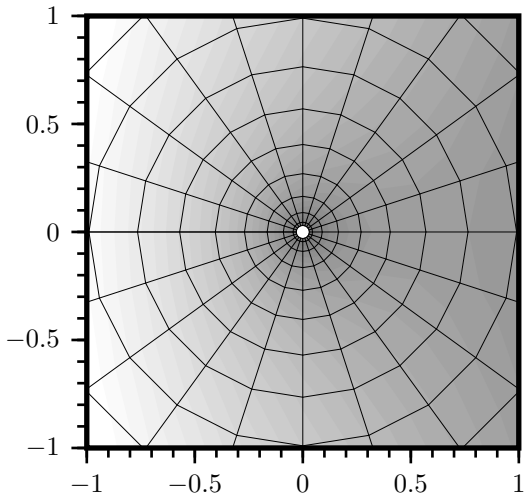


Figure 9: Another sample graph created by MetaContour and MetaPlot, illustrating a filled contour-plot style rather than using contour lines.

if we wanted to plot the contour lines all in black, we could do so simply by specifying:

draw *continstA.plot*(*LinePlot*) **withcolor** black;

Aside from the color contour-line plot just described, the MetaContour output contains a filled contour plot, and an image of the mesh of data points. Some examples of these are shown in Figure 8 and Figure 9; although these are much more complex than the examples from preceding sections, the MetaPlot commands to generate them are nearly identical.

4 Conclusion

The examples that have been shown illustrate only a small sampling of the capabilities of MetaPlot. In using METAPOST to generate the figures, it provides an easily extensible layout capability that is not limited by the imagination of the package author. The standardized plot-object interface simplifies the process of writing plot-generation programs, as they can leave the details of layout and annotation to the MetaPlot postprocessing.

MetaPlot and MetaContour are available from CTAN in the `/graphics/metaplot` directory. They are still very much works in progress; I look forward to suggestions and improvements, and hope that others will find them to be useful tools.

References

[1] Hwang, A., ePiX, `http://mathcs.holycross.edu/~ahwang/current/ePiX.html`.

[2] Gnuplot, `http://www.gnuplot.info`.

[3] Phan, A., m3D, `http://www-math.univ-poitiers.fr/~phan/m3Dplain.html`.

[4] Batchelor, G. K., *An Introduction to Fluid Dynamics*, Cambridge University Press, 1967.

TEX and Scripting Languages

William M. Richter
Texas Life Insurance Company
900 Washington Avenue, Waco, TX 76703, USA
wrichter@texaslife.com

Abstract

TEX is an ASCII text-based markup language. In a scheme of automated document preparation TEX provides the foundation. The idea is for programs to do the work of 1) generating the TEX code for documents, 2) running TEX on these documents, and 3) post-processing the resulting `.dvi` files to obtain the finished documents. Resulting PostScript documents may be further post-processed to produce files that exploit the output capabilities of various printers. Discussed herein are the techniques and benefits of such a scheme and how scripting languages (those languages outside the traditional edit/compile/link/run cycle) can make the whole process fun and easy.

1 Introduction

In his web essay *Hackers and Painters* [2], Paul Graham equated the much maligned and misunderstood activity of "hacking" [6] with the long-esteemed tradition of painting (e.g., portrait painting, as opposed to painting of porches, peeling house trim, and such). He observed that what, today, we acknowledge as masterworks actually evolved during the artist's act of creation from a sketch, the details only gradually being filled in, to a finished, glorious work of art. He argued that a writer goes through the same process of refinement, starting from rough outline or foggy idea until she finds nothing which needs refining. One reason TEX is appealing to authors is that it makes the process of refinement secondary. The tasks of *creation* (thinking is hard work for most of us) and *presentation* are orthogonal. Moreover the presentation task is assumed almost entirely by TEX.[1] One can, after all, create a TEX document that is 90% complete using nothing more than a tool as simple as NotePad. The implication being that simple tools equate to less loss of creative energy.

Graham believes that authors of computer code (programmers, we often call them) follow the same nonlinear/circuitous paths of painters and authors. Seldom, if ever, is software conceived of and implemented by following in a direct route from beginning to end. Most great software, Graham claims, is the product of hacking, that the implications for software design are significant, and that what a computer language is and how an author interacts with it defines the end result. In his view it means ...

> ... a programming language should, above all, be malleable. A programming language is for thinking of programs, not for expressing programs you've already thought of. It should be a pencil, not a pen.

And he continues,

> We need a language that lets us scribble and smudge and smear, not a language where you have to sit with a teacup of types[2] balanced on your knee and make polite conversation with a strict old aunt of a compiler.

A class of programming languages, called "scripting languages", is compatible with Graham's ideas of what a hacker's language should be. "Malleable" in nature, and easy to *think with*, scripting languages are similar in spirit to TEX. Indeed, TEX itself may even be considered as a scripting language for typesetting.

So, on the one hand, we have TEX, a tool which lets authors "scribble and smudge and smear" about with their ideas. On the other hand we have hackers using scripting languages pursuing similar creative avenues. The question then arises, "What happens if these two tools are combined and used in a collaborative effort?" We now explore various ways that TEX and scripting languages can be combined.

[1] Except when we TEXnicians decide we know better and begin to muck around in TEX's own internal affairs.

[2] For readers unfamiliar with the art of computer programming, the "teacup of types" to which he is referring will be addressed in a subsequent section on the attributes of scripting languages, where static vs. dynamic data types are discussed.

2 Scripting Languages

Before delving into scripting languages proper, let us review a few of the attributes of traditional computer languages (the ones compiled with Paul Graham's strict old aunts).

3 Traditional Computer Languages

For readers unfamiliar with the art of authoring computer software (programming computers), here is what programmers do: they think of a task that computers can accomplish better than humans (say, typesetting, for example). Then they sit and think, potentially at length, about how humans would go about doing that task, and how to express those steps algorithmically [3]. After sketching said algorithm, they formalize and codify it in a so-called "language" that is a sort of half-way meeting ground between the way humans think and the way computers operate. This prose, called a program, consists of two distinct entities: variables, which declare what it **is** that the computer will be working on, and imperative procedures that define what is to be **done** to that data.

Some salient details about these traditional languages:

1. The variables: Computer hardware can work with data in different formats: numbers (integers and real numbers), strings of character data, etc. Each variable in a program must be defined in advance of its use to be of a specific type. In computer science lingo this is called *static typing*.
2. The code: Codifying an algorithm in a particular computer language isn't really enough for computer hardware. More work must be done. This language must be *compiled* by Graham's "aunt" into "machine code" on which the computer's logic circuits can act.
3. But even the work of the compiler-aunt isn't enough. The fruit of her strict dominance must then be *linked* with the work of other compiler-aunts to produce a final collection of unreadable "goo" that only a computer can understand (machine code is unreadable to all but the most deviant of human brains).
4. Nor is this the end of the story. When an edited/compiled/linked program (called an executable) has finally been produced and a blazingly fast 3-gigahertz CPU is unleashed to execute it the first time, the most likely end result is either an almost immediate decision by the CPU that its human programmer is capable only of producing flawed code for it to execute (it communicates this fact by printing some rude message like **"Segmentation Violation"** and producing a huge file on disk containing the entire contents of its memory), or it lapses into a seemingly semi-comatose state consuming large amounts of CPU time until its programmer/master gets its attention with the violence of the `kill` command.

One can see a definite "cycle of pain": *Edit, Compile, Link, Test* that must be repeated many times until a flawless executable is produced. No wonder computer programming is seen by many an outsider as a black art to be pursued by only the most intrepid and determined souls.

4 Why Scripting Languages are Better, and Why More People Should be Hackers

Scripting languages [9] shrink the cycle of pain to *Edit, Test*. With the crufty old compiler-aunt gone, the whole process of software development proceeds in a more efficient and pleasant manner with attention shifting to the "creative", editing part and the refinement, or testing part. But measure of pain is not the only attribute that makes scripting languages attractive. Other important attributes are:

1. Simple syntax;
2. High-level data types;
3. Loosely typed;
4. Standard control structures: if/else, while, for;
5. Interfaces well with host operating system;
6. Plays well with external entities;
7. Embeddable inside more complex systems;
8. Often used as "glue" languages to link multiple standalone applications and tools together;
9. Requires a runtime interpreter to execute the script;
10. Compiles to bytecode which executes on a virtual machine;
11. Often 'dynamic' in nature.

We need to expound on a few of these points.

4.1 Simple Syntax

If a language is to satisfy Graham's requirement that it be a malleable pallet for the smearing and smudging of ideas, it cannot be verbose (we don't want to spend time typing). So scripting languages (henceforth SLs — I'm tired of typing, too) are succinct in nature; able to convey a significant amount of procedural instruction in as few words as necessary to maintain clarity of meaning.[3]

[3] For programming language scholars, the language APL may come to mind, but perhaps not *that* succinct. It would

4.2 High-level data types

The concept of high-level data types parallels simple syntax. Just as we need to state procedural algorithms in a succinct fashion, we also need constructs that allow for the representation of bundles of data that may be arbitrarily complex. We demand more than simple integer, floating point, and strings of character data that traditional languages like C and C++ provide.[4]

Usually these higher-level data types come in the form of lists and dictionaries; containers that hold other data elements and allow for the expression of relationships between our data.

4.3 Loosely Typed / Dynamic Nature

Discussion of esoteric topics like *Strongly vs. Loosely Typed Data* and *Early vs. Late Binding* is more than can be discussed here (see [1]). Some understanding is essential, however. Earlier, we pointed out that in traditional languages, each element of data that a program will use (its variables) must be defined to exist as a particular type before it can be used.[5] Moreover, as variables are passed between parts of a program (function calls) the type of each variable passed must match exactly the type expected by the called function. This check is done by the strict old compiler-aunts, and was designed to keep programmers from making errors that would only manifest themselves during the test phase. Strict type checking makes a lot of sense with traditional languages.

However, with dynamic SLs, there is a critical difference, rooted in the 'dynamicness' of the language. SLs need not declare variables in the first place. Variables are created or 'allocated' (on-the-fly, as it were) when they are first referenced. When a variable is allocated it is associated with a particular type that is implied from the context in which it was initially used. The association to type is permanent and observable. So not only can one ask, "What *value* does a variable contain?", one can also make an inquiry about its *type*.

For example, the statement `A = 123` allocates a data element called `A` whose value is 123 and whose type is integer. The statement `B = 3.14` allocates a variable called `B` whose value is 3.14 and whose type is floating point. `B` was made a variable of type floating point because, contextually, the statement contained a decimal point in the value implying a floating point value. Had we desired `A` to be a floating point variable we would have coded `A = 123.0`.

This leads to a new world of ways in which to think about writing code. Functions, now dynamic in nature, can easily accept an arbitrary number of arguments, the type of each being any of a range of possible types. Depending on the number and type of variables passed to a function, the function may act in different ways. This goes to the heart of malleability. In the creative process if we change our mind and decide to "smudge and smear" in a different direction, our existing code may not go to waste. It may be possible just to extend it to conform to our new conditions.

A world of new and easier programming languages, the SLs, may also introduce hacking to a wider audience. Whereas the "old world" traditional languages excluded or intimidated many people for the reasons above (there are, after all, only so many work hours in a day), SLs remove the complexity of programming and make hacking the creative process that it should be.

Finally, there is another reason more people (at least for those who must live with a computer) should become hackers. While most of us are not master software developers, developing cathedral-size financial accounting packages, for example, we do a surprising amount of "sketch" work (in Graham's paradigm) and having skills to write small programs can be effective.

5 Real Scripting Languages

A mid-June 2004 *google-search* for the keywords

```
script language programming
```

returned approximately 1,570,000 hits. Top-ranked pages returned from a search of keywords `scripting languages` reside on the sites:

1. `www.php.net`
2. `www.python.org`
3. `www.ruby-lang.org`
4. `www.perl.org`

All these websites are homes of important scripting languages. And there are more SLs; many more . . . a veritable zoo, with names like: Awk, JavaScript, Lisp, Lua, Perl, PHP, Python, Rebol, Ruby, Small, Groovy, Tcl. If one were to rank SLs in order of popularity, the top of that list would include:[6]

- Perl
- Python

be nice for non-hackers to be able to read and understand our prose, too.

[4] Admittedly, C, C++, and other traditional languages may be made to represent arbitrarily complex data, but those types are not intrinsic in the language.

[5] This isn't actually true. Data elements may be dynamically allocated in traditional languages, but this introduces additional complexity in both the design and debugging steps.

[6] Not listed in order.

- Tcl/Tk
- JavaScript
- Unix shell scripts (`sh`/`bash`/`csh`/etc.)

Several of these SLs have outgrown their scripting origins and have gone on to become "general purpose programming languages of considerable power" [5]. The only argument for continuing to use the term "scripting language" is the lack of a better term.

6 A Particular Scripting Language: Python

Chapter one of the official Python Tutorial reads:[7]

> Python is simple to use, but it is a real programming language, offering much more structure and support for large programs than the shell has. On the other hand, it also offers much more error checking than C, and, being a very-high-level language, it has high-level data types built in, such as flexible arrays and dictionaries that would cost you days to implement efficiently in C. Because of its more general data types Python is applicable to a much larger problem domain than Awk or even Perl, yet many things are at least as easy in Python as in those languages.

The tutorial continues to highlight these important attributes:

1. It has a modular architecture so that code developed for one application can be reused in other programs. Likewise, it comes with a large number of built-in modules for things like file I/O, system calls, sockets, and many common Internet protocols (FTP, HTTP, SMTP, etc.).
2. It is an interpreted language conforming to the edit/test cycle discussed previously.
3. Its interpreter can be used interactively, making it easy to experiment with features of the language, or to test code before actually running a program (we'll see an example later, in fig. 11).
4. It has a high-level syntax that allows for writing compact, readable programs.
5. It has high-level data types allowing for expressions of complex data relationships.
6. It is object-oriented [10], but does not require the use of those object-oriented features, or O-O programming skills to use the language.
7. Statement grouping is done by indentation instead of explicit begin/end brackets.
8. It is extensible: if you know how to program in C it is easy to add a new built-in function or module to the interpreter, either to perform critical operations at maximum speed, or to link Python programs to libraries that may only be available in binary form (such as a vendor-specific graphics library).
9. It is embeddable: You can link the Python interpreter into an application written in C and use it as an extension or command language for that application.

An excellent first book for readers unfamiliar with but interested in learning Python is Mark Lutz's *Programming Python* [4].

Finally, the tutorial enlightens us regarding the name:

> . . . the language is named after the BBC show *Monty Python's Flying Circus* and has nothing to do with nasty reptiles. Making references to Monty Python skits in documentation is not only allowed, it is encouraged!

7 Combining Python and TEX

There are a number of ways in which to combine TEX and Python to automatically produce documents. If one considers the amount of "work" necessary to produce a document as fixed, then that work can be allocated partly to TEX and partly to Python. One can then imagine a scatter diagram with X and Y axes that represent, for any possible scheme, the amount of work allocated to Python and TEX, respectively. Such a diagram is illustrated in fig. 1.

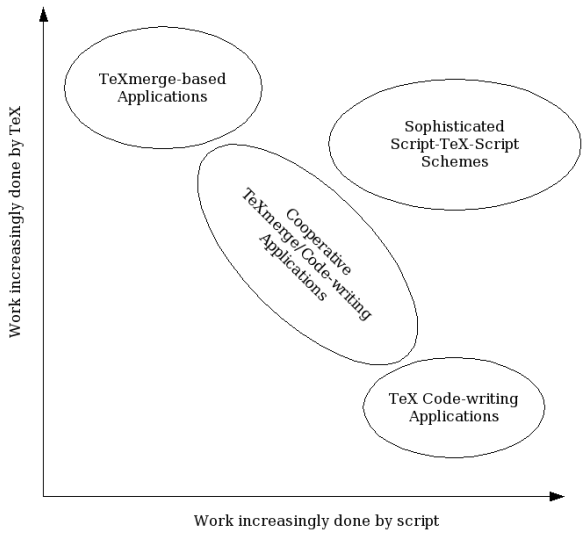


Figure 1: Application Domains of Python/TEX integration.

[7] `www.python.org`

The diagram shows several different "application domains", defined by which component (TEX or Python) receives the most development effort, or places the most demands on computing resources. These domains allow us to classify various approaches to Python/TEX integration.

7.1 The Imperative Approach

Imagine writing a Python script that produces a file of TEX code by executing a series of `write` statements as in fig. 2 and then runs TEX and `dvips` on that file. Here the emphasis is clearly all on the Python script and the details of how the TEX code is to be produced; we know TEX will dutifully do its job if it is provided good code. Applications of this nature we call *imperative*, and occupy the lower right region of fig. 1.

Figure 2: Imperative TEX code-writing script.

```
#!/usr/bin/env python
import sys
import os
f = open('MyDocument.tex', 'w')
f.write('\\nopagenumbers\n')
f.write('This is my first \\TeX\\ document \
    produced from a script.\n')
f.write('\\vfil\\eject\\bye\n')
f.close()
os.system('tex MyDocument.tex')
os.system('dvips MyDocument')
print 'Done.'
```

This technique is the simplest way to integrate Python and TEX,[8] and is surprisingly effective. Although the example in fig. 2 is trivial, the imperative technique can be used in applications where documents are assembled from a large *database* of text "snippets". Logic in the Python script provides the "smarts" that determine what snippets to select and how to arrange them for presentation to TEX. More logic and scripts of increasing complexity push the application further to the right on the X-axis in fig. 1.

7.2 Using m4

A slight increase in sophistication (but still remaining near the X-axis of fig. 1, is to employ the macro processor program, m4.[9] m4 [8] is an elaborate search-and-replace engine for text. For example, given the text:

```
Hello, NAME, today is DATE.
```

If we present that text to m4 as input with the following command-line:

```
m4 -DNAME=Sally -DDATE='22-June-2004'
```

the output from m4 would appear as:

```
Hello, Sally, today is 22-June-2004.
```

Now we can play the same game as in the imperative approach, but with a new wrinkle: tags can be embedded in our text snippets. Once the TEX code is assembled, it is preprocessed through m4 and *then* presented to TEX. Here are the steps:

1. Assemble TEX code from snippets of text,
2. Gather data for tag-replacement from a data source,
3. Build m4 command line with `-Dname=value` arguments for each unique tag in the TEX file,
4. Execute the command just built and save the output,
5. Present the saved output to TEX.

8 TEXmerge

We now move away from the X-axis of fig. 1.

The m4 approach introduced an important concept: the idea of *template* files. There exist a large class of applications whose function is to produce, for lack of a better term, "form letters".[10] The m4 technique of the previous section lends itself precisely to this *merging* application: Build a `.tex` file with tag names, then repeat steps 2–5 above until end of data. The end result will be a stack of form letters ready to print and drop in the mail.

While m4 is an efficient macro-replacement engine, we know of another engine that eclipses it: TEX. Consider the TEX document in fig. 3.

Figure 3: `form.tex`: A merge-ready TEX file.

```
\nopagenumbers
This is my first \TeX\ document produced
from a script.
\par
Hello, \NAME, today is \DATE.
\vfil\eject
```

Alone, this file will result in undefined macro references because the macros `\NAME` and `\DATE` are not

[8] The other simple extreme would be to prepare an entire document by hand-editing and then have Python run TEX on that file. Quite uninteresting.

[9] Quoting from the m4 manual page: "The m4 utility is a macro processor that can be used as a front end to any language (e.g., C, ratfor, fortran, lex, yacc) ..." and now, TEX!

[10] Every technological advance seems to bring with it a raft of nastiness. With email comes spam, with computer-aided printing comes the dreaded form letter. At least with TEXmerge, they can be beautiful form letters.

defined. However, when used in conjunction with the Python script in fig. 4, it works beautifully.

Figure 4: Imperative TEX code-writing script relying on TEX's macro replacement facility.

```
#!/usr/bin/env python
import sys
import os
f = open('temp.tex', 'w')
f.write('\\def\\NAME{Sally}\n')
f.write('\\def\\DATE{22-June-2004}\n')
f.write('\\input form.tex\n')
f.write('\\bye\n')
f.close()
os.system('tex temp.tex')
os.system('dvips temp')
print 'Done.'
```

Scripts like 4 can be represented schematically as in fig. 5. It is important to note that in this scheme we are dealing with *two* (or more) `.tex` files: 1) The template file(s) containing the structure of our form letter(s) (more than a single type of form letter can be produced in a single run simply by inputting different template files), which have tags where merge variables are to be inserted, and 2) the temporary file which defines macros for the merge variables and has input commands to bring in the templates. Inside the temporary `.tex` file there can be many occurrences of the `\def`... and `\input`... lines; one occurrence for each letter to be produced.

8.1 TEXmerge API

The technique illustrated in fig. 4 works well. Data for the merge variables can be arbitrarily long, for example, and TEX will 'do the right thing' and wrap the merged text into our form, etc. But there are problems:

1. The biggest problem is data containing tokens having special meaning to TEX. If our merge data contains `$,%,&`, etc., we have a problem,
2. It's rather tedious to read the script, and we find ourselves repeatedly re-implementing this tedious code for every application.

The whole process of opening the temporary `.tex` file, protecting sensitive tokens, preparing the `\def` lines for the merge variables, doing the `\input`..., executing TEX and the DVI backend need to be formalized inside an application programming interface (API).

We call that API "TEXmerge". It was first presented [7] as a C-language API with a Python extension wrapper module. Since that time, the API has been re-written in pure Python and is presented here (see appendix A for a technical description of the API).

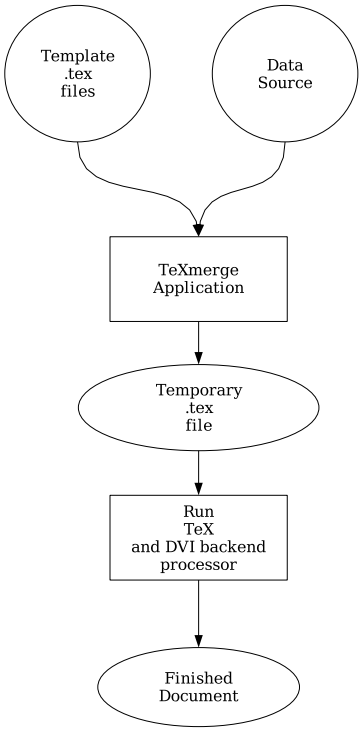


Figure 5: Schematic overview of document production via the TEXmerge API.

First, an example using the TEXmerge API (the TEXmerge *module*). Fig. 6 re-implements the script presented in fig. 4 using the module-level interface:

Figure 6: A simple Python script using the TEXmerge module-level API functions.

```
#!/usr/bin/env python
import sys
import os
import TeXmerge
f = TeXmerge.openOutput('temp.tex')
mergeVars = {'NAME': 'Sally',
             'DATE': '22-June-2004'}
TeXmerge.merge('form.tex', mergeVars)
TeXmerge.closeOutput(f)
TeXmerge.process('temp.tex', 'dvips')
print 'Done.'
```

Note the following:

1. Access to the TEXmerge module is provided via the import statement: `import TeXmerge`.

2. The native Python `open/close` calls have been replaced with calls to `TeXmerge.openOutput()` and `TeXmerge.closeOutput()`.
3. Merge variables are formally presented to the API as a Python dictionary object.
4. The `merge()` call takes care of protecting sensitive tokens in the merge data that would otherwise confuse TEX.
5. The `os.system()` calls have been replaced with `TeXmerge.process()`.

Finally, Python is an object-oriented language, so the TEXmerge module also offers a TEXmerge *class*. Fig. 7 re-implements fig. 6 using the object-oriented interface:

Figure 7: A simple Python script using the TEXmerge object-oriented interface.

```
#!/usr/bin/env python
import sys
import os
import TeXmerge
mergeObj = TeXmerge.TeXmerge('temp.tex')
mergeVars = {'NAME': 'Sally',
             'DATE': '22-June-2004'}
mergeObj.merge('form.tex', mergeVars)
mergeObj.process('dvips')
print 'Done.'
```

9 Going Further with Macros

Now it is time to move up the Y-axis of fig. 1, focus attention on the TEX domain and investigate what benefits can be gained by writing specialized macros to enhance integration with TEXmerge.

9.1 Do-Nothing Macros

The first class of macros to be considered is the "do-nothing" macros. These macros, from TEX's view, evaluate to `\relax`. They exist in a TEXmerge template file in order to communicate information to a Python script which scans the template file. A more traditional method used to communicate information to an external entity would be to embed that information in comment strings within the file. Writing first-class macros, however, seems to produce a cleaner, readable file, and is more flexible since a do-nothing macro could, in the future, be turned into a "*do-something*" macro.

9.1.1 Classic Merge Variable Declarations

Do-nothing macros were introduced in the first release of TEXmerge, with the `\texmergevar` macro. Just looking at a merge-ready template `.tex` file, it is not immediately clear what the names of all the merge variables are. `\texmergevar` allows the author of the template file to explicitly state the names of all merge variables that will be referenced in the file by coding:

`\texmergevar` *name*

for each merge variable. The TEXmerge module has a module-level method, `getNames`, which scans a passed `.tex` file name (and recursively any included files) and returns a list of all declared variable names. Python scripts can inspect TEX template files and determine the names of all declared merge variables.

9.1.2 Extended Merge Variable Declarations

Several years' use of the TEXmerge API has shown that document-producing applications could be made more robust if a template `.tex` file could specify precisely what *values* a merge variable should contain. The need for merge variables to take on only one value from a small set of possible values stems from the use of conditional TEX code, via the `\ifx` control sequence, etc. Conditional typesetting is powerful because it allows documents to become intelligent. *A single `.tex` source file can produce entirely different finished documents by testing the value of merge variable(s) and typesetting text accordingly.*

A life insurance company, for example, falls under the jurisdiction of every state in which it is licensed to conduct business. A document, say a "sales practice guide", often must contain language mandated by a particular state. Sales practice guides for forty different states may have 90% of their language in common, but each may also have unique state-specific language that none of the others contains. Having a single, intelligent source file, `salesPracticeGuide.tex`, lowers the cost of change management substantially; changes made to shared text need only be made *once*.

The do-nothing macro `\texmergevardef` defines merge variables with extended attributes, like this:

`\texmergevardef[`*attrName=attrValue,...*`]`

Attributes of the merge variables that can be specified are:

- `name` = the name of the merge field.
- `type` = the type of merge field. The intended use of this attribute is to convey a recommended style of data entry element for graphical (GUI) applications. Valid types are:
  - `entry`: a simple text entry field,

Figure 8: A sampling of extended merge variable declarations.

```
\texmergevardef[name=ISTATE, type=optionmenu,values=TX|OK|AZ|CA|OR|WA,descr=Issuing state]
\texmergevardef[name=ONAME,type=entry,descr=Owner name]
\texmergevardef[name=APPTYPE,type=radiobutton,values=1|2|3,labels=Employee|Spouse|Child,
                descr=Applicant type]
```

Figure 9: Result of `getExtendedNames()`: a Python dictionary of field-attribute dictionaries

```
{'ISTATE': {'name': 'ISTATE', 'type': 'optionmenu', 'values': ('TX', 'OK', 'AZ', 'CA', 'OR', 'WA'),
'descr' : 'Issuing state'}, 'APPTYPE': {'name': 'APPTYPE', 'type': 'radiobutton','values':
('1', '2', '3'), 'labels': ('Employee', 'Spouse', 'Child'), 'descr': 'Applicant type'}, 'ONAME: {
'name': 'ONAME', 'type': 'entry', 'descr': 'Owner name'}}
```

  - `text`: a multi-line text entry field,
  - `toggle`: a toggle button field,
  - `optionmenu`: a drop-down option menu of choices,
  - `radiobutton`: a set of mutually-exclusive toggle buttons.
- `values` = a list of valid values for the variable, separated by `|`'s.
- `labels` = a list of alternate labels that should be associated with the `values` attribute for display purposes. Used with the `toggle`, `optionmenu`, and `radiobutton` field types.
- `descr` = a description of the merge variable.

The TEXmerge module-level function `getExtendedNames` extracts these extended merge variable definitions, parses them, and returns them in a dictionary (keyed by the `name` attribute's value) of field attribute dictionaries.[11] Fig. 8 shows an example `.tex` file with extended merge variable definitions. Fig. 9 shows the return value from applying `getExtendedNames` on that file.

9.1.3 Named Text Blocks

Another class of applications has the need to share identical text between two markup languages: TEX and HTML. Here it is *language within the document* that needs to be identical (for legal reasons, say) and not the structure of the document that is constant between the two presentation platforms. Indeed, structure of the printed TEX document may be substantially more complex than its briefer, lightweight, HTML cousin. How can the common text be shared between the markup languages?

One way is to make the TEX document "own" the text. It declares, via a set of macros, where the common blocks of text begin and end. We refer to these blocks as *named text blocks*. The demarcation macros look like this:

- `\StartNamedTextBlock[`*attrName*=*value*...`]`
  Text block attributes are as follows:
  - `name` = Name of the text block,
  - `seq` =*Integer*; several sections of text can be assigned the same name, but with unique sequence numbers. The extracted text will be a concatenation of like-named blocks, ordered by sequence number,
  - *subkey* =*subvalue*: See the text for full discussion.
- `\StopNamedTextBlock`

Once text boundaries have been marked and named with these macros, the text can be extracted and used by the HTML producing part of the application. The TEXmerge module provides a module-level function, `getNamedTextBlocks`, to extract the named text blocks, and two helper classes `TextBlock` and `TextBlockManager` to make accessing the extracted blocks simpler.

We explain the functional use of named text blocks by way of the example file in fig. 10 and the interactive Python interpreter session shown in fig. 11.[12]

Note the following:

1. The block demarcation macros are essentially invisible to TEX and have no effect on typesetting.
2. The `TextBlockManager` class is used to extract the named blocks. One simply passes a pathname to the `.tex` file containing named text

[11] `getExtendedNames` also detects occurrences of the prior `texmergevar` macro and treats them as extended merge fields having an attribute `type` = `entry`.

[12] About the interactive interpreter session: >>> is the interpreter's prompt. Text appearing after that prompt was entered by the user. Python's response appears on the line immediately below the prompt input line.

Figure 10: TEX file `test.tex` containing four named text blocks: B1, B2, C1, D1.

```
This is a test document containing \textit{named text blocks.}
\StartNamedTextBlock[name=B1]
This is the first block.
\StopNamedTextBlock
Now for a second block:
\StartNamedTextBlock[name=B2]
Second block
\StopNamedTextBlock
Now for a series of sequenced blocks \ldots
\line{\hbox{\StartNamedTextBlock[name=C1,seq=1]C1.Left\StopNamedTextBlock}\hfil
      \hbox{\StartNamedTextBlock[name=C1,seq=2]C1.Right\StopNamedTextBlock}
}
Finally, a named text block having a subkey:
\StartNamedTextBlock[name=D1,istate=TX]
This text is specific to the state of Texas.
\StopNamedTextBlock
```

Figure 11: Interactive Python interpreter session. Working with named text blocks.

```
[hawkeye2:~/sftug] williamr% python
Python 2.3.2 (#1, Nov  6 2003, 13:18:07)
[GCC 2.95.2 19991024 (release)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import TeXmerge
>>> o = TeXmerge.TextBlockManager('test.tex')
>>> o
<TeXmerge.TextBlockManager instance at 0x750648>
>>> o.getBlockNames()
['C1', 'B1', 'B2', 'D1']
>>> b1 = o.getBlock('B1')
>>> b1
<TeXmerge.TextBlock instance at 0x72b5d0>
>>> b1.getText()
'This is the first block.'
>>> c1 = o['C1']
>>> c1.getTextSegments()
{1: 'C1.Left', 2: 'C1.Right'}
>>> c1.getText()
'C1.Left C1.Right'
>>> d1 = o['D1']
>>> d1.getSubkeys()
['istate']
>>> d1.getSubkeyValues('istate')
['TX']
>>> d1.getText('istate','TX')
'This text is specific to the state of Texas.'
```

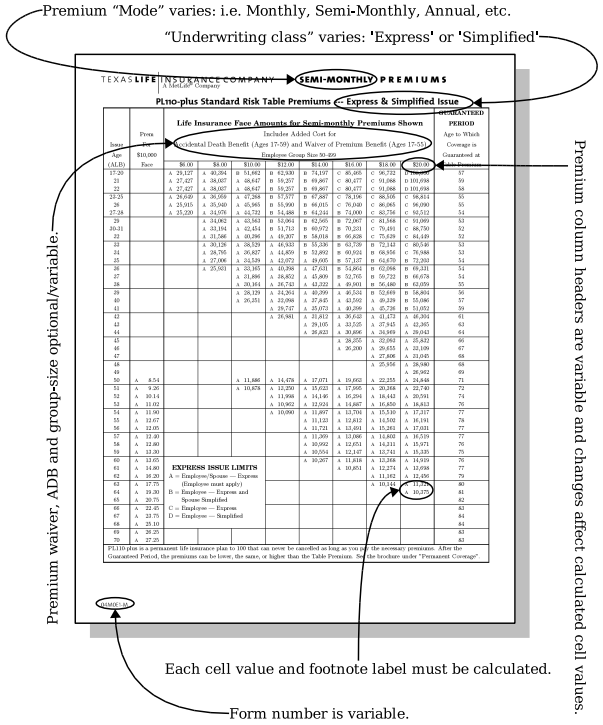


Figure 12: Complex document produced by Hybrid Script-TEX-Script scheme.

blocks in order to instantiate an object of the `TextBlockManager` class.

3. The names of all the text blocks in the file can be retrieved by calling the manager object's `getNamedTextBlocks` method.
4. Individually named text blocks are retrieved via the manager object's `getTextBlock` method, or simply by indexing the manager using the name of a text block as the index key (as was done for block C1 in fig. 11). Either operation will return a `TextBlock` object.
5. Access to the text of a `TextBlock` object is via its `getText` method.

9.2 Do-Something Macros

9.2.1 Hybrid Script-TEX-Script Scheme: A Case Study

If we have an application where a substantial amount of the document's content may vary, the merge paradigm of TEXmerge begins to break down under the complexity of so many variables. This is especially true of variable tabular data.

Example: The annotated page shown in fig. 12 is a rate sheet of life insurance premiums. As the figure shows, there is more variable data than static text on the page. The rate sheet, however, is only one page of a twenty page document. Other pages in its parent document also have variable data, and state-specific language, as well. Overall the document's nature fits well in the TEXmerge scheme; the rate sheet page is the "trouble maker". Another important consideration: the rate sheet needs to be embeddable in many other documents.

We desire a TEX macro as in fig. 13 that, when executed, magically produces a finished rate sheet.[13]

Figure 13: Rate sheet macro.

```
\MakeRateSheet[uwclass=express,
               mode=semi-monthly,
               groupsize=150,
               formno=test,
               waiver=yes,
               adb=yes
]
```

`\MakeRateSheet[...]` is definitely a *do-something* macro. The trick is to do as little work as possible in TEX and most of the *something* in a Python script. The work for TEX in this case is in two parts:

1. Gather macro arguments and marshal them into a Python script command-line, then execute the command with `\write18`.
2. Input and typeset the TEX code produced by the Python script.

We call schemes such as these *hybrid* or *Script-TEX-Script* schemes. The job of the secondary script (the one executed by TEX via `\write18`) is to act on arguments received from TEX, or from some other external source, do whatever calculations, etc., and output TEX code. The whole scheme is represented in fig. 14. Since the secondary script is unbounded by the complexity and amount of TEX code that may be returned, hybrid schemes are the ultimate in flexibility.

9.2.2 Document Template Macros

Document template macros also fall into the class of *do-something* macros. Another case study will serve as a description of their functionality. TEXmerge is in widespread use at Texas Life having applications in almost every major department, from Marketing, to New Business, to Policy Owner Service, to Computing Services. Several years ago, a graphic artist was hired to develop a new 'look-and-feel' for all printed material disseminated from the company. A

[13] Writing parameter based macros such as these is effortless with the aid of support macros found in Hans Hagen's ConTEXt macro package.

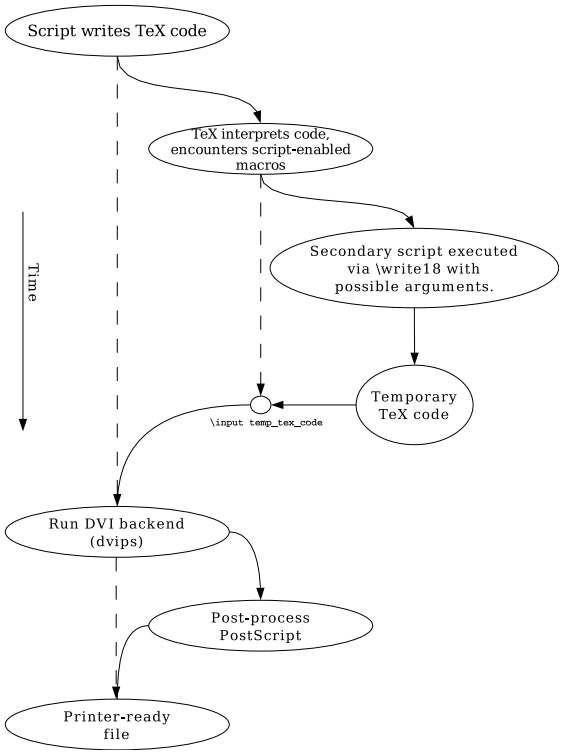


Figure 14: Schematic overview of document production via the hybrid technique.

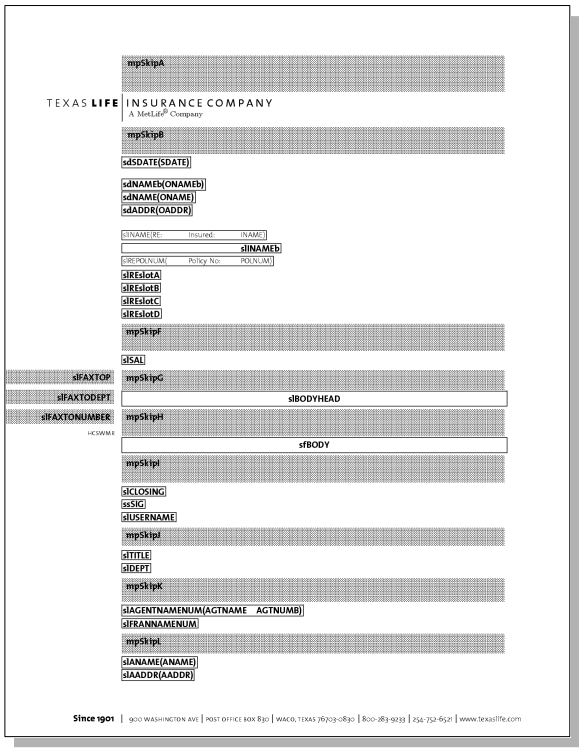


Figure 15: Template view for the client-letter macro.

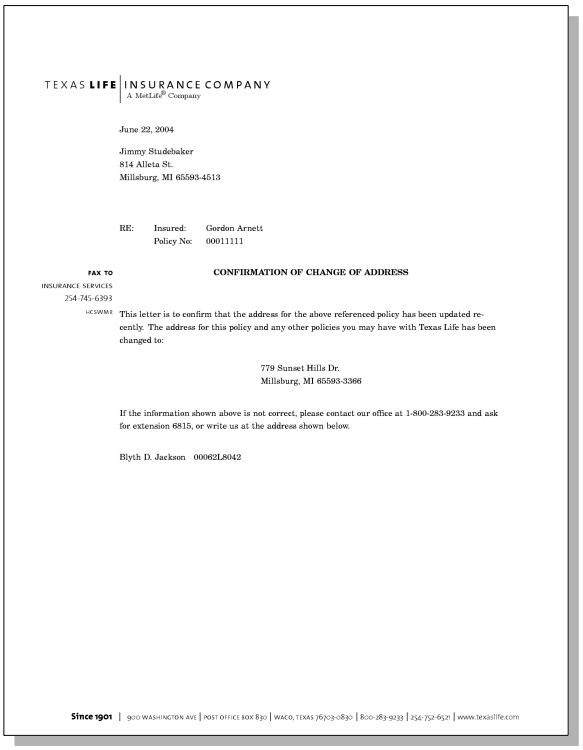
new graphics standards manual was written and all parts of the company were informed that compliance with the new standard was mandatory by a set date. This directly affected users of TEXmerge. The Policy Owner Service department, for example, had 600+ TEXmerge-based form letters used daily for corresponding with clients. Compounding the problem were the non-standard fonts and a peculiar format to which standard letterhead should conform: a wide left margin, except for various items that were to remain left hanging, right-justified. How could over 600 documents be quickly converted to this new format? Language inside the documents could remain unaltered; only the structure was changing.

Serendipitous earlier decisions, made when originally planning and setting up the TEXmerge letters, made conversion to the new graphics standard straightforward. The serendipity was in a decision to separate the text for the body of each letter into its own `.tex` file. That being the case, all that was needed was a mechanism to enforce the policy of the graphics standard; a way to automatically produce the required layout of the document. This we do with so-called *template* macros. Fig. 15 shows the structure enforced by the `\StartClientLetter` macro. Based on a *plug-and-socket* model, it relies heavily on macro parameters (almost all having default values), as can be seen in the figure. Template macros classify parameters into three categories:

- Simple parameters: parameter names beginning with `mp`,
- Data sockets: parameter names beginning with `sd`,
- Slots: parameter names beginning with `sl`.[14]

The `mpSkip...` parameters (gray strips shown in fig. 15) can be specified to alter whitespace. Merge variable data is connected to a template using a *plug-and-socket* model. Merge variable names are termed *plugs* and the `sd...` macro parameters are termed sockets. One plugs a variable to particular position on the letter by equating the name of the plug with the desired socket name. The socket names are shown on the template letter in fig. 15 with default plug values in parenthesis. Finally, *slots* are macro parameters that can accept arbitrary TEX code as arguments.

[14] There are two other prefixes: `ss`, related to insertion of digitized versions of handwriting signatures; and `sf`, related to input files.



TEXAS LIFE | INSURANCE COMPANY
A MetLife® Company

June 22, 2004

Jimmy Studebaker
814 Alleta St.
Millsburg, MI 65593-4513

RE: Insured: Gordon Arnett
Policy No: 00011111

FAX TO
INSURANCE SERVICES
254-745-6393
HCSWM4

CONFIRMATION OF CHANGE OF ADDRESS

This letter is to confirm that the address for the above referenced policy has been updated recently. The address for this policy and any other policies you may have with Texas Life has been changed to:

779 Sunset Hills Dr.
Millsburg, MI 65583-3366

If the information shown above is not correct, please contact our office at 1-800-283-9233 and ask for extension 6815, or write us at the address shown below.

Blyth D. Jackson 00062L8042

Since 1901 | 900 WASHINGTON AVE | POST OFFICE BOX 830 | WACO, TEXAS 76703-0830 | 800-283-9233 | 254-752-6521 | www.texaslife.com

Figure 16: Sample letter produced using the client-letter macro.

The body of the letter can be supplied to the template macro in one of two ways:

1. Put the text of the body into a separate `.tex` file and pass the name of the file in the `sfBODY` parameter,
2. Code text of the body immediately after invoking `\StartClientLetter`. In this case the letter must end with `\FinishClientLetter`.

Finally, fig. 16 shows a sample letter produced from the `\StartClientLetter` macro.

10 Building GUI Applications with TEXmerge

So far, our discussion of TEXmerge has tended toward batch-style applications. The API is also effective in building GUI applications. The module's `getNames` and `getExtendedNames` functions provide useful *metadata* about merge fields, which can be used to construct user interfaces. Python is equally effective in programming GUI interfaces. The "Gimp Toolkit"[15] is especially easy to access from Python and provides a robust set of GUI interface components, including Pixmap buffers which, along with Ghostscript[16], can be used to effectively render PostScript.

10.1 TEXmerge — the Application

The TEXmerge API was originally developed for use in an interactive application, also called TEXmerge, for production of form letters. Originally written in C and based on the Motif toolkit, the current version is written in pure Python and is based on GTK+ 2.4. The application is arranged around *categories* of correspondence (collections of form letters, grouped by activity. Each activity category's letters are stored in a category subdirectory.

A sample TEXmerge main application window is shown in fig. 17. A category frame consists of the document selection window on the left, and a set of merge variable data entry fields on the right. A single set of input fields (a *record*), generates a single copy of the associated letter. Control buttons exist along the bottom to accomplish tasks such as adding new records, removing records, printing, and saving. A built-in PostScript viewer (not visible) is also provided to view the letter before printing or saving.

10.2 TEXtool

As long as we're writing GUI applications, why not write one that aids in the development of TEXmerge documents? TEXtool is an integrated development utility for editing, "TEX'ing", and viewing TEXmerge documents. Figs. 18, 19, and 20 are three successive views of the application, each view revealing one of the major notebook tab pages: Document, Editor, and Preferences.

Applications of this style exist that are more effective in general; however, TEXtool is unique because it is oriented especially for TEXmerge documents. It also shows the feasibility of integrating TEX into a non-trivial GUI application written in a scripting language. As can been gleaned from the figures, the Document tab displays the input frame of TEXmerge variables as they will appear in the normal TEXmerge application. The edit/test cycle can be quickly done all inside a single application window.

11 The Big Picture at Texas Life

As mentioned in the case studies earlier, TEXmerge is in widespread use at Texas Life. Fig. 21 is reproduced from [7]. It is a convincing illustration of how effective TEX can be as a document production engine, especially if combined with the right scripting

[15] `www.gtk.org` and `www.pygtk.org`.

[16] `www.ghostscript.com`

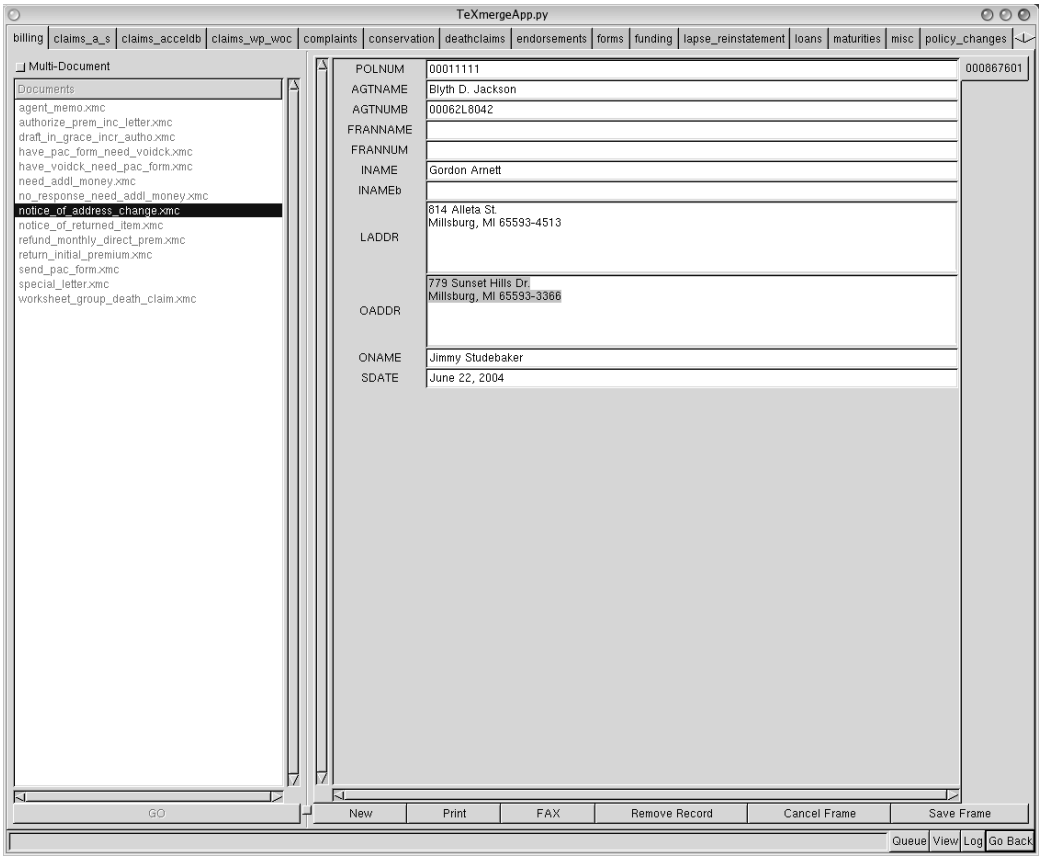


Figure 17: The TEXmerge application main window.

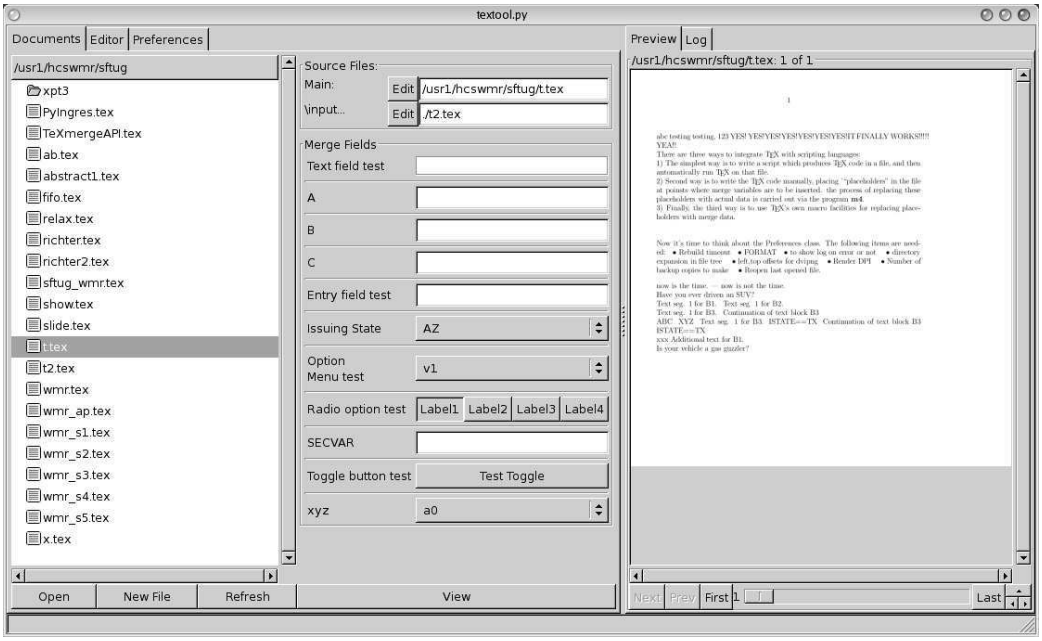


Figure 18: The `textool` application with the Documents tab visible.

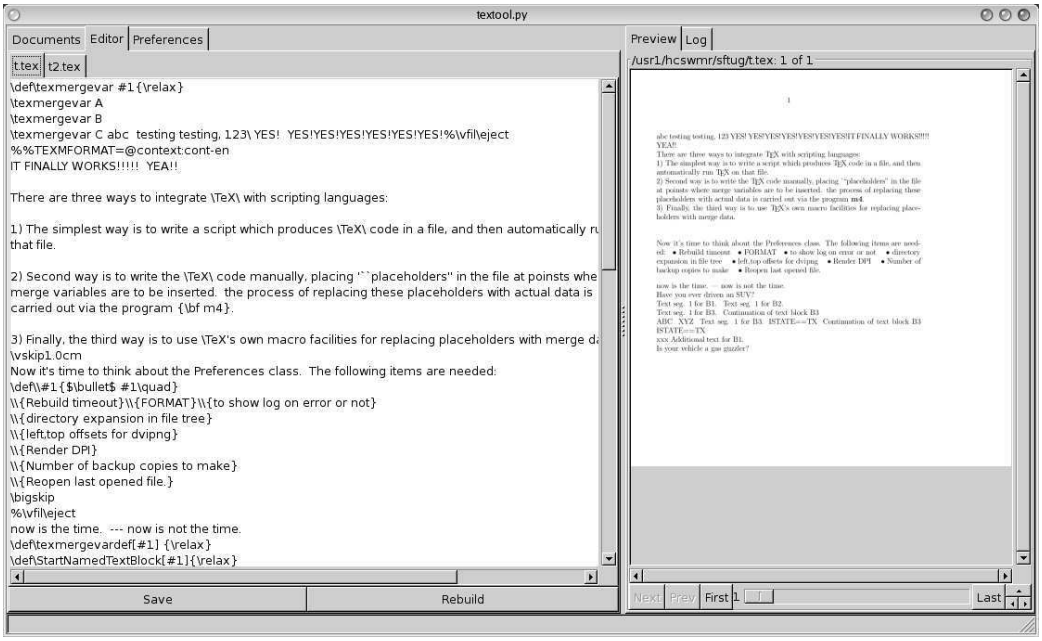


Figure 19: The `textool` application with the Editor tab visible.

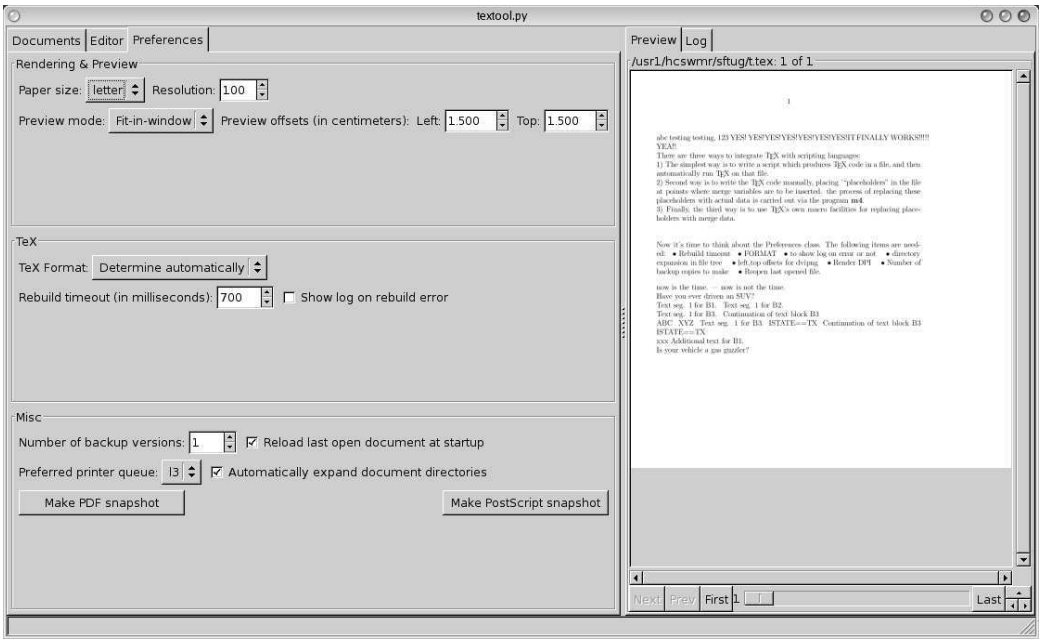


Figure 20: The `textool` application with the Preferences tab visible.

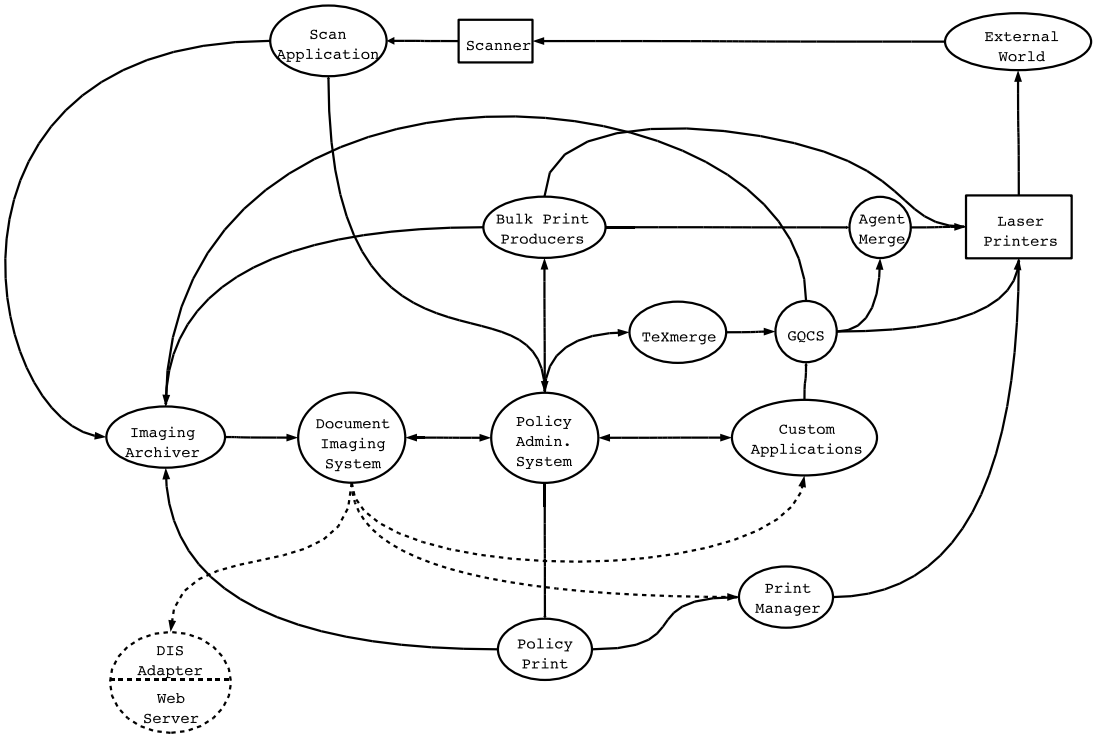


Figure 21: The big picture of TEXmerge at Texas Life.

language (Python). Most of the ovals in the figure use TEXmerge in some fashion. An important lesson learned is that once a facility like TEXmerge is available, the movement of documents between systems becomes much simpler. Only data required to build documents need be communicated along the arrows in the figure. Documents are only built and rendered when necessary for viewing or printing.

12 Conclusion

Because TEX is an ASCII text markup language, it is effective to write computer codes to process the TEX code for purposes other than typesetting. Scripting languages simplify writing these extraction codes. Embedding metadata into TEX files via simple macros allows the TEX author to communicate information to other computer applications. And, finally, using TEX alongside scripting languages in an automated document production environment provides flexibility and robustness to meet almost any demand imaginable. "Hacking" with scripting languages has never been simpler. Now is the time for more people to become *script literate*; the author encourages those with little or no programming experience to mix up a scripting language with their favorite TEX macro package.

References

[1] Bruce Eckel. Strong typing vs. strong testing. 2003. http://www.mindview.net/WebLog/log-0025.

[2] Paul Graham. Hackers and painters. 2004. http://www.paulgraham.com/hp.html.

[3] Donald E. Knuth. *The Art of Computer Programming*, volume 1. Addison-Wesley, third edition, 1997.

[4] Mark Lutz. *Python Programming*. O'Reilly and Associates, Inc., first edition, 1996.

[5] Eric S. Raymond. The art of Unix programming. 2003. http://www.faqs.org/doc/artu/ch14s01.html.

[6] Eric S. Raymond. The meaning of 'hack'. 2003. http://www.catb.org/~esr/jargon/html/meaning-of-hack.html.

[7] William M. Richter. Integrating TEX into a document imaging system. *TUGboat*, 22(3), 2001.

[8] René Seindal. GNU m4 development site. 2003. http://www.seindal.dk/rene/gnu.

[9] Unknown. Technical definition of scripting language. 2003. http://c2.com/cgi/wiki?ScriptingLanguage.

[10] Webopedia. What is object oriented programming? 2003. http://webopedia.com/TERM/O/object-oriented-programming-OOP.html.

Appendix A The TEXmerge Python API

A.1 How TEXmerge Runs TEX

Because there are a significant number of macro packages available as TEX formats, TEXmerge needs to be adaptable, both to the format to use, and to the way in which the TEX interpreter is started. To allow for this flexibility, many of the functions below take two arguments, `format` and `strategy`; `format` specifies what TEX format to use and `strategy` specifies the way in which TEX will be started. In many cases, these arguments are optional and appropriate values will be derived, either from the context of use or from the environment variable `TEXMFORMAT`. The environment variable has two different forms:

1. `TEXMFORMAT=`*format*
2. `TEXMFORMAT=@`*strategy*`:`*format*

The second form allows for specification of both the strategy and format. Currently `strategy` can be set to one of: `context`, `latex`,[17] or `plain`. The table below maps strategies to command lines:

strategy	command line
context	`texexec --format` *format* `--once %s`
latex	`latex %s`
plain	`tex &`*format* `%s`

A.2 Module-level Functions

`getNames(`*pathname*`)` → [*name*$_1$, *name*$_2$, ...]
Recursively scans the passed *pathname* and returns a list of merge variable names declared by instances of the `\texmergevar` macro.

`getExtendedNames(`*pathname*`)` → {*attrDict*$_1$, *attrDict*$_2$, ...}
Recursively scans the passed *pathname* and returns a dictionary of merge variable field attribute dictionaries. These dictionaries are created from instances of the `\texmergevardef` macro, which defines merge variables with extended attributes, like this:

`\texmergevardef[`*attrName=attrValue,...*`]`

Attributes of the merge variables that can be specified are:

- `name` = the name of the merge field,
- `type` = the type of merge field. The intended use of this attribute is to convey a recommended style of data entry element for graphical (GUI) applications. Valid types are:
  - `entry`: a simple text entry field,
  - `text`: a multi-line text entry field,
  - `toggle`: a toggle button field,
  - `optionmenu`: a drop-down option menu of choices,
  - `radiobutton`: a set of mutually-exclusive toggle buttons
- `values` = a list of valid values for the variable, separated by `|`'s.
- `labels` = a list of alternate labels that should be associated with the `values` attribute for display purposes. Used with the `toggle`, `optionmenu`, and `radiobutton` field types.
- `descr` = a description of the merge variable.

`hashNames(`*fieldAttributesDict*`)` → *StringObject*
Computes a 64-bit MD5 hash of the passed field attributes dictionary and returns it as a string object of hexadecimal characters.

`getInputFiles(`*pathname*`)` → [*pathname*$_1$, *pathname*$_2$, ...]
Recursively scans the passed *pathname* for occurrences of `\input` control sequences and returns a list of pathnames.

`openOutput(`*pathnameOrFileObject*`,` *preambleCode*`=None,` *formatIn*`=None,` *strategyIn*`=None)` → *FileObject*
Prepares a temporary work file for merge operations. The first argument can be either a string object or a file object. In the case of a string object, it is interpreted as the pathname to a file where the temporary merge file should be created. If it exists, it will be removed and re-created. In the case of a file object, the argument is assumed to be a previously opened file. Any write operations issued by TEXmerge will be executed against the passed file object.

`preambleCode`, if specified will be written at the beginning of the file in place of TEXmerge's normal preamble code. `formatIn` is currently unused. `strategyIn` determines the default form of preamble code to write. Valid values are `context`, `latex`, or `plain`.

`closeOutput(`*fileObject*`,` *postambleCode*`=None,` *formatIn*`=None,` *strategyIn*`=None,` *keepOpen*`=False)` → `None`
Completes preparation of a temporary work merge file for processing. *postambleCode* is written to the file if passed, otherwise an appropriate postamble will be supplied depending on the values of *formatIn* and *strategyIn*, if passed, or a default postamble will be written. The passed

[17] For the `latex` strategy, `format` = `latex` is always assumed.

fileObject will be closed unless *keepOpen* is passed as `True`.

`merge(`*targetPathname*`,` *mergeVariableDict*`,` *fileObject*`,` *options*`=0)` → `None`

Encapsulates the merge variables passed in *mergeVariableDict* for use in *targetPathname*. The merge variables are written to the merge work file as `\def` control sequences, and *targetPathname* is referenced via `\input` *targetPathname*.

Several merge options can be passed in the *options* argument:

1. `TXM_FRAMEVARS`: draw a box around every merged variable.
2. `TXM_DUPLEX`: assume the output will be printed on a duplexing device and insert `\eject` macros between merge invocations, when appropriate, to ensure that each merge invocation starts on the front side of the printed sheet.

`process(`*pathname*`,` *driverCommand*`,` *format*`,` *strategy*`)` → *IntegerObject*

Runs the TEX interpreter and a DVI backend against the merge work file *pathname*. The command used to run the TEX interpreter is derived from the *format* and *strategy* parameters. *Strategy* may be one of `context`, `latex`, or `plain`. If *strategy* is set to `context` then the environment variable `TEXENGINE` is used as the TEX processor, if set, or `texexec` otherwise. The DVI command string passed in *driverCommand* is used to run the DVI backend. It can contain a single `%s` which will be replaced with *pathname*. If no `%s` is present, *pathname* will be appended to *driverCommand*.

Returns the exit status of TEX interpreter or of the DVI backend command.

`processWithExtendedOutput(`*pathname*`,` *driverCommand*`,` *format*`,` *strategy*`)` → `(texstderr, texstdout, texlog, dvistderr, dvistdout)`

Works like the `process` function above, except for error handling. Failure of the TEX interpreter raises the exception `TeXException`. Failure of the DVI backend command raises the exception `DviException`. Successful completion of both the TEX interpreter and the DVI backend returns a tuple as above, providing complete diagnostics of the run.

`getNamedTextBlocks(`*pathname*`)` → {*block*$_1$: {*block*$_1$*AttrDict*}, ...}

Recursively scans *pathname* for occurrences of *named text blocks*, demarcated by the pair of macros `\StartNamedTextBlock[`*attrName* `=` *value*`, ...]` and `\EndNamedTextBlock`.

Text block attributes are as follows:

- `name` = Name of the text block,
- `seq` = *Integer*; several sections of text can be assigned the same name, but with unique sequence numbers. The extracted text will be a concatenation of like-named blocks, ordered by sequence number,
- *subkey*=*subvalue*; subkey name/value pairs provide a way to declare multiple blocks with the same name. Assigning differing name/value pairs makes each like-named block unique.

The class `TextBlockManager` can be used as an alternative to this function; it provides a simple frontend to this function's return value.

A.3 TEXmerge Class

The TEXmerge class provides an object-oriented interface to the module-level functions shown above.

Constructor

`TeXmerge(`*mergeTargetPathname*`=None,` *workPathname*`=None,` *mergeOptions*`=0,` *preambleCode*`=None,` *postambleCode*`=None,` *texmformat*`=None,` *strategy*`=None,` *keepIntermediateFiles*`=False)`

Methods

`setMergeTargetPathname(`*pathname*`)` → `None`

Sets the default merge target pathname for subsequent merge operations.

`setMergeOptions(`*self*`,` *mergeOptions*`)` → `None`

Sets the default merge options for future merge operations.

`setFormatAndStrategy(`*self*`,` *texmformat*`,` *strategy*`=None)` → `None`

Sets the default format and strategy to be used for future merge operations.

`probeMergeTargetAndSetFormat()` → `None`

Scans the current merge target pathname to determine the appropriate format and strategy that should be used during the `process()` method call.

`setFormatFromMergeTargetParentDirectory()` → `None`

Checks the merge target's parent directory for existence of the file `.texmformat`. If found, the contents of the file is assumed to be the format and strategy (specified similarly to the environment variable `TEXMFORMAT`) to be used when processing the merge file.

`getVariables()` → {*mergeVariableAttrDict*}

Calls the function `getExtendedNames` described above, passing the currently set merge target

pathname as an argument. Returns the result of the call.

`openOutput(`*workPathnameOrFileObject*`=None)` → *FileObject*
Prepares the work file for subsequent merge operations. If no argument is passed, a default filename will be constructed.

`closeOutput()` → `None`
As above.

`merge(`*mergeVars*`=None,` *altMergeOptions*`=None,` *altMergeTargetPathname*`=None)` → `None`
Performs a merge operation using *mergeVars*, if passed, and alternate merge options and merge target pathname, also if passed.

`process(`*driverCommand*`)` → `None`
Run TEX interpreter according to currently set strategy and format. See `process` description above for details on the *driverCommand* string.

A.4 TextBlock Manager Class

Constructor

`TextBlockManager(`*pathname*`)`

Methods

`setPathname(`*pathname*`)` → `None`
Requests the `TextBlockManager` instance to scan *pathname* for named text blocks. Any information about previously scanned blocks is lost.

`getBlockNames()` → `[`$block_1$`,` $block_2$`, ...]`
Returns a list of the names of all named text blocks in the pathname last scanned.

`getBlock(`*blockName*`)` → *TextBlock*
Returns a `TextBlock` instance representation of the text block named *blockName*. Returns `None` if no such named block exists.

This same operation can be performed by using array indexing notation against the instance, i.e., indexing as with a dictionary object.

A.5 TextBlock Class

Constructor

`TextBlock(`*text-block-descriptor-dictionary*`)`

Methods

`getName()` → *StringObject*
Returns the instance's block name.

`getSubKeys()` → `[`$blockName_1$`, ...] | None`
Returns a list of unique subkey names associated with the given text blocks, or `None` if there are no associated subkeys.

`getSubkeyValues(`*subkeyName*`)` → `[`$subkeyName_1$`, ...]`
Returns a list of all the subkey values corresponding to the passed subkey name.

`getTextSegments(`*subkeyName*`=None,` *subkeyValue*`=None)` → {1: $textSeg_1$, 2: $textSeg_2$, ...}
Returns a dictionary of text segments, keyed by segment sequence number. The *subkeyName* and *subkeyValue* are optional; if specified, they are used to select the specific text block to access.

`getText(`*subkeyname*`=None,` *subkeyValue*`=None)` → *StringObject*
Returns a concatenation of all text segments in order by sequence number. The *subkeyName* and *subkeyValue* are optional; if specified, they are used to select the specific text block to access.

A.6 Exceptions

Exceptions can be raised by some of the class methods above. The exception objects have attributes which provide diagnostics about the associated error condition.

A.6.1 TeXException

This exception is raised when TEX cannot successfully interpret a file. Attributes:

- `stdout`: *StringObject* containing the standard output stream from the interpreter invocation,
- `stderr`: *StringObject* containing the standard error stream from the interpreter invocation,
- `logText`: *StringObject* containing the log file written by TEX.

A.6.2 DviException

This exception is raised when a DVI backend driver fails. Attributes:

- `stdout`: *StringObject* containing the standard output stream from the backend invocation,
- `stderr`: *StringObject* containing the standard error stream from the backend invocation.

A Bibliographer's Toolbox

Nelson H. F. Beebe
University of Utah
Department of Mathematics, 110 LCB
155 S 1400 E RM 233
Salt Lake City, UT 84112-0090
USA
WWW URL: http://www.math.utah.edu/~beebe
Telephone: +1 801 581 5254
FAX: +1 801 581 4148
Internet: beebe@math.utah.edu, beebe@acm.org, beebe@computer.org

Abstract

This article surveys a portion of a set of software tools that I have developed over the last decade for the production, maintenance, testing, and validation of very large bibliographic archives.

It provides resource locations for all them, and shows how they can make bibliography preparation and maintenance more productive, and much more reliable.

Summary

1 Introduction

2 The problem(s)
2.1 Data errors
2.2 Markup features
2.3 Markup deficiencies

3 Creating bibliographic data
3.1 The emacs environment
3.2 Converting Web data to BibTeX
3.3 Other external tools
3.4 XML for bibliographic data

4 Checking bibliographic data
4.1 Spelling
4.2 Delimiter balance
4.3 Doubled words
4.4 File validation
4.5 Field-value validation
4.6 Typesetting

5 Other platforms

6 Conclusions

Editor's note: As an experiment, this paper is not typeset in Computer Modern. The main text is Bitstream Charter, with monospaced material in Bigelow & Holmes LuxiMono. We do not plan at this time to change from Computer Modern in general, but we still welcome your comments.

1 Introduction

LEXICOGRAPHER, N. A WRITER OF DICTIONARIES; A HARMLESS DRUDGE THAT BUSIES HIMSELF IN TRACING THE ORIGINAL, AND DETAILING THE SIGNIFICATION OF WORDS.
SAMUEL JOHNSON
Dictionary of the English Language (1755)

BIBLIOGRAPHER, N. SEE LEXICOGRAPHER.
ANONYMOUS

Large documents, especially in technical fields, often contain a list of other related documents, in the form of a bibliography or reference list. That list usually appears at the end of the document, but may instead be sprinkled through it in footnotes, or collected in endnotes, or be divided in multiple parts, with one part at the end of each major document division, such as a chapter.

Before computers came into wide use for document preparation, these reference lists were tedious to prepare, and were often sparse, with authors after the first reduced to the Latin catch-all *et al.*,[1] article titles, issue numbers, and months omitted, and page ranges reduced to the initial page.

This is a disservice to the reader, who has little idea what the referenced publication is about, and who then must work harder to find it. Much of the chemistry literature still follows this practice.

[1] A colleague once quipped: "When you see a paper cited as *Jones et al.*, it means that Jones got the credit, but Al did the work."

The tedious manual labor of preparing a reference list had to be repeated if the publication style changed, and error rates were high.

Bibliographic database systems, such as bib/refer, bibix, BibTeX, EndNote,[2] Papyrus,[3] and ProCite,[4] have made it possible to remedy this situation. Bibliographic data can now be carefully prepared once and for all, freely used by anyone, automatically reformatted into scores of styles, and converted between different database formats with reasonable ease.

As one example of a significant shared collection, the *Karlsruhe Bibliography Archive*[5] in mid-2004 contained about 1.4M references in BibTeX form, covering major areas of computer science, numerical analysis, electronic document production, fonts, and typography. A good portion of that archive is the result of my own work.[6]

A key feature of all of bibliographic database systems is the separation of *database markup* (data types `author`, `title`, `journal`, `year`, etc.) from *presentation markup*. Style files guide bibliographic software in the conversion from database form to presentation form. For example, a famous letter to the editor might be marked up like this in a BibTeX file:

```
@Article{Dijkstra:1968:GSC,
  author =       "Edsger Wybe Dijkstra",
  title =        "Go to statement considered
                 harmful",
  journal =      j-CACM,
  volume =       "11",
  number =       "3",
  pages =        "147--148",
  month =        mar,
  year =         "1968",
  CODEN =        "CACMA2",
  ISSN =         "0001-0782",
  note =         "This letter inspired scores of
                 others, published mainly
                 in SIGPLAN Notices up to
                 the mid-1980s. The best-known
                 is \cite{Knuth:1974:SPG}.",
}
```

For comparison, a subset of that data might be marked up in bib style like this:

```
%A Edsger Wybe Dijkstra
%T Go to statement considered harmful
%J Comm. ACM
%V 11
%N 3
%P 147-148
%D March 1968
```

In author-date reference style, inline citations to it might read *(Dijkstra, 1968)*, with the bibliography item in a `.bbl` file marked up like this for TeX or LaTeX:

```
\bibitem[\protect\citename{Dijkstra, }1968]
       {Dijkstra:1968:GSC}
Dijkstra, Edsger~Wybe. 1968.
\newblock Go to statement considered harmful.
\newblock \emph{Communications of the ACM},
\textbf{11}(3), 147--148.
\newblock This letter inspired scores of others,
published mainly in SIGPLAN Notices up to the
mid-1980s. The best-known is
\cite{Knuth:1974:SPG}.
```

It produces typeset output like this:

> Dijkstra, Edsger Wybe. 1968. Go to statement considered harmful. *Communications of the ACM*, **11**(3), 147–148. This letter inspired scores of others, published mainly in SIGPLAN Notices up to the mid-1980s. The best-known is (Knuth, 1974).

Notice in this example that it is possible for bibliographic entries to cite other entries; the cited entries are automatically retrieved and formatted by the software, without any extra effort by the document author.

A chemistry-literature citation of the same data might have a numeric superscript pointing to a footnote,[7] without any change to the citation in the document.

Bibliographic databases deserve to be widely used, freely shared, and contributed to by many. The time has come to abandon the cryptic reference-list practices of the past that were developed primarily as labor-saving devices, and replace them with accurate, and detailed, reference lists, as exemplified by the bibliography at the end of this article.

The databases that I have developed intentionally avoid unnecessary abbreviations: does *J. chem. phys.* mean *Journal of Chemical Physics*, or *Journal de chimie et physique*, or maybe even *Journal of Chemosurgery and Physiology*? They also supply details that historically were omitted, such as book and periodical numbers, library catalog numbers, and publisher states-or-provinces and countries. While there are many publishers in London, England, and in Paris, France, there are also some in

[2] http://endnote.com/

[3] http://www.researchsoftwaredesign.com/

[4] http://www.procite.com/

[5] http://liinwww.ira.uka.de/bibliography/

[6] http://www.math.utah.edu/~beebe/bibliographies.html

[7] E. Dijkstra, Comm. ACM **11** 147 (1968). [Notice here that the reader may have to search up to twelve monthly issues of this journal to find the article, because page numbers restart at one each issue, and the abbreviated reference style omits the issue and month information.]

London, Ontario, Canada, and maybe even a small one in Paris, Idaho, USA, not very far from my home in Utah.

The rapid spread of the Internet, the use of the World-Wide Web for sharing electronic documents, and free and independent search engines for finding them, requires a radical change to past practices. We need to get our old documents onto the Web, and we need to make our new documents include pointers to electronic versions of all documents that we reference.

If Brewster Kahle's magnificent vision [29, 16] of having all of the world's entire historical literary production online in the Internet Archive Wayback Machine[8] for free access by everyone comes to fruition, we will have an unprecedented resource of human knowledge. We will need to index it, search it, and be able to reference any part of it, so that others can find the same material. We also need to replicate it in many countries, and in many storage technologies, to avoid repeating the catastrophic destruction of the great Library of Alexandria, though just when, or how often, that happened is uncertain.[9]

2 The problem(s)

2.1 Data errors

The biggest problem with databases is *errors*: insurance-company studies have reported errors in as many as a third of all stored fields [31], and sometimes in up to 80% [26].

Bibliographic data is no exception; the quality of much of the Karlsruhe archive (apart from my own section) is poor, entirely due to careless preparation and checking on the part of human contributors.

The same criticism applies to many commercial and governmental bibliographic databases, such as *Compendex*, *Medline*, *Science Citation Index*, and the many *OCLC* databases. The *Uncover* database is so bad that virtually all of its data is suspect.

You may well be able to *find* a publication listed in these sources, but you cannot rely on their information to provide you with correct bibliographic data. Every field of data potentially contains errors introduced by careless, or low-paid, human typists. Mathematical markup is almost nonexistent, and when it is there, it is frequently unintelligible or completely wrong. Accents on letters are not recorded, which is particularly offensive when personal names are grossly misspelled.

Notable exceptions in database quality are the American Mathematical Society's *MathSciNet* facility,[10] the European Mathematical Society's *Zentralblatt MATH* archives,[11] and the SIAM journal publication lists.[12] These organizations deserve high praise for the care with which they have recorded bibliographic data.

Fortunately, major journal publishers now offer Web pages with issue tables of contents, and it is often possible to write software to automatically derive bibliographic entries directly from the HTML or XML markup. Since the publisher has the original data from which the bibliographic information is derived, and has a business interest in its quality, we can hope that fewer errors exist. I'll have more to say about this later.

Why does citation accuracy matter? Here are some reasons:

- Getting your references correct is not just a matter of ethical and professional responsibility: it shows respect for your reader.
- There are many more readers than writers, authors, and bibliographers. Your work will be seen by many, and for shared bibliographic data, also used and reused by many. Errors in such data will be amplified many times.
- When literature references are inaccurate, your competency is called into question. Erroneous bibliographic citations suggest carelessness in the rest of your work.
- An encouraging recent trend is for the full text of articles to be available electronically, with Web cross-links from the references to the publications that they cite. The ACM, AMS, EMS, and IEEE all have digital-library projects underway that provide such linking. Accurate bibliographic data is essential for correct links.

  The huge ArXiv e-print service[13] in physics, mathematics, non-linear science, computer science, and quantitative biology does not currently provide such links, possibly to avoid competition with professional-society and commercial collections, but live links between documents are too valuable to continue to ignore in the future.
- In business and law, contracts and legal decisions depend critically on document accuracy, including references to prior agreements and legal cases.

[8] http://www.archive.org/
[9] http://en.wikipedia.org/wiki/Library_of_Alexandria
[10] http://ams.org/mathscinet/
[11] http://www.emis.de/
[12] http://epubs.siam.org/
[13] http://www.arxiv.org/

2.2 Markup features

BIBTEX goes further than most other bibliographic systems in addressing the need for adequate markup. While its name implies a connection with TEX, its design was strongly influenced by the Scribe document-formatting system, and there was a period, now long past, where database interchangeability between the two systems was essential. There are BIBTEX styles available that format references for consumption by other typesetting systems, notably, nroff and troff, and some of the commercial bibliographic database systems can import and export BIBTEX data.

One of the most important design decisions in BIBTEX is that the publication types and the data field types are *not* hard-coded into the BIBTEX program. Instead, that program knows what their syntax is, but their names are defined only in style files. All such style files recognize a standard set of document types (`Article`, `Book`, `PhDThesis`, `TechReport`, `...`), and a standard set of field names (`author`, `title`, `year`, `...`). However, database entries can contain additional field names: unknown ones are silently ignored, even though they might simply be misspelled!

This flexibility has been enormously helpful in extending the markup to handle needs that were not foreseen when BIBTEX was designed. Perhaps the most notable of these is the World-Wide Web, with uniform resource locators (URLs) providing location information as a supplement, or alternative, to conventional data, like `journal`, `volume`, `pages`, `year`, and so on.

Other examples include the `DOI` (Digital Object Identifier),[14] `ISBN` (International Standard Book Number), `ISSN` (International Standard Serial Number), and `CODEN` (Chemical Abstracts serial number) field names, which provide handles that uniquely identify a document, a book or a periodical. The `is-*.bst` style files are extensions of the BIBTEX base styles that recognize these field names, and several others, and also understand an additional document type, `Periodical`.

Compared to other systems, BIBTEX markup is clear and simple. It requires only ordinary text files, and those files are readily understandable by anyone who can read English, even an elementary-school child. That is not the case with other systems; for example, in Unix bib, you might be able to guess that `%A` stands for *author* , but you probably have no idea what `%Q` means. [It is used for a corporate or 'foreign' author.]

[14] `http://www.doi.org/`

The clarity of BIBTEX markup has turned out to have an unexpected, but extremely valuable, benefit. Entries from BIBTEX files are readily found by Web search engines: try searching for *+@Book +Knuth* in your favorite search engine.

One important problem that BIBTEX has solved cleanly is the need for identification of proper nouns, so that those styles that downcase titles can be correctly supported. Thus, a field assignment like

```
title = "The Use of {Green} Functions for
         Modeling Growth of Green Algae",
```

lets BIBTEX preserve lettercase on the first instance of *Green*, and downcase the second, when the style calls for it. German capitalizes all nouns, so titles in that language need only an outer brace layer to eliminate disastrous downcasing:

```
title = "{Einschlie{\ss}en der L{\"o}sungen
         von Randwertaufgaben}. ({German})
         [{Bracketing} Solutions to Boundary
         Value Problems]",
```

The lack of such markup in other systems forces downcasing of the title data, thus losing possibly-important information about the lettercasing in the original publication.

It is certainly bad form for the database to lose information; any such data reduction should be entirely up to the style.

2.3 Markup deficiencies

While I am convinced that BIBTEX markup is currently the best choice for bibliographic databases, the experience of personally creating more than a third of a million entries, and using them daily for more than a decade, has turned up limitations that must be addressed in the final, and frozen, release of BIBTEX. Its author (Oren Patashnik) and I have had numerous electronic and face-to-face exchanges about these issues, and I'm confident that proper solutions will be found.

Here are just two examples of markup deficiencies:

- There is no author/editor value markup to distinguish between levels of authors. For example, my entry for the second edition of *The LATEX Companion* [30] includes this (unused) field:

  ```
  remark = "Authors listed as: Frank
            Mittelbach and Michel Goossens
            with Johannes Braams, David
            Carlisle, and Chris Rowley, and
            with contributions by Christine
            Detig and Joachim Schrod.",
  ```

  These wonderful people have done a truly outstanding job in this new edition, and it is unfair

to omit any of their names in the database. Consequently, they each appear in the author field separated by the BibTeX keyword and, so the book entry at the end of this article loses the level information.

- Parsing of personal names into *first*, *von*, and *last* parts is not general enough to handle different name order, such as Chinese, Hungarian, Japanese, and Vietnamese, where the family name comes first.

  Nor does it properly handle the case of South Indian authors with a single name: *Arvind* is a well-known example in computer science.

  It also fails for Spanish names with paternal and maternal contributions: *Juan García y Rodriguez* may be known as Juan García R., or just J. García, when the maternal part is abbreviated or dropped.

  As we learn more about the personal-name conventions of other parts of the world, more limitations of the current markup will certainly be found.

Much more could be said about markup deficiencies and challenges, but instead, I now want to turn to a review of tools that can be profitably used to ease the job of bibliography-entry preparation, checking, and typesetting.

3 Creating bibliographic data

A common way to create bibliographic entries is by manual data entry. For most systems, this is best done with the help of templates. I am not at all fond of the window-based clients, provided with most commercial systems, that present little boxes to be filled in. Their editing capabilities are badly crippled, and they provide no automated way to create parts of entries in advance, something that I routinely do in preparation of entries for journal-specific bibliographies. The little-boxes clients may be tolerable for creation of a dozen bibliographic entries, but they are completely hopeless for the creation of hundreds of thousands.

3.1 The emacs environment

The "Extensible, Customizable, Self-Documenting Display Editor", emacs, [3, 13, 14, 15, 17, 21, 28, 34, 35, 36, 37] provides the finest editing environment that I know of, far beyond that of any other editor ever developed and widely deployed. emacs celebrates its thirtieth birthday in 2006[15,16,17] and for most of its users, has changed their lives, making computers, document entry, and programming, more accessible. Its age is a badge of honor, one that it shares with TeX, two years its junior. My computer-input capability jumped tenfold when I adopted emacs more than twenty-five years ago.

What makes emacs different from many other editors arises from five fundamental design principles:

- Commonly-used editing commands are bound to easily-learned keys, but those bindings are always customizable. Most users retain the default bindings of the general commands for copying, deletion, insertion, and movement. However, key bindings of more specialized editing commands usually change with the type of data being edited.
- Time-critical editing tasks and display management are carried out by the emacs kernel (originally written in PDP-10 assembly code, now in C), but the bulk of editing functionality is handled by interpreted code (originally TECO, now Lisp). That code can be developed and tested in an interactive editing session, saved in a library file, optionally compiled for efficient reuse, and dynamically loaded into other editing sessions.
- While the emacs kernel remains under control of a handful of architects, led by Chief Architect and Head Gnu Richard Stallman, the emacs community is encouraged, and even expected, to develop specialized libraries that are freely shared with others.

  The assembly-code file, `teco.mid`, that forms the kernel of the original emacs ends with these comments:

```
;;; ITS TECO and EMACS should serve as a
;;; lesson to all of what can be achieved
;;; when programmers' creativity is not
;;; crushed by administrators whose main
;;; concern is stifling humor, stamping out
;;; all possibility of enthusiasm, and
;;; forbidding everything that isn't
;;; compulsory.
...
;;; You owe your improvements to us in
;;; return for what you see here.  If anyone
;;; asks you for a copy, make sure he gets
;;; in touch with the MIT AI Lab so he can
;;; get the latest stuff.
```

- All commands are accompanied by a short, but usually entirely sufficient, string of documentation that can be displayed with just a few keystrokes.

[15] http://www.gnu.org/software/emacs/emacs.html
[16] http://en.wikipedia.org/wiki/Emacs
[17] http://www.jwz.org/doc/emacs-timeline.html

- Full documentation is provided in the form of online text files, extensively cross-referenced, and linked into a tree or graph structure.

  Two keystrokes get you into the emacs info system for viewing the documentation, and two dozen single-character commands let you easily navigate through it. When you leave it to return to editing, and then subsequently reenter the info system, you are positioned exactly where you were when you left it, and the history of where you've been is intact.

  The info system contains highly-readable self-guided tutorials on info and emacs, making both systems rapidly accessible to new users. When emacs is built for a windowing system, it provides a conventional toolbar that makes it easy for novices to perform basic editing tasks without any knowledge of key bindings.

  The info system was hypertext [11] fifteen years before the idea was reinvented for the World-Wide Web, but the idea goes back to at least Vannevar Bush's far-sighted article in 1945 [12], and to work at Stanford University (the home of TEX, and much else) in 1962 by Douglas Engelbart and Ted Nelson. Engelbart is credited with the invention of the computer mouse and windows, and Nelson with coining the term *hypertext*.

When emacs was reimplemented in the mid-1980s for the GNU Project, it got an unlimited *undo* capability, which users find enormously liberating. If you make a mistake, you don't have to think about what editing commands to issue in order to recover: you just press the *undo* key until things are right. This facility is a sixth important design goal, even though it was not part of the original implementation.

TEX users will see a strong similarity between the developments of TEX and METAFONT, and the emacs editor, which is not surprising, since the Grand Wizard of TEX himself is a diehard emacs user. The emacs libraries have their analogues in the packages of the LATEX world.

emacs has superb support for the creation of BIBTEX data, and I've written more than a dozen related libraries[18] to further enhance the editing provided by the default BIBTEX library. These libraries provide more than 350 specialized functions for editing BIBTEX data.

For example, in an emacs session for a BIBTEX file, three quick keystrokes, or more awkwardly, selection of an item from a pull-down menu, generates a template like this, with the cursor positioned before the comma on the first line, ready for data entry:

```
@Article{,
  author =       "",
  title =        "",
  journal =      "",
  year =         "",
  OPTvolume =    "",
  OPTnumber =    "",
  OPTpages =     "",
  OPTmonth =     "",
  OPTnote =      "",
  acknowledgement = ack-nhfb,
  bibdate =      "Tue Jun 29 11:54:21 2004",
}
```

A tab character moves to the next value field, and two keystrokes remove the OPT prefix, which is there to remind the typist that BIBTEX considers that particular field optional.

The acknowledgement and bibdate fields are my own personal customizations. The latter holds a revision date that provides critical information for other software tools, such as bibextract.[19] They can extract entries matching specified patterns, for example, all those with 2003 and 2004 in the bibdate value, making it easy to find out what's new.

Once the entry is complete, two keystrokes generate a standard citation label, and when desired, two more keystrokes save the results in the filesystem.

If further processing of the new BIBTEX entry, or a block of entries, is required, it takes only two keystrokes to mark a region, and then two more to get a prompt for a shell command to run on that marked data, optionally replacing it with the command output. I use that capability constantly in my bibliographic work.

Part of the extended BIBTEX support in emacs provides commands for commonly-needed BIBTEX-specific editing activities, such as moving from field to field, justifying string values, bracing words before and after the cursor, and supplying TEX accents.

The accent support provided by the BTXACCNT and LTXACCNT libraries is particularly convenient and noteworthy. Rather than your having to laboriously enter a backslash and a (sometimes) look-alike punctuation character or mnemonic letter, and possibly also braces, a single function key pressed *after* a character supplies the next accent from a list known to be valid for that character. Repeated

[18] http://www.math.utah.edu/pub/emacs/

[19] http://www.math.utah.edu/pub/bibextract/

presses of the *same* function key cycle through the list until the desired accent is located.

For example, after the letter o, repeated accent-key presses replace it successively by

```
{\"o}   {\'o}   {\.o}   {\=o}   {\H{o}}
{\^o}   {\`o}   {\b{o}} {\c{o}} {\d{o}}
{\r{o}} {\t{o}} {\u{o}} {\v{o}} {\~o}
```

The list is cyclic, so you never bump into its end, and if you go too far, the emacs *undo* key backtracks as far as you need to go.

Once you have found the needed accent, which you indicate just by pressing any key other than the accent key, the accent list for that character is then rotated to place the just-selected accent at the head of the list. It is likely that the same accent will be needed again, and a single accent keystroke will then retrieve it.

If you press the accent key after a letter with no known accents, emacs beeps and warns *No accented letter match* in the message area at the bottom of the screen.

Because TEX provides many more accents than any particular language needs, the accent lists, and thus, the accent-key presses, can be substantially shortened by selecting a language from a menu of a score of languages, including Faroese, Gaelic, Latin, Romaji, and Turkish. Of course, those lists are all customizable, so support for new ones can be added in just a few minutes.

For convenience, commonly-required external tools can be invoked from menus provided by the BIBTEX-TOOLS library, making it easy to find and run them without having to remember their sometimes long descriptive names. The menu with the functions and tools that I use often is shown in Figure 1.

Emacs is available on all popular desktop systems, although on the commercial ones, you may have to install it yourself. There is then no need to learn a new editor each time that you change platforms.

Figure 1: Partial toolbar menu for BIBTEX-TOOLS.

```
update citation label table
print citation label table
bibcheck
bibparse
check-bbl
check-page-gaps
check-page-range
chkdelim
find-author-page-matches
find-braceable-initial-title-words
find-crossref-year-mismatches
find-duplicate-author-editor
find-duplicate-pages
find-german-titles
find-hyphenated-title-words
find-math-prefixes
find-missing-parbreaks
find-page-matches
find-possessive-title-words
find-superfluous-label-suffixes
...
```

3.2 Converting Web data to BIBTEX

About the mid-1990s, another source of bibliographic data began to be available: the World-Wide Web. This takes primarily two forms: document texts that happen to contain references to other documents, and Web pages listing the contents of journal issues and conference proceedings.

The first is completely devoid of markup, and thus, requires considerable manual work to reconstruct bibliographic-database entries. All of the caveats that I raised earlier about errors apply!

The second, while varying from site to site, and even from month to month at the same site, often has enough HTML or XML markup that software can be written to construct rough BIBTEX entries that can be further cleaned up with a combination of manual editing, and many of the tools that are described elsewhere in this article.

In the best case, this conversion is nearly perfect, and fast: shortly after receiving a publication announcement in e-mail from the publisher or editor, I can sometimes create, validate, and install in the archive BIBTEX entries for a new journal issue in under five minutes. By contrast, manual creation of entries averages about that amount of time for just one.

Of course, software for the conversion of Web data to BIBTEX form is not simple to write, and could never be justified unless there is an expectation that multiple Web pages in the same format will be available for other journals of interest, and into the future. Fortunately, this has proved to be the case for some important publishers and databases, so I have been able to maintain coverage of about 300 journals. Some of them are only updated at long intervals, such as yearly, but others are updated as each new issue appears. Coverage is complete for 115 of these journals; the oldest of them is the *American Mathematical Monthly*, which goes back to 1894, and contains over 45,000 extensively-cross-referenced articles. [The *Monthly*'s Editor-in-Chief for many years is a member of my Department.]

The programming language that I have found

most suitable for the conversion task is awk [2, 32, 33]. The awk language has a clean and simple syntax that borrows heavily from a small subset of the C language, and avoids the kitchen-sink syntax of certain scripting languages currently in vogue. Programs in awk are quite often right the first time, and very clear. Several of the bibliography tools described in this article are written in awk, and the books cited above contain numerous examples of awk programs.

Importantly, and unlike almost all other current scripting languages, awk has multiple implementations, three freely-distributable, and two commercial. Commercial Unix vendors supply snapshots of up to three of the free versions. This means that when a bug surfaces, it is a trivial matter to run your program with a different implementation: if the bug persists, it's yours; otherwise, it may be in the implementation.

awk's roots go back to 1978 [1], and it was designed by leading researchers in the field of parsing and compilation of programming languages at the Unix research group in *AT&T Bell Laboratories*. The language received a major overhaul in 1987, and is well described in a classic small book that appeared the following year [2].

A file count in my bibliography archive directories found 283 distinct awk programs ranging in length from about 10 lines to 12,900 lines. The arithmetic mean is 480 lines, but the geometric mean provides a more typical value: 180 lines. The total collection has nearly 122,000 lines of awk code. For comparison, TEX and METAFONT are each about 20,000 lines of prettyprinted Pascal code.

The Web conversion task does not, however, rely entirely on awk programs. Without exception, all Web pages are first processed by html-pretty,[20] my prettyprinter for HTML that standardizes markup and layout. This makes it *much* easier to write the journal- and publisher-specific awk programs.

Because these programs need occasional revision to adapt to the often-whimsical changes in Web-page format at publisher sites, I have not released them for general use, but there is little need to, at least before my demise, since they are only needed at the one site where the conversion to BIBTEX takes place. However, they can certainly be made available to people with reasonable requests and expectations, perhaps for use in converting Web data for other journals.

There is a great deal of commonality in the processing of Web pages to create BIBTEX output, so for most journals, a single master shell script of about 500 lines manages the conversion. It contains a giant case statement with customizations for each of about 150 journals and journal families, and ends with a loop over command-line arguments and a body with a Unix pipeline that looks roughly like this:

```
eval $PREHTMLFILTER |
  html-pretty |
    eval $POSTHTMLPRETTYFILTER |
      eval $PREAWKFILTER |
        gawk -f $BASENAME.awk \
             -v Filename=$f \
             -v JOURNAL=$JOURNAL \
             -v Journal=$JOURNAL |
          gawk -f HTML-entity-to-TeX.awk |
            gawk -f iso8859-1-to-TeX.awk |
              $POSTAWKFILTER
```

For readers unfamiliar with Unix pipelines, the vertical bar is called the *pipe operator*: it means that the output from the program on its left is the input to the one on its right. Pipeline data flows through memory buffers, rather than through files in permanent storage media, and all programs in the pipeline run simultaneously.

The output of this pipeline is trapped in a temporary file, and then further cleaned up like this:

```
biblabel $TMPFILE |
  citesub -f - $TMPFILE |
    bibsort |
      biborder |
        bibclean $BIBCLEANFLAGS |
          $POSTPOSTFILTER |
            $COMMENTFILTER
```

Most of the tools in these pipelines are described elsewhere in this article. However, these two code fragments make one thing clear: the Unix *small-is-beautiful* philosophy of software design [7, 8, 9, 19, 10, 20] is extremely powerful. Unlike the commercial offerings for bibliographic databases, there is no megalithic monstrosity that does 'everything'. Instead, fifteen separate programs each handle part of the task. Each does its job well, and each remains ignorant of what the others do.

3.3 Other external tools

There are several other tools that are worth noting briefly.

3.3.1 String abbreviations

One of the wise decisions that I made early on was to use standard abbreviations for journals, publishers, and publisher addresses, like this:

```
@String{j-QUEUE = "ACM Queue: Tomorrow's
                   Computing Today"}
```

[20] http://www.math.utah.edu/pub/sgml/

```
@String{pub-GNU-PRESS     = "GNU Press"}
@String{pub-GNU-PRESS:adr = "Boston, MA, USA"}
```

The strings `j-` and `pub-` are two of a small set of prefixes that divide BibTeX abbreviations into namespaces, reducing collisions. Others include `inst-` for institutions, and `org-` for organizations.

The string abbreviations serve three important purposes:

- They supply a unique handle for the resource, for example, making it easy to find all of the articles in the database that are published in *ACM Queue*.
- They eliminate inconsistencies from use of differing abbreviations for the same resource.
- They provide a single point of redefinition of the resource name.

People are often surprised to learn that there are no 'official' abbreviations for journal names: publishers often disagree. All of my bibliographies therefore use full journal names in the string definitions, but if a user requires abbreviated names, it is a trivial matter to insert a following alternate definition.

Two programs, journal.awk and publisher.awk, filter an input BibTeX stream, and output a new stream in which `journal`, `publisher`, and `address` values that have been found to match any of several common variations are replaced by the standardized abbreviations, and the data stream is prefixed by corresponding `@String{...}` definitions.

The preferred name for the string abbreviation is constructed by uppercasing an abbreviation from a large and reputable source and replacing runs of nonalphanumerics with hyphens. I prefer the catalogs of the *U.S. Library of Congress* and the *University of California Melvyl* for these sources. Thus, the journal *Biochemistry and Molecular Biology International* is found to have an abbreviation *Biochem. mol. biol. int.*, and is therefore given the handle `j-BIOCHEM-MOL-BIOL-INT`.

3.3.2 CODEN, DOI, and ISSN data

Once a handle for a journal name in a BibTeX entry has been identified by journal.awk, it automatically supplies values for CODEN and ISSN fields, when it knows them: it does, for about 2500 journals. This makes it simple to retrofit those values into all BibTeX entries for a particular journal.

Each BibTeX entry, together with the string abbreviations, should be a complete and independent record of the document citation. It is definitely *not* sufficient to record those values just once, in the comment header of a BibTeX file, because those values are lost when people copy entries from one BibTeX file into another.

If a publisher chooses a predictable DOI or URL for a journal article, such as one based on the ISSN, volume, number, and initial page, it is then possible to automatically retrofit values for those fields into each BibTeX entry for that journal. Sadly, the practice so far on the part of most publishers has been to use apparently-random numbers, or otherwise-unpredictable strings, in forming DOIs. This is a great shame, and a terrible loss of a great opportunity to make DOI assignment trivial for much of the world's existing and future periodical literature.

3.3.3 Missing brace protection

One of the more tedious tasks that a bibliographer must deal with properly is identification and bracing of proper nouns in titles.

In the Web pages of some publishers, there is sufficient markup and consistency that the conversion software can automatically supply those braces.

For others, the software has internal lists of names, like *Einstein*, *Navier-Stokes*, and *Schrödinger*, that frequently occur in titles, and are known to always be proper names. The *Green* example given earlier is one of the difficult cases where human understanding of the title is needed before protecting braces can be properly supplied. The easy cases are those words with mixed case, such as *BiCGS* and *McLeod*, which are always known to be proper nouns in need of protection; the conversion software braces them automatically. Another clue to a proper noun is its appearance as a possessive, as in *Another View of Einstein's Theory*.

These steps help to identify most of the proper nouns in titles, which is a particularly common practice in some areas of science, but there are always new names that turn up. For the volumes of bibliographic data that I deal with, visual examination of entries to find improperly-downcased title words is *not* practical.

To help solve that problem, and sharply reduce the number of instances of missing protecting braces, I wrote the check-bbl.awk program, with a companion shell-script wrapper, check-bbl.sh. The program searches the formatted bibliography file, usually BibTeX's `.bbl` output file, but a thoughtless user might have created such a file by hand as well, looking for words that occur both in protecting braces, and entirely in lowercase. That way, it will likely spot an instance of *einstein* in a physics bibliography.

To make it much more likely that such errors

will be detected, check-bbl.awk keeps an exception list that it updates on each run, recording protected names, and in what BibTeX entry and file they were found. A typical entry in that list is a line like this:

```
{Wolfram}    Varney:1991:WBM    mathematica.bbl
```

At the time of writing, the exception list for my archives contains nearly 18,000 entries: that provides a *huge* reduction in the number of missed protecting braces.

3.3.4 Breaking up large BibTeX files

When BibTeX files get too big to manage comfortably, or worse, overflow internal tables in TeX or BibTeX, they need to be subdivided.

The bibsplit[21] tool splits them into parts, separating entries into different output files according to one of several criteria: author, citation label, citation count, or year range.

I've had to do this numerous times with journal-specific bibliographies as their entry count grows. Usually, subdivision into decade-specific bibliographies suffices.

3.3.5 Database searching

For small databases, polished brute-force direct-search utilities like Unix grep and agrep[22,23] are reasonable solutions to the problem of finding an entry that you can only remember parts of.

agrep (approximate grep) deserves to be better known, and more widely used: it is capable of finding matches with data containing errors, such as transposition, truncation, or insertion. Here are some examples:

```
% echo Knuht | grep Knuth
# No output reported: there's an input typo

% echo Knuht | agrep Knuth
# No output: agrep normally works like grep

% echo Knuht | agrep -1 Knuth
Knuht

% echo Knuh | agrep -1 Knuth
Knuh

% echo Knooth | agrep -2 Knuth
Knooth
```

The numeric options allow matching of strings with, here one or two, errors. This sort of capability is important, because of the database-error problem discussed earlier.

Another valuable feature of agrep is its ability to return the complete paragraph where the match was found. Since I conventionally separate BibTeX entries by a blank line, an entry *is* a paragraph, and agrep can thus report complete entries.

For large collections, however, direct search is far too slow, since the entire database corpus must be read.

One possible solution is provided by glimpse,[24] which is free to academic institutions, but requires a license fee for others. It functions with the help of a list of all unique words to each of which is attached a list of 256 buckets, each containing the names of 1/256 of the files being indexed. That list is created by glimpseindex, which is run at suitable intervals, such as nightly. The list is collapsed to 256 bits (32 bytes), and a nonzero bit in the list means that the word is found in at least one file in the corresponding bucket. glimpse then uses agrep to do its searching, but only for the subset of files known to contain the sought-for word or phrase.

For file collections running into the hundreds of megabytes, as mine do, even glimpse is not fast enough.

The solution that I happily use is bibsearch,[25] which is a frontend for the mg database [38]. Once the database index is loaded into memory, lookups are extremely fast: on our newest servers, bibsearch lookups take under one millisecond.

Unfortunately, mg has three deficiencies that prevent its use as a general Web search engine, something that I'd very much like to offer for my archives:

- mg lacks subfield searching, so it is impossible to restrict a search to find entries with Knuth in the title, but excluding entries where that name occurs elsewhere. Consequently, bibsearch often returns more than you really wanted.
- mg always strips suffixes, so a search for *hyperbola* also reports entries containing *hyperbolic*, once again returning more results than expected.
- mg supports shell escapes, making it quite difficult to provide search access to your system without giving away login access.

These are all soluble problems, but I lack the time to tackle them. Volunteers, anyone?

It may be that a new generation of Web search engines will provide a solution. While this article

[21] http://www.math.utah.edu/pub/bibsplit/

[22] ftp://ftp.cs.arizona.edu/agrep/agrep-2.04.tar.Z (Unix)

[23] http://www.tgries.de/agrep/337/agrep337.zip (Windows)

[24] http://www.webglimpse.org/

[25] http://www.math.utah.edu/pub/bibsearch/

was being written, I learned of the new estraier[26] engine that looks promising. It is also possible that conventional relational databases, such as the commercial DB2, Oracle, or Sybase systems, or the free postgresql or mysql programs, will have acceptable performance, but I have not yet found time to experiment with them for BibTeX data retrieval.

3.4 XML for bibliographic data

In my keynote address at TUG 2003 [6], I discussed the BibTeXML project at the Swiss Federal Institute of Technology (ETH) in Zürich, Switzerland, which has developed software for conversion between XML and BibTeX markup of bibliographic data. At the time, their Web site was inaccessible. I'm pleased to report that the project is now back online, but at a new location.[27]

Although the syntax of XML is quite different from that of BibTeX, both supply a lot of useful markup. If adequate support tools, analogous to BibTeX and the many others described in this article, can be developed, then XML will be an important alternative for the representation of bibliographic data.

One important bibliography project that uses XML is the *Digital Bibliography and Library Project* (DBLP)[28] maintained by Michael Ley at Universität Trier, Germany, with about 500,000 entries in the field of computer science.

4 Checking bibliographic data

4.1 Spelling

A typical BibTeX entry in my bibliography archives has about 750 characters and 20 lines. That is a lot of opportunity for fumble fingers to introduce typos, even if the typist is a near-perfect speller.

It is therefore imperative that bibliographic data be spell checked, preferably by more than one spelling checker. In my bibliographic work, I now use three such programs: traditional Unix spell, GNU ispell, and a new one that I developed for a book [33] that is in press at the time of writing this article. Its code is at the book's Web site.

The spelling checks are applied to the entire bibliographic file, not just to individual fields. That way, the extensive comment headers present in each file are also checked. Each file has its own exception dictionary to augment the spell checkers' own dictionaries, because the large numbers of proper names and technical words include many words that are absent from standard lists.

4.2 Delimiter balance

It takes an extremely careful proofreader to catch delimiter-balance errors, such as an open parenthesis followed by a long block of text without a matching close parenthesis. Humans can never do this job reliably, but a computer can. More than a decade ago, I wrote chkdelim,[29] and I apply it routinely to all updates of the bibliographic data in my archives. chkdelim has special knowledge of BibTeX, Lisp, Scribe, TeX, and Texinfo. It also allows the user to request that checks for certain delimiters be suppressed: for example, angle brackets come in matching pairs in SGML, HTML, and XML, but almost never in mathematical documents.

4.3 Doubled words

Doubled-word errors (as occurs here: *in the **the** book*) are difficult for humans to spot. My dw program[30] finds them easily. It caught previously-unreported errors in both the *TeXbook* and the first edition of the *LaTeX User Guide*, even though over 100,000 copies of the former had been sold, and its author offered monetary rewards for the first reports of errors in the book and its software.

4.4 File validation

While the ability of a bibliographic system to process the database is an essential check, it may not be sufficient, because uncited entries might receive only rudimentary parsing, or be ignored entirely. Worse, the exact syntax of bibliographic data may be uncertain, for lack of a formal description.

At TUG 1993, I presented a rigorous grammar for BibTeX, and four tools based on it [4, 5]:

- bibparse[31]
- biblex (included with bibparse)
- bibunlex (included with bibparse)
- bibclean[32]

The first of these, bibparse, merely confirms that its input conforms to the grammar: a successful validation produces no output. I use this as an initial check of updates to the bibliographic archives before even attempting to run LaTeX and BibTeX: any error from bibparse immediately aborts the entire automated installation process.

[26] http://estraier.sourceforge.net/
[27] http://bibtexml.sourceforge.net/
[28] http://dblp.uni-trier.de/
[29] http://www.math.utah.edu/pub/chkdelim/
[30] http://www.math.utah.edu/pub/dw/
[31] http://www.math.utah.edu/pub/bibparse/
[32] http://www.math.utah.edu/pub/bibclean/

The second, biblex, parses the input and produces a token stream that is much easier to handle by other tools.

The third, bibunlex, reassembles a biblex token stream, possibly after filtering, reconstructing a valid BIBTEX file.

The fourth, and most powerful, bibclean, has been of enormous importance in my bibliographic work. It is based on the same rigorous grammar as bibparse, but is implemented completely independently with a carefully-hand-coded parser, instead of the machine-generated parser in bibparse. That way, there are two independent checks on the validity of the syntax of BIBTEX data.

bibclean normally produces a prettyprinted bibliography file in which numerous repairs and checks have been made on the data. For hundreds of thousands of examples, see the *TEX Users Group Bibliography Archive*[33] and the *BibNet Project* archive.[34] On request, however, bibclean produces the same token streams as biblex.

4.5 Field-value validation

One of the advantages of data markup is that it restricts the possible data values. For example, spreadsheets permit the user to assign a data type, such as currency, to a column, so that an attempt to store an alphabetic string there will be rejected. It may also be possible to specify a range of acceptable values, such as $[\$0, \$250]$ for a college student's weekly expenses.

bibclean can validate the data for many field names, and it does so primarily with the help of user-supplied patterns. The patterns are specially-designed for use with bibliographic data; they can, for example, check that a page number is a roman numeral, an arabic number, or an uppercase letter followed by a number. bibclean has special internal support for verifying the check digits in CODEN, ISBN, and ISSN values, and it also gets startup initializations that identify the valid ranges of ISBN values, which expand from year to year.

bibcheck[35] is another useful tool. Like lacheck and the more recent chktex, both of which look for common typographic-markup errors in LATEX files, bibcheck applies a number of heuristic checks to BIBTEX files.

These automated tools help, but they cannot tell whether a year value of 2003 should really be 2004. To solve problems like that, it is necessary to collect bibliographic data from multiple independent sources, and then merge that data, looking for discrepancies. Except for tiny bibliographies, this is far too tedious, and much too unreliable, to do by hand. Instead, a combination of tools provides a solution:

- bibclean first standardizes the format of the BIBTEX data, greatly simplifying many other tools.
- biblabel[36] and its companion tool citesub, and an independent implementation in the emacs BIBTEX-LABELS[37] library, generate standardized citation labels that are unlikely to conflict with those of other entries, and importantly, that are easy for humans to predict as well.
- bibsort[38] sorts entries in a bibliography by any of a half-dozen different criteria.
- biborder[39] reorders fields within a BIBTEX entry into a standard order, making the entries much easier to read.
- bibjoin[40] merges adjacent BIBTEX entries that appear to describe the same publication, discarding duplicate data, choosing more detailed values over less detailed ones (e.g., in the author field, the longer of Donald E. Knuth and D. E. Knuth), and otherwise leaving the duplicate fields adjacent for manual correction.
- bibdup[41] checks for duplicate abbreviations and entries.
- For interactive location and repair of problems, the emacs function find-duplicate-label from the BIBTOOLS library[42] finds the next occurrence of consecutive entries with the same citation label. The function find-duplicate-key from the same library finds the next instance of duplicate adjacent field names in a single BIBTEX entry.

4.6 Typesetting

Because bibliographic database software may do only cursory interpretation of field values, it is important to verify that the data can be processed by the document-formatting system.

This is particularly easy to do with BIBTEX databases. Each of the BIBTEX files in my archives is accompanied by a small LATEX wrapper file that

[33] http://www.math.utah.edu/pub/tex/bib/index-table.html
[34] http://www.math.utah.edu/pub/bibnet/
[35] http://www.math.utah.edu/pub/bibcheck/
[36] http://www.math.utah.edu/pub/biblabel/
[37] http://www.math.utah.edu/pub/emacs/
[38] http://www.math.utah.edu/pub/bibsort/
[39] http://www.math.utah.edu/pub/biborder/
[40] http://www.math.utah.edu/pub/bibjoin/
[41] http://www.math.utah.edu/pub/bibdup/
[42] http://www.math.utah.edu/pub/emacs/

in minimal file-independent generic form looks like this:

```
\documentclass{article}
\begin{document}
    \nocite{*}
    \bibliographystyle{unsrt}
    \bibliography{\jobname}
\end{document}
```

The archives include these wrappers, and their output DVI, PostScript, and PDF files, to demonstrate that there are no show-stopping TEX syntax errors in the data, at least that which makes it into BIBTEX output. Errors in ignored fields will not be caught, but then, those fields are also unlikely to appear in the published bibliographies, precisely because they are ignored.

The wrappers that are actually used are somewhat more complex, because they also include a comprehensive title-word index, allowing readers who have retrieved only a typeset bibliography to quickly locate entries of interest. The index is important, because the average size of the typeset bibliographies is 125 pages, and they range from 2 to 939 pages.

5 Other platforms

All of the tools described here, and the more than 500 BIBTEX files, have been developed in the fantastic Unix programming environment. A project of this magnitude would have been infeasible on other operating systems.

The tools, and their daily invocations, make extensive use of pipelines, shell scripts, awk programs, filename pattern matching, and the ability to run multiple simultaneous processes. Repetitive tasks are managed by the make utility, the greatest software tool ever written (emacs comes second in that list).

Where does this leave potential users who are stuck with a different operating system? The answer is POSIX (pronounced *pahz-icks*, as in *positive*), a term coined by Richard Stallman for the *IEEE Standard Portable Operating System Interface* [22, 25, 24, 23]. Besides the formal Standard, POSIX is described in a few books [18, 27, 39]. Because POSIX is a significant standard, it has been implemented on scores of operating systems,[43] including all of those that you are likely to use. It just may not come by default with your vendor's operating-system distribution.

Apple Macintosh users need only upgrade to Mac OS X, which is a Unix system derived from FreeBSD and others, and then drag the /Applications/Utilities/Terminal icon onto the toolbar to make a Unix shell readily accessible.

Apple includes a recent version of emacs in /usr/bin, but it was unfortunately not built with X Window System support. However, Apple now makes available an X Window System package. Once it is installed, you can drag the /Applications/XDarwin icon onto the toolbar for ready use.

Work is underway to support both X and the native window system for emacs on Mac OS X, and there is a Web page describing the status of that work.[44] Should it become inaccessible, try a Web search for *Emacs 21 for Mac OS X*.

Following the instructions on that Web page, while writing this article, I successfully built and installed on a recent MacOS 10.3.2 system a working development-release of emacs for the native window system using the Apple-supplied /usr/bin/gcc compiler, which has extensions needed to build software that communicates with the operating system. Those extensions are not in standard gcc distributions.

For Microsoft Windows users, there are at least four packages that layer a POSIX or Unix-like environment on top of Windows, giving access to all of the tools described here. Consult the Web pages that I maintain on these topics for full details.[45,46,47] For an emulation-free native Unix environment, I highly recommend the outstanding commercial VMware system described in those Web documents. VMware makes it possible to run multiple native operating systems on Intel IA-32 hardware, with either a Windows or a GNU/Linux base operating system. At my Department, we use VMware to run FreeBSD, NetBSD, OpenBSD, Plan 9, Solaris x86, and Microsoft Windows on top of GNU/Linux.

6 Conclusions

This article has surveyed about two dozen software tools that support the creation and maintenance of large collections of bibliographic data in BIBTEX markup. Programs like emacs, bibclean, html-pretty, and others can be used productively in a Unix or POSIX environment to tackle projects whose size would be unthinkable for a single person on other

[43] http://standards.ieee.org/regauth/posix/

[44] http://members.shaw.ca/akochoi-emacs/stories/obtaining-and-building.html

[45] http://www.math.utah.edu/~beebe/gnu-on-windows.html

[46] http://www.math.utah.edu/~beebe/windows-on-gnu.html

[47] http://www.math.utah.edu/~beebe/unix.html

operating systems, or lacking most of these tools.

Space has not permitted descriptions of more than a hundred other specialized tools, but if you would find them useful, I could probably be encouraged to package up more of them on a Web site for free access and distribution. The many URLs for bibliographic resources at Utah cited in footnotes in this article are already conveniently accessible via links from a master file.[48]

A carpenter's toolbox contains large things, like drills, hammers, and saws, but it also has lots of small specialized items, such as a 3 mm nail countersink, that are used only occasionally, but do a single job well that few other tools can manage. I have no hesitation in drawing an analogy between that hardware toolbox, and my software *Bibliographer's Toolbox*.

References

[1] Alfred V. Aho, Brian W. Kernighan, and Peter J. Weinberger. Awk — A pattern scanning and processing language. *Software—Practice and Experience*, 9(4):267–279, April 1979.

[2] Alfred V. Aho, Brian W. Kernighan, and Peter J. Weinberger. *The AWK Programming Language*. Addison-Wesley, Reading, MA, USA, 1988. ISBN 0-201-07981-X. x + 210 pp. LCCN QA76.73.A95 A35 1988.

[3] Larry Ayers. *GNU Emacs and XEmacs*. Prima Publishing, Roseville, CA, USA, 2001. ISBN 0-7615-2446-0. xxxv + 508 pp. Includes CD-ROM.

[4] Nelson Beebe. Bibliography prettyprinting and syntax checking. *TUGboat*, 14(3):222, October 1993.

[5] Nelson Beebe. Bibliography prettyprinting and syntax checking. *TUGboat*, 14(4):395–419, December 1993.

[6] Nelson H. F. Beebe. 25 years of TEX and METAFONT: Looking back and looking forward: TUG 2003 keynote address. *TUGboat*, 25(1), 2004. In this volume.

[7] Jon Louis Bentley. *Programming Pearls*. Addison-Wesley, Reading, MA, USA, 1986. ISBN 0-201-10331-1. viii + 195 pp. LCCN QA76.6.B453 1986.

[8] Jon Louis Bentley. *More Programming Pearls: Confessions of a Coder*. Addison-Wesley, Reading, MA, USA, 1988. ISBN 0-201-11889-0. viii + 207 pp. LCCN QA76.6.B452 1988. US$18.75.

[9] Jon Louis Bentley. *Programming Pearls (reprinted with corrections)*. Addison-Wesley, Reading, MA, USA, 1989. ISBN 0-201-10331-1. viii + 195 pp. LCCN QA76.6.B453 1989.

[10] Jon Louis Bentley. *Programming Pearls*. Addison-Wesley, Reading, MA, USA, second edition, 2000. ISBN 0-201-65788-0. xi + 239 pp. LCCN QA76.6.B454 2000. US$24.95. This differs greatly from the first edition: both are well worth reading.

[11] B. Brown. The theory of HyperText. *WebNet Journal: Internet Technologies, Applications & Issues*, 2(1):46–51, 1999.

[12] Vannevar Bush. As we may think. *The Atlantic Monthly*, 176(1):101–108, July 1945.

[13] Debra Cameron. *GNU Emacs Pocket Reference*. O'Reilly & Associates, Inc., 981 Chestnut Street, Newton, MA 02164, USA, 1999. ISBN 1-56592-496-7. iii + 58 pp. LCCN QA76.76.T49 C348 1998. US$6.95.

[14] Debra Cameron. *GNU Emacs — kurz & gut*. O'Reilly & Associates, Inc., 981 Chestnut Street, Newton, MA 02164, USA, 2000. ISBN 3-89721-211-0. 60 pp. German translation of [13].

[15] Robert J. Chassell. *An Introduction to Programming in Emacs Lisp*. GNU Press, Boston, MA, USA, 2001. ISBN 1-882114-43-4. 320 (est.) pp. US$30.

[16] Stuart Feldman. A conversation with Brewster Kahle. *ACM Queue: Tomorrow's Computing Today*, 2(4):24, 26–30, 32–33, June 2004.

[17] Craig A. Finseth. *The Craft of Text Editing—Emacs for the Modern World*. Springer Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 1991. ISBN 0-387-97616-7 (New York), 3-540-97616-7 (Berlin). xii + 220 pp. LCCN QA76.76.T49 F56 1991. Contains extensive discussion of design issues for text editors, with examples from Emacs. Appendix B gives sources of numerous Emacs implementations. Appendix D summarizes the TECO command set.

[18] Bill Gallmeister. *POSIX.4: Programming for the Real World*. O'Reilly & Associates, Inc., 981 Chestnut Street, Newton, MA 02164, USA, January 1995. ISBN 1-56592-074-0. xviii + 548 pp. LCCN QA76.76.O63 G34 1995. US$29.95.

[19] Mike Gancarz. *The UNIX philosophy*. Digital Press, 12 Crosby Drive, Bedford, MA 01730,

[48] http://www.math.utah.edu/pub/bibtools.html

USA, 1995. ISBN 1-55558-123-4. xix + 151 pp. LCCN QA76.76.O63 G365 1995.

[20] Mike Gancarz. *Linux and the Unix Philosophy*. Digital Press, 12 Crosby Drive, Bedford, MA 01730, USA, 2003. ISBN 1-55558-273-7. xxvii + 220 pp. LCCN QA76.76.O63 G364 2003. US$34.99.

[21] Bob Glickstein. *Writing GNU Emacs Extensions*. O'Reilly & Associates, Inc., 981 Chestnut Street, Newton, MA 02164, USA, 1997. ISBN 1-56592-261-1. xviii + 215 pp. LCCN QA76.76.T49 G56 1997. US$29.95.

[22] IEEE. *IEEE Std 1003.1-2001 Standard for Information Technology — Portable Operating System Interface (POSIX) Base Definitions, Issue 6*. IEEE, New York, NY, USA, 2001. ISBN 1-85912-247-7 (UK), 1-931624-07-0 (US), 0-7381-3047-8 (print), 0-7381-3010-9 (PDF), 0-7381-3129-6 (CD-ROM). xliv + 448 pp. Revision of IEEE Std 1003.1-1996 and IEEE Std 1003.2-1992) Open Group Technical Standard Base Specifications, Issue 6.

[23] IEEE. *IEEE Std 1003.1-2001 Standard for Information Technology — Portable Operating System Interface (POSIX) Rationale (Informative)*. IEEE, New York, NY, USA, 2001. xxxiv + 310 pp. Revision of IEEE Std 1003.1-1996 and IEEE Std 1003.2-1992) Open Group Technical Standard Base Specifications, Issue 6.

[24] IEEE. *IEEE Std 1003.1-2001 Standard for Information Technology — Portable Operating System Interface (POSIX) Shell and Utilities, Issue 6*. IEEE, New York, NY, USA, 2001. xxxii + 1090 pp. Revision of IEEE Std 1003.1-1996 and IEEE Std 1003.2-1992) Open Group Technical Standard Base Specifications, Issue 6.

[25] IEEE. *IEEE Std 1003.1-2001 Standard for Information Technology — Portable Operating System Interface (POSIX) System Interfaces, Issue 6*. IEEE, New York, NY, USA, 2001. xxx + 1690 pp. Revision of IEEE Std 1003.1-1996 and IEEE Std 1003.2-1992) Open Group Technical Standard Base Specifications, Issue 6.

[26] Dave Lenckus. Data integrity problem is creating converts: Several trends cause a switch to new systems. Technical report, Business Insurance, December 1997. From the text: "Mr. Dorn estimates there is an 8% to 20% error rate in data, spiking as high as 80% in some cases.".

[27] Donald A. Lewine. *POSIX programmer's guide: writing portable UNIX programs with the POSIX.1 standard*. O'Reilly & Associates, Inc., 981 Chestnut Street, Newton, MA 02164, USA, 1991. ISBN 0-937175-73-0. xxvii + 607 pp. LCCN QA76.76.O63 L487 1991b. US$34.95. March 1994 printing with corrections, updates, and December 1991 Appendix G.

[28] Bil Lewis, Dan LaLiberte, Richard Stallman, and the GNU Manual Group. *GNU Emacs Lisp Reference Manual, for Emacs Version 21*. Free Software Foundation, 59 Temple Place Suite 330, Boston, MA 02111-1307, USA. Phone: 617-542-5942, 2000. ISBN 1-882114-73-6. 974 pp. Two volumes.

[29] Paul Marks. Way back when: Interview with Brewster Kahle. *New Scientist*, 176(2370):46–48, November 2002.

[30] Frank Mittelbach, Michel Goossens, Johannes Braams, David Carlisle, Chris Rowley, Christine Detig, and Joachim Schrod. *The LaTeX Companion*. Tools and Techniques for Computer Typesetting. Addison-Wesley, Reading, MA, USA, second edition, 2004. ISBN 0-201-36299-6. xxvii + 1090 pp. LCCN Z253.4.L38 G66 2004. US$59.99, CAN$86.99.

[31] Office of Program Policy Analysis and Government Accountability. License plate seizure program's error rate still high; program should be abolished. OPPAGA Program Review 00-25, Florida State Legislature, Tallahassee, FL, USA, December 2000. 4 pp. From the abstract: "A department study conducted in October 2000 determined that the error rate for seized license plates was 34.8%.".

[32] Arnold Robbins. *Effective AWK Programming*. O'Reilly & Associates, Inc., 981 Chestnut Street, Newton, MA 02164, USA, third edition, 2001. ISBN 0-596-00070-7. xxiv + 421 pp. LCCN QA76.73.A95 R63 2001. US$39.95.

[33] Arnold Robbins and Nelson H. F. Beebe. *Learning Shell Scripting*. O'Reilly & Associates, Inc., 981 Chestnut Street, Newton, MA 02164, USA, 2004. ca. 512 pp.

[34] Dominique Rodriguez. *L'essentiel de LaTeX et GNU-Emacs: manuel de réalisation de documents scientifiques, CD-ROM TeXlive'4 GNU-Emacs 20.5 pour Windows, exercices corrigés*. Informatiques. Série Réseaux et télécoms. Dunod, Paris, France, 2000. ISBN 2-10-004814-7. xv + 352 pp. Includes CD-ROM.

[35] Michael A. Schoonover, John S. Bowie, and William R. (William Robert) Arnold. *GNU Emacs: UNIX text editing and programming*. Hewlett-Packard Press series. Addison-Wesley,

Reading, MA, USA, 1992. ISBN 0-201-56345-2. xxvii + 609 pp. LCCN QA76.76.T49 S36.

[36] Richard M. Stallman. EMACS: The extensible, customizable, self-documenting display editor. In *Interactive Programming Environments*, pages 300–325. McGraw-Hill, New York, NY, USA, 1984. ISBN 0-07-003885-6. LCCN QA76.6.I5251 1984. US$34.95.

[37] Richard M. Stallman. *GNU Emacs Manual*. GNU Press, Boston, MA, USA, fifteenth edition, 2002. ISBN 1-882114-85-X. 644 (est.) pp. US$45.00.

[38] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishers, Los Altos, CA 94022, USA, second edition, 1999. ISBN 1-55860-570-3. xxxi + 519 pp. LCCN TA1637.W58 1994. US$54.95.

[39] Fred Zlotnick. *The POSIX.1 standard: a programmer's guide*. Benjamin/Cummings Pub. Co., Redwood City, CA, USA, 1991. ISBN 0-8053-9605-5. xi + 379 pp. LCCN QA76.76.063 Z57 1991.

Reports

Editor's note: The following short reports are reprinted from MAPS 31 by permission of the authors and editors. They are included in this issue of *TUGboat* simply for the sake of timeliness; they were not presented at the Practical TEX conference.

MetaPost developments

Taco Hoekwater

The MetaPost system by John Hobby implements a picture-drawing language very much like that of MetaFont except that it outputs Encapsulated PostScript files instead of run-length-encoded bitmaps. MetaPost is a powerful language for producing figures for documents to be printed on PostScript printers, either directly or embedded in TEX documents. It includes facilities for directly integrating TEX text and mathematics with the graphics.

The version number of the MetaPost executable is still well below the 1.0 mark (0.641 is current), but not much has happened in recent years. This situation is far from satisfactory, especially since a fairly large number of bugs are known to exist at this date, but John Hobby simply could not find the time to solve these bugs, let alone handle feature requests.

Resulting from a renewed community interest in MetaPost, last summer a small group of people have made a proposal to Hobby for the creation of a special development group that would take care of the development of MetaPost from then on. Luckily, he agreed, on the condition that he will only allow tested code to be inserted into the MetaPost distribution. Among the currently active group are the following people:

- Karl Berry
- Giuseppe Bilotta
- Hans Hagen
- Taco Hoekwater
- Bogusław Jackowski

Karl Berry has created a home page on the TUG server for MetaPost:

- `http://www.tug.org/metapost`

He also created a mailing list for discussions and questions. Details can be found at:

- `http://www.tug.org/mailman/listinfo/metapost`

Taco Hoekwater has set up a project at Sarovar that hosts a source repository as well as a bug / feature request tracker:

- `http://www.sarovar.org/projects/metapost`

The MetaPost manuals (`mpman`, `mpgraph`, and `mpintro`) have recently been released under a BSD-ish license, with John Hobby's blessing. Dylan Thurston at Debian converted the sources to LATEX, and in the future they will become a standard part of the distribution.

As of today, the known errors in the documentation have been removed, and a number of bugs have already been fixed in the repository. More bugs will be fixed in the near future, and the group hopes that a new bugfix release will be available around EuroTEX 2005.

◇ Taco Hoekwater
`taco@elvenkind.com`

The ℵ (Aleph) project

Giuseppe Bilotta

Abstract

A brief introduction to the ℵ project, a TEX extension providing most Ω and ε-TEX features.

The path

TEX was created by Donald E. Knuth more than 20 years ago. Initially, it was supposed to serve one main purpose (providing a high-quality typesetting workbench for Knuth's books), but was general-purpose and powerful enough to be quickly adopted as a more or less standard environment in the scientific community, thanks to its capability to easily typeset complex formulas.

Usage of TEX outside the scientific/technical domain has always been confined to niche applications, partly due to the absence of high-level formats like LATEX that are geared more towards nontechnical writing, and partly because TEX, in its original design, had very limited support for languages other than English.

Efforts to push the limits of TEX have been made, in at least three different directions, by different teams. This led to the creation of multiple, sometimes incompatible extensions of the original engine; we have for example

- pdfTEX, which gives TEX the capability to produce output directly in PDF form, and introduces micro-typesetting capabilities;

- ε-TEX, based on prior extensions of TEX which added right-to-left typesetting capabilities, which strives to remove some of the structural limitations of TEX while maintaining maximum compatibility;
- Ω (Omega), an effort to bring the TEX world to up-to-date standards and push it towards a multicultural world.

As mentioned before, not all these extensions are compatible with each other; specifically, while pdfTEX and ε-TEX can be merged in a single program, the changes in Ω are so extensive that they put the program in a rather isolated position.

While hopes and desires for a unified TEX extension have always been present, they have not been pressing because the most common format (LATEX) did not make use of ε-TEX extensions, and other formats that did take advantage of those extensions (like ConTEXt) didn't have enough market to be of interest to Ω users. Things started to change recently, as the LATEX team found the original TEX more and more restrictive, and ConTEXt started spreading, its power and flexibility appealing to users outside the domain of Latin scripts.

Birth of a new branch

The Ω project started with the best of intentions and reached some outstanding results; for example:

- 16-bit registers allow Ω to typeset documents too complex to be handled by TEX (or even ε-TEX);
- Ω introduces the concept of ΩTP (Ω Translation Process), a way to transcode texts, therefore allowing the input of text in any script in any encoding, in a font independent way and without the use of active characters; for example, one can write in Greek or Arabic using a plain English keyboard: ΩTPs take care of translating Latin characters (or sequences thereof) into the appropriate Arabic or Greek characters;
- Ω can typeset text in many directions; ε-TEX provided some support for right-to-left typesetting, but Ω brings this capability to any combination of direction (left to right, right to left, top to bottom, bottom to top), easily combining these layouts in the same page.

But Ω presents some characteristics that can make its adoption a difficult choice with an uncertain future. These are the characteristics of an experimental project with a very broad (maybe too broad) final destination, a moving target where stability is only a secondary if not even tertiary target. While this experimental nature of the project is not intrisically negative (on the contrary, it guarantees that the project has enough dynamism to project itself into the future; in fact, it has been what brought Ω to its current status) it does hinder a widespread adoption of the tool for production use.

We therefore have a program, Ω, that has a lot of potential, and that needs to continue to develop; but this potential has to be somewhat harnessed to make the results available in production use contexts. Therefore, we chose to take the Ω code base and use it to start a new project, called ℵ (Aleph),[1] which could provide the power of Ω in an "affordable" manner.

Four main goals were set:

- it had to be stable;
- it had to be fast;
- it had to be powerful;
- it had to be readily available.

Meeting goals

Stability and speed When the ℵ project was started in late 2002, there were two publicly available versions of Ω which could be called "current": version 1.15 and version 1.23 (which I would call the "old" and "new" version respectively).

The reason why the old one was not taken off the distributions when the new one was published is that the new one suffered from "excessive bloat" which rendered it essentially unusable: processing even a simple document with the new version could take from five to ten times longer than processing it in the old version, memory consumption during the processing was at least twice as much, and the resulting DVI was enormous. These shortcomings of the new version were a result of the introduction of a very powerful enhancement, with interesting potential but that needed to be greatly refined before it could become of common usage.

Because of this, most Ω people kept using the old version, which was almost as fast as TEX (although obviously not as slim). This version, on the other hand, had some extremely serious bugs which caused it to crash whenever overfull boxes were present.

Finding which version to choose to base ℵ on was not easy. Indeed, the first release of ℵ (at the time called ε-Ω, Release Candidate 0), which was just a proof of concept that ε-TEX could be merged with Ω, was available both in a 1.15-based version and in a 1.23-based version, although the officially supported one was the former, therefore with an

[1] The project was originally named ε-Ω, since it provided both ε-TEX and Ω features.

implied preference for speed over stability, on the assumption that fixing bugs would have been easier than solving the speed/bloat problem. This has later proved to be indeed the best choice.

Power The immediate outcome of the third goal was that ℵ should have provided both ε-TEX and Ω features; since of course Ω already provides some of the extensions provided by ε-TEX, we could limit ourselves to the programming enhancement (extra marks, protected macros, `\scantokens`, etc.).

Availability This was probably the most important goal, since it would have been the one that "made the difference" with, e.g., Ω 2: ℵ had to be available in a usable status as quickly as possible; this led, among other things, to the choice of stripping from the Ω base the code that dealt with SGML and XML, since it conflicted with the code that implemented the `\middle` primitive in ε-TEX. Priorities led to this decision.

History of releases

The first version of ε-Ω was released in December 2002; while the "official" version merged ε-TEX on Ω 1.15, a parallel release based on Ω 1.23 was also made available. That version was no more stable than any of its components, and had an extra few bugs that crept in during the adaptation of the ε-TEX change files to the Ω structure. In particular, it had all the bugs present in Ω 1.15, which made it scarcely usable due to the major problem with overfull boxes.

The second step was trying to fix the most outstanding bugs coming from the Ω 1.15 codebase. This led to the first version of ε-Ω testable in production-use environments, Release Candidate 1, around June 2003. This was the version presented at TUG 2004.

Subsequent versions finally officially switched to the ℵ name; the last public release (Release Candidate 2) also fixed some other significant bugs and started introducing some minor new features (the most important being the `\boxdir` primitive to retrieve/change the direction of a box, a feature backported from Ω 1.23).

Status

ℵ is actively developed on the TEX Live repository. A mailing list for discussions concerning the present and future of the project, including both the core program itself and any ancillary tool, is available (`aleph@ntg.nl`).

Development, as always, is focused mainly on the discovering (and fixing) of bugs, but discussions on possible future features are welcome. Currently, issues and bugs in ΩTPs and their interaction with certain ε-TEX features (namely protected macros and the `\scantokens` primitive) are the most prominent targets.

Progress

The main focus of ℵ will always be stability. This means, among other things, that each new release is supposed to be at least as stable as the previous one. A test suite analogous to the TRIP test for TEX is being discussed. Indeed, TRIP proved itself a trusted friend in the discovery and resolution of the most notable bugs coming from the Ω 1.15 code base, which dealt with overfull boxes and leaders.

Given the stability of ℵ, it is important to remark that this does not imply a static, frozen behavior (à la TEX); on the contrary, ℵ should be considered a foundation on which to build: experimental projects to test the implementation of new features and ideas are welcome, provided they are developed separately; once they reach enough stability to be available for production uses, they might be candidates for introduction in future versions of ℵ.

Acknowledgements

I wish to thank

- Donald Ervin Knuth, for providing us all with TEX
- John Plaice and Yannis Haralambous, for giving us Ω
- Peter Breitenlohner and the $\mathcal{NTS}$ team, for giving us ε-TEX
- Idris S Hamid, Alan Hoenig and Hans Hagen for pushing me into attempting the merge and supporting me for all this long time
- all the distribution maintainers for their constant feedback, help and support, with particular thanks to Christian Schenk and Fabrice Popineau for their essential help in getting me started with the coding
- everybody in the TEX world for making it the great community it is

◇ Giuseppe Bilotta
Dipartimento di Matematica e Informatica
Università di Catania
viale A. Doria, 6
95125 Catania, Italy
`gip.bilotta@iol.it`

The TEX Live 2004 collection

Hans Hagen

Abstract

The past and future of the TEX Live Collection is described.

Introduction

It must have been in the second half of the eighties that I obtained a copy of *The TEXbook*. It contained what appeared to me as fascinating magic. Then our company purchased MicroTEX, the software program ready to run on a personal computer. It came with a DVI viewer and a printer driver for a matrix printer. From there we moved on to a big pcTEX, Y&Y's dvipsone, BlueSky's outline fonts, now all history.

A few years later we learned of the Dutch speaking TEX User Group NTG and, because we had run into some limitations of TEX —too small a hash— we tried emTEX, which later became part of 4TEX. 4TEX was one of the first TEX distributions on CD-ROM, an integrated set of the most popular programs available in the TEX world. We depended on the yearly updates of 4TEX and later TEX Live, of which version 8 was released in 2003, until today.

Beginning with version 8 TEX Live has become the TEX Collection. It combines an out-of-the-box TEX system and the complete CTAN repository (Comprehensive TEX Archive Network: a snapshot of almost all that is available for TEX users). TEX systems started on floppy disks but soon filled CD-ROM's and now DVD's. An archive of a couple of hundred files grew into tens of thousands.

tree	directories	files	bytes
texmf	3,750	45,000	626 M
texmf-extra	115	1,500	66 M
bin	16	2,500	250 M
source	380	6,900	104 M

If the CTAN archive is included we have a grand total of 138,000 (unzipped even 420,000) files, organized in 10,000 directories, totaling 5,906,870,829 bytes, or about 6 GB.

With version 8 the organizers realized that comprehensive began to become incomprehensible. Even though the TDS, the TEX Directory Structure, had brought some order in grouping files they were still faced with the fact that old TEX systems had been replaced with new systems in a continuous process to adapt to changing operating systems, improved text editors and more sophisticated and generally available viewers and printers. Fundamental changes appeared necessary and are implemented in the TEX Collection 2004. This paper will focus on some of the most important of these changes.

The engine

Donald Knuth's TEX was the ground breaking program that could typeset and be a programming language at the same time. TEX as a typesetting engine has been adapted to handle larger size memory, extended with features, translated into other programming languages, like C, and with the coming of PDF, the Portable Document Format, is now capable of producing PDF output directly with pdfeTEX. The most important change in the 2004 release is that pdfeTEX has become the main TEX engine. pdfeTEX incorporates all 'accepted' extensions with proven reliability, produces DVI output by default, PDF when commanded, and ε-TEX is in there once explicitly enabled. To trigger PDF output ConTEXt users just add as the first line in their text files:

```
% output=pdftex
```

ConTEXt is a monolithic and coherent package of macro definitions that use the programming abilities of almost any TEX to accomplish a large variety of easy to use special typesetting functions.

Other macro packages have often been associated with a specific TEX binary. In practice this leads to several combinations of so-called format files holding the macro definitions and binaries.

For plain TEX the system call (on the command line) and the engine are the same.

system call	format	engine
tex	plain.fmt	tex
etex	etex.efmt	etex
pdftex	pdftex.fmt	pdftex
pdfetex	pdfetex.efmt	pdfetex

For LATEX the system call matches not the engine but the format name. Here the command that starts TEX and loads a format is just a shortcut to calling the engine with a specific format.

system call	format	engine
latex	latex.fmt	tex
pdflatex	pdflatex.fmt	pdftex

For ConTEXt each format is named after the user interface language, the language of commands, messages, keywords, and so forth. This must not be confused with the language of the document text to be typeset. Each interface can handle all document languages.

system call	format	engine	interface
cont-cz	cont-cz.efmt	pdfetex	czech
cont-de	cont-de.efmt	pdfetex	german
cont-en	cont-en.efmt	pdfetex	english
cont-it	cont-it.efmt	pdfetex	italian
cont-nl	cont-nl.efmt	pdfetex	dutch
cont-ro	cont-ro.efmt	pdfetex	romanian

Normally, however, these names are not typed directly; rather, ConTEXt is launched by TEXEXEC, a Perl script that automates many annoying user tasks.

So, what is the importance of the change to PDFETEX in the 2004 Collection? Very little for the user, the system calls are unchanged! For TEX Live system maintenance, however, the change means that the various different TEX binaries can be removed and replaced by a single TEX engine that combines them all: PDFETEX. Extensions like ε-TEX, pdfTEX, MLTEX and encTEX are no longer needed as separate entities. Plain TEX, however, still has the original engine, at least this year. Also, the `.efmt` extension has been dropped; all format file are now `.fmt`.

system call	format	engine
tex	plain.fmt	tex
etex	etex.fmt	pdfetex
pdftex	pdftex.fmt	pdfetex
pdfetex	pdfetex.fmt	pdfetex
latex	latex.fmt	pdfetex
pdflatex	pdflatex.fmt	pdfetex

Because of the growing dependency on this engine PDFETEX has rigourous quality assurance and DANTE, NTG, and TUG have decided to financially support its primary author Hàn Thế Thành to extend and improve the program.

A change such as this is not trivial since it must be certain that existing documents can be processed without change, and macro packages must still believe that the correct binary is available. Macro packages may use undocumented features and nasty tricks to determine what engine is present. Currently PDFTEX is extended to take care of this problem. The configuration file has gone, more extensive map file handling has been implemented, and extensions are being separated to allow for experimental versions (XPDFETEX).

PDFETEX, although quite universally useful, still lacks some features such as Unicode awareness. TEX engine development, therefore, must continue. Those on the ConTEXt mailing list may know Giuseppe Bilotta as an enthusiastic user and advocate of TEX. In 2003 Giuseppe published ϵ-Omega, an extended version of TEX that uses Unicode natively. His initiative evolved into the Aleph project which aims at merging ε-TEX with Omega. This is because some ConTEXt users wanted to use Omega features. LATEX is also moving towards ε-TEX, enhancing the importance of the Aleph initiative.

Those who have become dependent on Omega may get attracted by Aleph's image: stable realware thus giving it a good chance to become the default engine under the Omega based formats on TEX Live. Producing PDF output directly is not a feature but the DVIPDFMX converter can produce the same rich PDF output as PDFETEX does for ConTEXt users.

Latin Modern

What more is new on the TEX Live 2004? First of all, the Latin Modern fonts. This project was funded by user groups. The fonts are extended versions of Computer Modern, with additional characters covering all western languages. Latin Modern will replace the textual part of Computer Modern Roman. For instance, `cmr10`, `aer10`, `plr10`, `csr10` as well as in the near future `vnr10` will be replaced by `lmr10`. This change is downwards compatible. It removes a lot of nearly duplicate files from TEX Live. If all works out well, users will not notice the font change. Of course, the original `cmr10` will still be present.

Currently extra instances are made with a few more glyphs, more kerning pairs. Visual improvements are made based on suggestions by Donald Knuth in his errata documents.

Font files

A more drastic change is that some files have changed places in the TDS tree. Until now the en-

coding (enc) and the fontmap (map) files were located under the DVIPS and PDFTEX paths:

```
texmf/dvips
texmf/dvips/config
texmf/dvips/config/whatever
texmf/pdftex
texmf/pdftex/config
texmf/pdftex/config/whatever
```

The configuration file texmf.cnf informs applications about where to find these encoding and fontmap files. A changed texmf.cnf assures that most applications and users will not encounter problems. The new locations are:

```
texmf/fonts/enc/whatever
texmf/fonts/map/dvips/whatever
texmf/fonts/map/pdftex/whatever
texmf/fonts/lig/whatever
```

Note the new ligature path. It is used by for instance afmtopl. Some changes are already reflected in the current TEX Live version but probably go unnoticed because both old and new locations are supported.

If you install your own fonts you need to relocate your map files. Font metrics remain in their usual place and encoding files are seldom made by users. Instead of relocating another option is to adapt the texmf.cnf file, but this would complicate future updating. It is better to not touch this file.

Scripts

ConTEXt includes some Perl scripts taking care of sorting indexes, managing multiple runs and other chores. Initially, the number of scripts was small and they ended up in a dedicated ConTEXt directory.

Since then other macro packages also come with Perl scripts and ConTEXt added Ruby scripts leading to these paths:

```
texmf/context/perltk
texmf/context/ruby
```

TEX Live uses stubs in the binary path to launch such scripts. The stubs use KPSEWHICH to locate the main script file. For reasons of consistency, maintainance and robust locating, scripts now have their own root path; for ConTEXt, it is:

```
texmf/scripts/context/perl
texmf/scripts/context/ruby
```

Companion files that do not fit in this directory structure remain where they are located presently. In practice users will not notice the changes because the stubs take care of things. Future versions of KPSEWHICH will provide more robust and convenient ways to locate such script files.

Beware: if you write your own scripts you should realize that calls to KPSEWHICH have to be adapted, for instance:

```
kpsewhich -progname=context
  -format="other text files" texexec.pl
```

is now:

```
kpsewhich -progname=context
  -format="texmfscripts" texexec.pl
```

A rather safe way to access files in the texmf tree is to use texmfstart (a Ruby script). This command is described in the manual at the Pragma web site. For now, here are two examples:

```
texmfstart texexec --pdf yourfile
texmfstart --direct scite kpse:texmf.cnf
```

More

AFM files will no longer be distributed in their compressed form (gzip). Engine dependent TEX source files end up in specific paths. Most common users will not notice because users of engine dependent sources have their own way of structuring the directory tree.

The KPSE file searching library and tools get a few more features. A future TEX Live will have a completely rewritten version of this library, one that opens some windows to the future such as automatic updating, remote processing, and fetching resources from zip archives.

Production

Getting TEX Live ready requires an enormous effort. Only a few macro collections are submitted in the right structure. Consequently, much scripting takes place to get the files where they belong in the tree. Interdependencies are not always made clear and maintainers of packages come and go. When the structure changes files need to be relocated. Bugs

in binaries need to be solved. New features have to be tested first. Documentation needs to be updated. Frequently new CD-ROM images are constructed and tested, on all platforms. Thus the TEX Live mailing list is a busy one. Last year we even had a show-stopper. At press time it was discovered that 8-bit file output no longer worked.

Finally, the Collection has to be produced. The 2003 Collection was the first to be distributed on DVD. Even after TEX Live and CTAN were put on the DVD plenty of space was available, so extras were added (in the `texmf-extra` area) and the next release will provide even more. The DVD is one of the first dual layer data DVD's. This meant producing special split ISO-images and proofing of the first DVD: the presses were actually stopped after the first copy for testing!

In 2003 and 2004 DANTE invited those involved in this monster performance to their main annual meeting, altogether some 15 contributors from all over the world. They discussed the present and the future of such distributions. I leave the reporting of that discussion to the chairman. Happy users of TEX Live, however, should recognize with gratitude that getting this job done is far from trivial and effortless. We all should treasure those who are making TEX Live happen year after year. You can find their names on the cover of the DVD and in the documentation.

Summary

When TEX Live 2004 shows up in your postbox, update and things will work as usual. If you have your own fonts installed, however, you need to relocate your personal mapfiles to `.../fonts/map`, and run `mktexlsr` to update your files database. Also, if your scripts use KPSEWHICH, check them.

◇ Hans Hagen
Pragma ADE, The Netherlands
`pragma@wxs.nl`

Calendar

2004–2005

Oct 30 – Apr 17 "Belles Lettres: The Art of Typography", exhibition at the San Francisco Museum of Modern Art. For information, visit `http://www.sfmoma.org/exhibitions/`.

2005

Jan 3 – 7 Rare Book School, University of Virginia, January Sessions in New York City. Two one-week courses: The printed book in the West since 1800, and Book illustration processes to 1890. For information, visit `http://www.virginia.edu/oldbooks`.

Jan 12 – Feb 17 Hans Schmoller: the Penguin Years. An exhibition at the St. Bride Printing Library, London, England. For information, visit `http://www.stbride.org/events.html`.

Jan 18 – Feb 25 In Flight: A traveling juried exhibition of books by members of the Guild of Book Workers. Scripps College, Claremont, California. Sites and dates are listed at `http://palimpsest.stanford.edu/byorg/gbw`.

Feb 23 – 25 Seybold Seminars, New York. For information, visit `http://www.seybold365.com/2005/`.

EuroTEX 2005
Abbaye des Prémontrés (Pont-à-Mousson, France).

Mar 7 – 11 The 15^{th} Annual Meeting of the European TEX Users, and the $2^{2^{2^{2^{-\infty}}}}$ anniversary of both DANTE and GUTenberg, "Let's TEX Together". For information, visit `http://www.gutenberg.eu.org/eurotex2005/`.

Mar 10 – Apr 22 In Flight: A traveling juried exhibition of books by members of the Guild of Book Workers. Rochester Institute of Technology, Rochester, New York. Sites and dates are listed at `http://palimpsest.stanford.edu/byorg/gbw`.

Apr 6 – 8 27^{th} Internationalization and Unicode Conference, "Unicode, Cultural Diversity, and Multilingual Computing". Berlin, Germany. For information, visit `http://www.unicode.org/iuc/iuc27/`.

Apr 14 – 16 TYPO.GRAPHIC.BEIRUT 2005 Conference, Lebanese American University, Beirut, Lebanon. For information visit `http://www.atypi.org/` and look for the entry under "News from members".

Apr 25 – 28 Book History Workshop, Institute d'histoire du livre, Lyon, France. For information, visit `http://ihl.enssib.fr/`.

Apr 26 Harry Carter, man of type: lecture by Martyn Thomas at the St. Bride Printing Library, London, England. For information, visit `http://www.stbride.org/events.html`.

Apr 30 – May 3 BachoTEX 2005, 13^{th} annual meeting of the Polish TEX Users' Group (GUST), "The Art of TEX Programming", Bachotek, Brodnica Lake District, Poland. For information, visit `http://www.gust.org.pl/BachoTeX/2005/`.

May 10 – Jul 17 In Flight: A traveling juried exhibition of books by members of the Guild of Book Workers. University of Texas, Austin, Texas. Sites and dates are listed at `http://palimpsest.stanford.edu/byorg/gbw`.

May 11 – Jun 16 From chisel to pen: inscriptional letterforms from early Christian Wales. An exhibition at the St. Bride Printing Library, London, England. For information, visit `http://www.stbride.org/events.html`.

Status as of 17 March 2005

For additional information on TUG-sponsored events listed here, contact the TUG office (+1 503 223-9994, fax: +1 503 223-3960, e-mail: `office@tug.org`). For events sponsored by other organizations, please use the contact address provided.

An updated version of this calendar is online at `http://www.tug.org/calendar/`.

Additional type-related events are listed in the Typophile calendar, at `http://www.icalx.com/html/typophile/month.php?cal=Typophile`.

May 22 – 27 Book History at A&M: The Fourth Annual Texas A&M Workshop on the History of Books and Printing. Texas A&M University, College Station, Texas. For information, visit http://lib-oldweb.tamu.edu/cushing/bookhistory/2005.html.

May 24 – 27 XTech Conference, "XML, the Web and Beyond", Amsterdam RAI Centre, Netherlands. For information, visit http://www.xtech-conference.org/.

May 25 – 28 CIDE.8, Conférence Internationale sur le Document Electronique, "Multilingualism", Beirut, Lebanon. For information, visit http://www.certic.unicaen.fr/cide8/.

Jun 1 – 3 Society for Scholarly Publishing, 27th annual meeting, "Expanding the World of Scholarly Publishing", Boston, Massachusetts. For information, visit http://www.sspnet.org.

Jun 6 – 9 Seybold Seminars, Amsterdam. For information, visit http://www.seybold365.com/2005/.

Jun 6 – Jul 29 Rare Book School, University of Virginia, Charlottesville, Virginia. Many one-week courses on topics concerning typography, bookbinding, calligraphy, printing, electronic texts, and more. For information, visit http://www.virginia.edu/oldbooks.

Practical TEX 2005
Friday Center for Continuing Education, Chapel Hill, North Carolina.

Jun 14 – 17 Workshops and presentations on LaTeX, TeX, ConTeXt, and more. For information, visit http://www.tug.org/practicaltex2005/.

Jun 15 – 18 ALLC/ACH-2005, Joint International Conference of the Association for Computers and the Humanities, and Association for Literary and Linguistic Computing, "The International Conference on Humanities Computing and Digital Scholarship", University of Victoria, British Columbia. For information, visit http://web.uvic.ca/hrd/achallc2005/ or the organization web site at http://www.ach.org.

Jun 20 – 23 Seybold Seminars Amsterdam 2005, Netherlands. For information, visit http://www.seybold365.com/2005/.

Jun 24 – 26 NTG 35th meeting, Terschelling, Netherlands. For information, visit http://www.ntg.nl/bijeen/bijeen35.html.

Jul 14 – 17 SHARP Conference (Society for the History of Authorship, Reading and Publishing), "Navigating Texts and Contexts". Dalhousie University, Halifax, Canada. For information, visit http://sharpweb.org/ or http://www.dal.ca/~sharp05/.

Jul 20 – 24 TypeCon2005, Type Directors Club, New York City. For information, visit http://www.tdc.org/news/\webbreak 2004typecon2005.html.

Jul 31 – Aug 4 SIGGRAPH 2005, Los Angeles, California. For information, visit http://www.siggraph.org/s2005/.

Aug 1 – 5 *Extreme* Markup Languages 2005, Montréal, Québec. For information, visit http://www.extrememarkup.com/extreme/.

TUG 2005
Wuhan, China.

Aug 23 – 25 The 26th annual meeting of the TeX Users Group. For information, visit http://www.tug.org/tug2005/.

Sep 7 – Oct 6 The Graven Image Press: Lettercutting and visual metaphor in the work of Stan Greer. An exhibition at the St. Bride Printing Library, London, England. For information, visit http://www.stbride.org/events.html.

Sep 11 – 14 Seybold Seminars, Chicago. For information, visit http://www.seybold365.com/2005/.

Sep 15 – 18 Association Typographique Internationale (ATypI) annual conference, Helsinki, Finland. For information, visit http://www.atypi.org/.

Sep 22 – 23 American Printing History Association conference, "[r]Evolution in Print: New Work in Printing History & Practice", Mills College, Oakland, California. For information, visit http://www.printinghistory.org/htm/conference/.

Oct 10 – 12 Fourth Annual St. Bride Conference, "Temporary Type", London, England. For information, visit http://www.stbride.org/conference.html.

Nov 2 – 4 ACM Symposium on Document Engineering, Bristol, UK. For information, visit http://www.documentengineering.org/.

Nov 29 – Dec 2 Seybold Seminars, San Francisco. For information, visit http://www.seybold365.com/2005/.

TUG 2005
International Typesetting Conference
Announcement and Call for Papers

TUG 2005 will be held in Wuhan, China from August 23–25, 2005. CTUG (Chinese TEX User Group) has committed to undertake the conference affairs.

Wuhan is close to the birthplace of Taoism and the Three Gorges Reservoir. China is also the birthplace of typography in ancient times, and is simply a very interesting place to go.

For more information, see the conference web page at `http://tug.org/tug2005`, or email `tug2005@tug.org`.

Call for papers

Please submit a title and abstract for papers or presentations by April 1, 2005, via email to `tug2005@tug.org`. Any TEX-related topic will be considered.

Conference fees

The conference fees and deadlines for members of any TEX user group (in US dollars):

Early registration	May 20, 2005	$100
Normal registration	July 1, 2005	$220
Late registration	August 1, 2005	$380

In all cases, non-user group members add $20.

Conference bursary

Some financial assistance is available. The application deadline is March 25, 2005. Please see `http://tug.org/bursary` for details.

Hope to see you there!

2005 TEX Users Group Membership Form

TUG membership rates are listed below. Please check the appropriate boxes and mail the completed form with payment (in US dollars) to the mailing address at left. If paying by credit/debit card, you may alternatively fax the form to the number at left or join online at `http://tug.org/join.html`. The web page also provides more information than we have room for here.

Status (check one) ☐ New member ☐ Renewing member

	Rate	Amount
☐ **Early bird membership** for 2005 After May 31, dues are $75.	$65	_____
☐ **Special membership** for 2005 You may join at this special rate ($45 after May 31) if you are a senior (62+), student, new graduate, or from a country with a modest economy. Please circle accordingly. See `http://tug.org/join.html` for more information.	$35	_____
☐ **Subscription** for 2005 (non-voting)	$85	_____
☐ **Institutional membership** for 2005 Includes up to eight individual memberships.	$500	_____
☐ **Send me CTAN 2005 on CD** (shipped on DVD to everyone)	n/a	
☐ **If instead of TEX Live 2005** with your membership, you want the 2004 software delivered right away, check here.	n/a	
Last year's materials (in addition to 2005)		
☐ *TUGboat* volume for 2004 (3 issues)	$20	_____
☐ TEX Collection 2004 2 CD's & 1 DVD with proTEXt, TEX Live, CTAN.	$20	_____
☐ CTAN 2004 CD-ROMs	$15	_____
Voluntary donations		
☐ General TUG contribution		_____
☐ Bursary Fund contribution Financial assistance for attending the TUG Annual Meeting.		_____
☐ TEX Development Fund contribution Financial assistance for technical projects.		_____
Total $		_____

Tax deduction: $30 of the early bird membership fee is deductible, at least in the US.
Multi-year orders: To join for more than one year at this year's rate, just multiply.

Payment (check one) ☐ Payment enclosed ☐ Visa/MasterCard/AmEx

Account Number: _____ Exp. date: _____

Signature: _____

Privacy: TUG uses your personal information only to send products, publications, notices, and (for voting members) official ballots. TUG does not sell or otherwise provide its membership list to anyone.

Electronic notices will generally reach you much earlier than printed ones. However, you may choose not to receive any email from TUG, if you prefer.

☐ Do *not* send me any TUG notices via email.

Name _____

Department _____

Institution _____

Address _____

_____

City _____ State/Province _____

Postal code _____ Country _____

Email address _____

Phone _____ Fax _____

Position _____ Affiliation _____

Promoting the use of TEX throughout the world.

mailing address:
P.O. Box 2311
Portland, OR 97208-2311 USA

shipping address:
1466 NW Naito PKWY, Suite 3141
Portland, OR 97209-2820 USA

phone: +1 503-223-9994
fax: +1 503-223-3960
email: `office@tug.org`
web: `http://www.tug.org`

Institutional Members

American Mathematical Society,
Providence, *Rhode Island*

Banca d'Italia,
Roma, *Italy*

Center for Computing Science,
Bowie, *Maryland*

Certicom Corp.,
Mississauga, *Ontario Canada*

CNRS - IDRIS,
Orsay, *France*

CSTUG, *Praha*, *Czech Republic*

Florida State University,
School of Computational Science and Information Technology,
Tallahassee, *Florida*

IBM Corporation,
T J Watson Research Center,
Yorktown, *New York*

Institute for Advanced Study,
Princeton, *New Jersey*

Institute for Defense Analyses,
Center for Communications Research, *Princeton*, *New Jersey*

KTH Royal Institute of Technology, *Stockholm*, *Sweden*

Masaryk University,
Faculty of Informatics,
Brno, *Czechoslovakia*

New York University,
Academic Computing Facility,
New York, *New York*

Princeton University,
Department of Mathematics,
Princeton, *New Jersey*

Springer-Verlag Heidelberg,
Heidelberg, *Germany*

Stanford Linear Accelerator Center (SLAC),
Stanford, *California*

Stanford University,
Computer Science Department,
Stanford, *California*

Stockholm University,
Department of Mathematics,
Stockholm, *Sweden*

University College, Cork,
Computer Centre,
Cork, *Ireland*

University of Delaware,
Computing and Network Services,
Newark, *Delaware*

Université Laval,
Ste-Foy, *Québec*, *Canada*

University of Oslo,
Institute of Informatics,
Blindern, *Oslo*, *Norway*

Uppsala University,
Uppsala, *Sweden*

Vanderbilt University,
Nashville, *Tennessee*

TEX Consultants

Ogawa, Arthur
40453 Cherokee Oaks Drive
Three Rivers, CA 93271-9743
(209) 561-4585
Email: arthur_ogawa@teleport.com
Bookbuilding services, including design, copyedit, art, and composition; color is my speciality. Custom TEX macros and LATEX2ε document classes and packages. Instruction, support, and consultation for workgroups and authors. Application development in LATEX, TEX, SGML, PostScript, Java, and C++. Database and corporate publishing. Extensive references.

Veytsman, Boris
2239 Double Eagle Ct.
Reston, VA 20191
(703) 860-0013
Email: boris@lk.net
I provide training, consulting, software design and implementation for Unix, Perl, SQL, TEX, and LATEX. I have authored several popular packages for LATEX and `latex2html`. I have contributed to several web-based projects for generating and typesetting reports.
For more information please visit my web page:
http://users.lk.net/~borisv.

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 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.

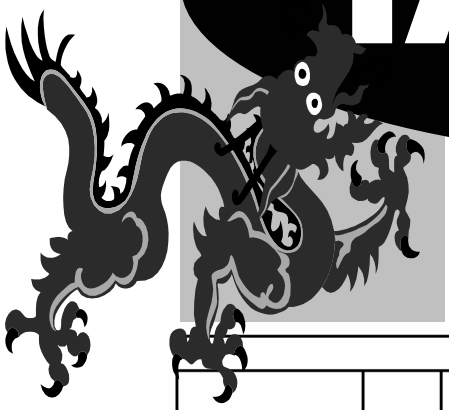
The TUG office mentions the consultants listed here to people seeking TEX workers. If you'd like to be included, or place a larger ad in *TUGboat*, please contact the office or see our web pages:

TEX Users Group
1466 NW Naito Parkway, Suite 3141
Portland, OR 97208-2311, U.S.A.

Phone: +1 503 223-9994
Fax: +1 503 223-3960
Email: office@tug.org
Web: http://tug.org/consultants.html
http://tug.org/TUGboat/advertising.html

EASY TABLE

KHANH HA



A TEX Table Macro Package

The usage of \xrow

STYLE	SIZE	ITEM NO. PLEASE INCLUDE ON ORDER FORM					FIRST† QUALITY 6 PAIR PRICE	SAVE 40% 6 PAIR PRICE	SAVE 50% 12 PAIR PRICE
		NUDE	BEIGE	TAN	TAUPE	SHEER BLACK			
SANDALFOOT	ONE SIZE	29	2021	30	31	2507	$5.10	$3.00	$5.10
REINFORCED TOE		32	2022	33	2023	2024			

†Manufacturer's suggested retail price.

CODES:

```
\eightpoint\aligncen\toprul
\tplain{1}{The usage of \\xrow}
\toprul
\tab{&&\bspan[3-7]ITEM NO.\nl PLEASE INCLUDE ON ORDER FORM\et}
\xbspan[3-7]\prul[3-7,3pt,1]
\tab{\xrow{STYLE}&\xrow{SIZE}&NUDE&BEIGE&TAN&TAUPE&SHEER BLACK&
\xrow{FIRST$\dagger$ QUALITY\nl 6 PAIR PRICE}&
\xrow{SAVE 40\%\nl 6 PAIR PRICE}&\xrow{SAVE 50\%\nl 12 PAIR PRICE}\et}
\hrul{6pt}{1}
\tab{SANDALFOOT&& 29& 2021& 30& 31& 2507\et}
\prul[1-1,3pt,1] \mrul[3-7,3pt,1] % when multipartial hrules on the same row
\tab{REINFORCED TOE&\xrow{ONE SIZE}& 32& 2022& 33& 2023& 2024&\xrow{\$5.10}&
\xrow{\$3.00}&\xrow{\$5.10}\et}
\hrul{6pt}{1}\sevenpoint\tfntindent{1em}
\normalfnt{$\dagger$Manufacturer's suggested retail price.}
```

◊ ◊ ◊

This TEX table macro package rivals the best of the commercial typesetting systems.

Cost? $49.95

- Dynamic table setting by template control
- Multiple mixed column spanners, subspanners, and row spanners
- Easy routines to split table footnotes and break extremely long tables
- Partial hrules, floating hrules anywhere, any length on exact baselineskip
- Automatic decimal alignment, or any special character, in irregular tables
- End columns with \et command anywhere and all vrules are automatically drawn
- Old article: http://tug.org/TUGboat/Articles/tb11-2/tb28ha.pdf
- Version10.04, far superior to the original 1989 version, is now available.

Inquire about EASY TABLE at:

www.authorkhanhha.com/EZ

301-523-4242

The TEX Users Group gratefully acknowledges Apple Computer's generous contributions, especially to the *Practical TEX 2004* and *TUG 2003* Conferences.

Thank you.

The Apple Store in San Francisco is located at One Stockton Street, San Francisco, CA 94108[1]

(This was typeset with the TEX variant XETEX[2] created by Jonathan Kew using the Apple System fonts Hoefler Text by Jonathan Hoefler, Zapfino by Hermann Zapf and Skia by Matthew Carter.)

[1] http://www.apple.com/retail/sanfrancisco [2] http://scripts.sil.org/xetex

Practical TEX 2005
Workshops and Presentations:
LATEX, TEX, ConTEXt,
and more

June 14–17, 2005
Friday Center for Continuing Education
Chapel Hill, North Carolina, USA

http://tug.org/practicaltex2005
conferences@tug.org

Keynote address: *Nelson Beebe, University of Utah*

Who should attend?

Mathematicians ∗ University & corporate (LA)TEX documentation staff ∗ Students ∗ Publishing company production staff ∗ Scientists ∗ Researchers

. . . and anyone who uses or is considering using the LATEX and TEX technical documentation system.

Further information

This four-day conference focuses on practical techniques for document production using LATEX, TEX, ConTEXt, MetaPost, and friends. It includes one day of classes and tutorials, followed by three days of presentations and workshops.

Conference attendees will enjoy an opening night reception and an (optional) banquet one evening. Coffee and lunch will be served each day of the meeting. Located in historic Chapel Hill, North Carolina.

Conference fee, hotel, and other information is available on the web site.

Pre-conference classes

On the first day, June 14, courses will be offered focusing on specific areas: Intermediate LATEX, Introduction to ConTEXt, and TEX on the Web.

- **Call for papers:** If you'd like to make a presentation, on any TEX-related topic, please email us by **March 1, 2005**.
- **Registration** forms and hotel reservation information are on the web site. Early bird discount for registrations before **March 31, 2005**.
- **Sponsorship:** If you'd like to promote your TEX products and services, or otherwise support the conference, see the web site for donation, sponsorship, advertising options. We are very grateful to Duke University for major support.

Hope to see you there!

Sponsored by the TEX Users Group.