

TUGBOAT

Volume 24, Number 1 / 2003
2003 Annual Meeting Proceedings

	2	William Adams / <i>Editorial remarks</i>
	3	Karl Berry / <i>Welcome to TUG 2003</i>
	5	Conference program and delegates
Publishing	10	Pablo Rosell-González / <i>Return to the classics</i>
	18	Alun Moon / <i>Digital illumination</i>
	23	Abstracts (Daly & Richter, Hagen, Janc, Moore)
Bibliographies	25	Oren Patashnik / <i>BIB_{TEX} yesterday, today, and tomorrow</i>
	31	Abstracts (Daly, Hufflen)
Interfaces	32	Richard Koch / <i>TeXShop in 2003</i>
	39	Francesco Costanzo and Gary Gray / <i>Creating labeled stand-alone figures in L_AT_EX using WARMreader and Adobe Illustrator under Mac OS X</i>
	50	Kalle Saastamoinen, Jaakko Ketola, Tuukka Kurppa and Liisa Torikka / <i>Enabling Web access to a database of calculus problems using L_AT_EX, PHP and L_AT_EX2HTML</i>
	53	Luca Padovani / <i>MathML formatting with T_EX rules and T_EX fonts</i>
	62	Abstracts (Gurari, Quirk, Hagen, Hefferon, Lehmké)
Fonts	64	Bogusław Jackowski and Janusz M. Nowacki / <i>Latin Modern: Enhancing Computer Modern with accents, accents, accents</i>
	75	Vladimir Volovich / <i>CM-Super: Automatic creation of efficient Type 1 fonts from METAFONT fonts</i>
	79	Hàn Thê Thành / <i>Making Type 1 fonts for Vietnamese</i>
	85	Candy L. K. Yiu and Wai Wong / <i>Chinese character synthesis using METAPOST</i>
	94	Abstracts (Adams)
Multilingual Document Processing	95	Giuseppe Bilotta / <i>ε-Ω: A step towards the future with a look at the past</i>
	97	Alan Hoenig / <i>Makor: Typesetting Hebrew with Omega</i>
	105	John Plaice, Paul Swoboda, Yannis Haralambous and Chris Rowley / <i>A multidimensional approach to typesetting</i>
	115	Abstracts (Cho)
Tools	116	Alun Moon / <i>Literate programming meets UML</i>
	120	Giuseppe Bilotta / <i>Math in ConT_EXt: Bridging the gap with (AMS-)L_AT_EX</i>
	122	Abstracts (Bazargan et al., Esser, Lehmké, Popineau, Wierda)
Sociology	124	Gary Gray and Francesco Costanzo / <i>Experiences and lessons learned teaching L_AT_EX to university students</i>
	132	Alexandre Gaudeul / <i>The (L_A)T_EX project: A case study of open source software</i>
	146	Andrzej Odyńiec / <i>A Polished T_EX story</i>
	151	Abstracts (Bibby, Ranade)
News & Announcements	152	Calendar
	c3	TUG 2004 conference / <i>Xanthi, Greece, August 30–September 3, 2004</i>
	154	TUG 2005 conference / <i>Wuhan, China, August 23–25, 2005</i>
TUG Business	155	Recognition of support from Apple
	155	Institutional members
Advertisements	156	T _E X consulting and production services
	153	<i>The L_AT_EX Companion</i> , 2 nd edition, by Frank Mittelbach et al.
	154	<i>Guide to L_AT_EX</i> , 4 th edition, by Helmut Kopka and Patrick W. Daly

T_EX Users Group

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

Memberships and Subscriptions

2003 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 T_EX 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 T_EX and the T_EX Users Group. For further information, contact the TUG office (office@tug.org), or see our web site.

T_EX is a trademark of the American Mathematical Society.

Copyright © 2003 T_EX Users Group.

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

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

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

Printed in U.S.A.

Board of Directors

Donald Knuth, *Grand Wizard of T_EX-arcana*[†]
Karl Berry, *President**
Kaja Christiansen*, *Vice President*
Sam Rhoads*, *Treasurer*
Susan DeMeritt*, *Secretary*
Barbara Beeton
Jim Hefferon
Stephanie Hogue
Ross Moore
Arthur Ogawa
Gerree Pecht
Cheryl Ponchin
Michael Sofka
Philip Taylor
Raymond Goucher, *Founding Executive Director*[†]
Hermann Zapf, *Wizard of Fonts*[†]

*member of executive committee

[†]honorary

Addresses

General correspondence,
payments, etc.
T_EX Users Group
P. O. Box 2311
Portland, OR 97208-2311
U.S.A.
Delivery services,
parcels, visitors
T_EX 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
T_EX 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: July 2004]

2003 Annual Meeting Proceedings

TeX Users Group
Twenty-fourth Annual Meeting
Waikoloa, Hawai'i
July 20–24, 2003

TUGBOAT

COMMUNICATIONS OF THE T_EX USERS GROUP

TUGBOAT EDITOR BARBARA BEETON

PROCEEDINGS EDITOR WILLIAM ADAMS

VOLUME 24, NUMBER 1

PORTLAND

•

OREGON

•

•

2003

U.S.A.

Editor's introduction

William Adams

75 Utley Drive; Camp Hill, PA USA

willadams@aol.com

<http://members.aol.com/willadams>

As I write this, it has now been very nearly a year since this conference took place. My apologies for the lengthy process, but I believe that the final results are worth the wait. This anniversary conference provided a unique overview of both the past history of \TeX and related systems, and concrete examples of its future usage — these proceedings encapsulate as much as is possible. More importantly, in addition to a “mere” recounting of history (“25 Years of \TeX and \METAFONT ”, “That Pesky Lion...”, “The spread of \TeX in India”, “ \TeX Heritage Panel”) or an analysis of what happened and why and a consideration of the future, (“The $(\text{\La})\text{\TeX}$ project: A case study of open-source software”, “The future of European LUGs”) they capture for posterity the unique nature of \TeX and its siblings and descendants (“The return to the classics”, “Digital illumination”, “Chinese character synthesis using \METAPOST ”, “POV-ray: A 3D graphics tool for \TeX ”) and the significant impact which it has had (or will continue to have) on academia (“ \BibTeX yesterday, today, and tomorrow”, “ \ML-BibTeX ’s version 1.3”, “Enabling web-access to a database of calculus problems using \LaTeX , PHP and \LaTeX2HTML ”, “ \XeMTeX : An integrated platform for high-quality scientific typesetting” “Scientific publishing with \pdfLaTeX ”) and publishing (“Typesetting nightmares”, “ \EXAMPLE ”, “ \ERCOTeX ”, “ \LaTeX in real-world math typesetting”) and its capabilities (“Using \TeX to manage IT for a Mathematics Congress”, “ \TeXPower — Dynamic presentations with \LaTeX ”, “A multidimensional approach to typesetting”, “A new system for typesetting Hebrew with Omega”, “Literate programming meets UML”), including significant interactions with long-standing (or awaited) standards implementations (“ \MathML formatting with \TeX rules and \TeX fonts”, “From \LaTeX to \MathML and beyond”, “Accents, accents, accents... Enhancing CM fonts with ‘funny characters’” “ \Math in \ConTeXt : Bridging the gap with \AMS-LaTeX ”) but to some extent, the nature of the effort which was necessary to make this possible (“A Polished \TeX story”), and a guide to how one can make use of all of this (“Web Services

for CTAN”, “Abusing \TeX : `custom-bib` as an example”, “Programming dynamic \LaTeX documents”, “Creating labeled ‘stand-alone’ figures in \LaTeX using \WARMreader and Adobe Illustrator under Mac OS X”, “New features in \TeXshop ”, “Experiences and lessons learned teaching \LaTeX to a group of university students”) or how it’s all made possible (“The \teTeX distribution”, “ ϵ - Ω : A step towards the future with a look on the past”, “Automatic creation of efficient Type 1 fonts from \METAFONT fonts”, “`dvipdfmx`, an eXtension of `dvipdfm`”).

I would like to thank everyone who worked to make this conference a success, especially my fellow presenters, all those who attended, and in particular Wendy McKay who very graciously invited me to speak at this conference (and earlier at TUG 2001 in Delaware), and Hans Hagen and Giuseppe Bilotta who very generously offered a post-conference workshop on \ConTeXt .

My sincere thanks also to the TUGboat production team, including Barbara Beeton, Karl Berry and, most especially, Mimi Burbank for their work to produce these proceedings. Karl and Barbara are both far better editors than I and fixed a myriad of errors which had slipped past me, while Mimi managed to create nicely crafted pages in instances where I’d failed utterly.

Ultimately, that’s what much of the use, purpose and intent of \TeX is: a nicely-balanced set of pages, *mise en page*, as Greer Allen describes it, most often in books. I find myself unable to avoid mentioning here a few books which were of especial significance to me in my traveling to and from Hawai’i: *Diver’s Down* by Hal Gordon, which was the first “chapter” book which I read as a youth, and which instilled in me a desire (now fulfilled) to visit Hawai’i, *Bridge of Birds* by Barry Hughart, a delightful fantasy which I re-read on the plane ride over and foisted off on a pair of presenters to make room for swag; and *Calligraphic Type Design in the Digital Age: An Exhibition in Honor of the Contributions of Hermann and Gudrun Zapf*, which should tide the reader over until my presentation is published in the next TUGboat.

Aloha!

Welcome to TUG 2003

Karl Berry

P. O. Box 2311

Portland, OR 97208-2311

USA

karl@freefriends.org

<http://freefriends.org/~karl/>

Welcome to sunny Hawaii, and greetings from a cold and rainy Oregon! Thank you all for attending this conference celebrating \TeX 's 25th anniversary. Unfortunately I can't be there with you in person, but Wendy graciously invited me to give this opening address via audio tape anyway. I am honored to do so in my capacity as the next president of the \TeX Users Group.

A silver anniversary conference is a time for reflection. I first became aware of \TeX in 1983, when I was an undergraduate at Dartmouth College. The college had just bought its first Unix system (a hot new VAX 11/730), and the computer science department had just hired a new professor out of Carnegie-Mellon, and he brought with him a 1/4" magnetic tape of software distributions, notably including \TeX . (The only other software I remember from that tape was Gandalf, a structured editor for Pascal and other languages.)

Professor Sherman turned me loose, informing me there was a `README` file in the top level directory, and I should read it. I happily started trying to understand how all these crazy `.web`, `.ch`, `Makefile`'s, and everything else tangled together. I eventually managed to get things installed, more or less, since fortunately BSD 4.2 came with a Pascal compiler. We couldn't actually see any output, though, since the only output device we had was a QMS laser printer. (Bitmapped screens were not even a dream, all our work was on Zenith z19 terminals. At least we'd graduated from yellow-paper teletypes.) So my next project was to adapt an existing DVI driver (unfortunately I've forgotten which one). A week or two of hacking later, the first page we saw was A.U. Thor's Short Story. Ah, bliss.

The next job was to get the latest version of `tex.web`. Dartmouth didn't have a direct ARPAnet connection at the time, but it was on CSnet. Professor Sherman typed some magic incantations to get to his account at CMU, and from there he typed `ftp score.stanford.edu`, and what the computer said back was: `Welcome to sunny California`. I turned to him, gaping, and said, what? Really? He

just shrugged, being an old hand at this network stuff, and said, sure, that computer's in California. Wow!!, I said.

Well, that was then and this is now. Here we are in the 21st century, and as we all know, the computing world has changed enormously. TUG has changed and grown, from its beginnings at the developer conferences at Stanford in \TeX 's early days to an organization with over two thousand members from about 70 countries. (I'll be especially sorry to miss Nelson's keynote address and the \TeX heritage session at this conference, to learn more about \TeX and TUG's history.)

During my term as president, I hope to make TUG more accessible and attractive to \TeX users worldwide, collaborate wherever possible with other \TeX user groups, and serve its current members better. We have an excellent infrastructure in place, both technical (the `tug.org` server, now located in Denmark thanks to Kaja Christiansen and Århus University) and human (our excellent office manager, Robin Laakso), to support any TUG activities.

Some steps along this path have been taken already. First, TUG was finally given tax-exempt charitable status, after much effort by Robin, Mimi Jett, and our lawyer Amy Silliman; this means most contributions to TUG, at least within the US, are now tax-deductible. Also, we awarded the first set of grants from a new \TeX Development Fund earlier this year. We were especially grateful to receive donations to this fund from over 100 individuals and institutions around the world. We're also slowly making progress with getting *TUGboat* caught up — we've published three issues so far this year, and are hard at work on more. We continue to support the annual software distributions of \TeX Live and CTAN (coming soon for 2003!). And, last but far from least, supporting conferences such as this one remain a priority.

In the final analysis, though, TUG is whatever its members and the \TeX community make of it. So if you have any ideas for other activities to undertake, or changes to propose, please speak up, so we

can all work together toward our shared goals.

\TeX itself has also changed and grown dramatically in 25 years, of course. Despite occasional rumours of \TeX 's demise, as far as I can tell such reports are greatly exaggerated. Its mathematical typesetting remains unsurpassed, there are always new projects underway to make the \TeX system ever more powerful and friendly, and most importantly, new users and programmers still appear with some frequency. One of the most interesting recent developments is Mac OS X, based on a Unix kernel, which is bringing \TeX to a large new population of computer users. This conference will dedicate an entire day to Mac OS X work. All in all, I personally remain hopeful about \TeX , and believe its future remains promising.

However, since this is a read-only presentation, I'll have to ask you to contact me separately if you have questions, comments, or suggestions. For the next two years, you can email me as president@tug.org. I never would have guessed it

on that day in 1983 when I first encountered \TeX .

In closing, I'd like to thank Wendy McKay, Patricia Monohon, Hans Hagen, Robin Laakso, and all the other volunteers and organizations who made this conference happen; Malcolm Clark for the great poster; the Hawaii Tourism Authority; and Addison-Wesley, Apple, W.H. Freeman, Integre, Kensington, Math Game House, and RoadTools for their corporate contributions. Also, the even larger group of volunteers who have done so much to support \TeX and TUG over the years, including my fellow directors on the TUG board and the board members of all the other \TeX user groups around the world. Also my fellow developers, who keep sticking to the details to make all the software actually work. And finally, of course, Donald E. Knuth, without whom the world of typesetting over the last 25 years would have been quite a different place. Happy birthday to \TeX !

Thank you for putting up with this recording, and have a great conference.

Organizing Committee

William Adams
Sue DeMeritt
Robin Laakso
Wendy McKay
Patricia Monohon
Anita Schwartz

Program Committee

Hans Hagen
Stephanie Hogue
Jerzy Ludwichowski
Wendy McKay
Ross Moore
Fabrice Popineau
Chris Rowley
Volker Schaa

Sponsors

<i>Addison-Wesley Publishers (UK)</i>	Michael Strang
<i>Addison-Wesley Publishers (USA)</i>	Curt Johnson
<i>Apple Computer Inc.</i>	Brian Frye
<i>Carleton Production Centre (Canada)</i>	Christina Thiele
<i>Math Game House</i>	Hyejung Kang, Seungoh Ryu
<i>Kensington Technology Group</i>	Roma Majumder
<i>Koenig Productions</i>	Cici Koenig
<i>Road Tools</i>	Jim MacEachern
<i>W. H. Freeman & Company / Worth Publishers</i>	Craig Bleyer
<i>Integre: Technical Publishing Company, Inc.</i>	Don Deland
<i>PC-TEX</i>	Lance Carnes
<i>TEX Merchandising Project</i>	Martin Schröder
<i>Y & Y, Inc.</i>	Blenda Horn
<i>University of Hawai'i at Hilo Conference Center</i>	Judith Fox-Goldstein
<i>Hawaii Tourism Authority</i>	
<i>Hawaii Island Economic Development Board</i>	

Grateful acknowledgements

Thanks also to:

- The T_EX user groups DANTE e.V., GUST, GUTenberg, and NTG for additional speaker support.
- Generous individual contributors: Dave Bailey, Greg Black, Daniel Boerner, Harriet Borton, Malcolm Clark, Daniel Haugland, Stephan Lehmke, Oren Patashnik, Pablo Rosell-Gonzalez, Bob Styer, Paul Thompson, Donald Tyson, Robert Utter, Alan Wetmore, Glenn Williams, Camerson Wright.
- Malcolm Clark (Menagerie Fleet Captain, Banbury Sailing Club) for the freeware lion drawing.
- William Adams, Duane Bibby, Bob Kerstetter and Wendy McKay for putting together the fund-raising T_EX Harvest Appeal.

**Participants at the 24th Annual TUG Meeting
July 20–24, 2003, Outrigger Waikoloa Beach Resort
Big Island, Hawai‘i**

Adams, William
ATLIS, Camp Hill, PA, USA

Ase, Harumi
ARS System Corporation,
Chiyoda-ku, Tokyo, Japan

Bailey, Dave
Covestic, Inc.,
Albuquerque, NM, USA

Bazargan, Kaveh
Focal Image Ltd,
Exeter, Devon, UK

Bhat, Harish
California Institute of Technology,
Pasadena, CA, USA

Beebe, Nelson
University of Utah,
Salt Lake City, UT, USA

Beeton, Barbara
American Mathematical Society,
Providence, RI, USA

Bibby, Duane
Lake Havasu City, AZ, USA

Bilotta, Giuseppe
Catania, CT, Italy

Brenton, Steven
Long Beach, CA, USA

Capkun, Jozo
Mississauga, Ontario, Canada

Carnes, Lance
Personal T_EX, Inc.,
Mill Valley, CA, USA

Cho, Jin-Hwan
Korea Institute for Advanced
Study, Seoul, Republic of Korea

Costanzo, Francesco
Penn State University,
University Park, PA, USA

Daly, Patrick William
Max-Planck-Institut für Aeronomie,
Katlenburg-Lindau, Germany

DeMeritt, Sue
IDA/CCR La Jolla,
San Diego, CA, USA

Doob, Michael
University of Manitoba,
Winnipeg, Manitoba, Canada

Esser, Thomas
FinanzIT GmbH,
Hannover, Germany

Frye, Brian
Apple Computer Inc., USA

Fuchs, David
Palo Alto, CA, USA

Gaudeul, Alexandre
GREMAQ - University of
Toulouse, France

Girard, Michel
LAMBESC, France

Gray, Gary
Penn State University,
University Park, PA, USA

Gurari, Eitan
Ohio State University,
Columbus, OH, USA

Hagen, Hans
Pragma ADE,
Hasselt, The Netherlands

Hàn Thế Thành
University of Education,
Ho Chi Minh City, Vietnam

Hefferon, Jim
Saint Michael's College,
Colchester, VT, USA

Hoening, Alan
John Jay College, City
University of New York, USA

Jackowski, Boguslaw
BOP s.c., Gdansk, Poland

Janc, Mirko
INFORMS, Linthicum, MD, USA

Jones, Shannon
Federal Reserve Bank of
Richmond, VA, USA

Kerstetter, Bob
Alt-N Technologies,
Richardson, TX, USA

Koch, Richard
University of Oregon,
Eugene, OR, USA

Laakso, Robin
T_EX Users Group,
Portland, OR, USA

Lehmke, Stephan
Universität Dortmund, Germany

Levine, Jenny
Duke University Press,
Durham, NC, USA

Ludwichowski, Jerzy
Nicolaus Copernicus University,
Toruń, Poland

MacKichan, Barry
Poulsbo, WA, USA

McKay, Wendy
California Institute of Technology,
Pasadena, CA, USA

Monohon, Patricia
Ventura, CA, USA

Montpetit, André
Université de Montréal,
QC, Canada

Moon, Alun
School of Informatics,
University of Northumbria, UK

Moore, Ross
Macquarie University,
Sydney, Australia

Padovani, Luca
University of Bologna, Italy

Patashnik, Oren
San Diego, CA, USA

Plaice, John
The University of New South
Wales, Sydney, NSW, Australia

Ponchin, Cheryl
Institute for Defense Analyses,
Princeton, NJ, USA

Popineau, Fabrice
Supélec Campus de Metz, France

Quirk, James
CCS Division, Los Alamos
National Laboratory, NM, USA

Ranade, Ajit
ABN AMRO Bank,
Mumbai, India

Rhoads, Sam
Honolulu Community College,
Honolulu, HI, USA

Roberts, Kim
HK Typesetting Ltd,
Salford, Lancs., UK

Rosell-González, Pablo
Ciudad Universitaria,
Mexico, Mexico

Rowley, Chris
Open University, London, UK

Saastamoinen, Kalle
Lappeenranta University of
Technology, Lappeenranta,
Finland

Schaa, Volker RW
DANTE e.V.,
Darmstadt, Germany

Sestrich, Heidi
Carnegie Mellon University,
Pittsburgh, PA, USA

Smith, Alistair
Sunrise Setting Ltd,
Torquay, Devon, UK

Stenerson, Jon
Las Cruces, NM, USA

Swanson, Steve
Las Cruces, NM, USA

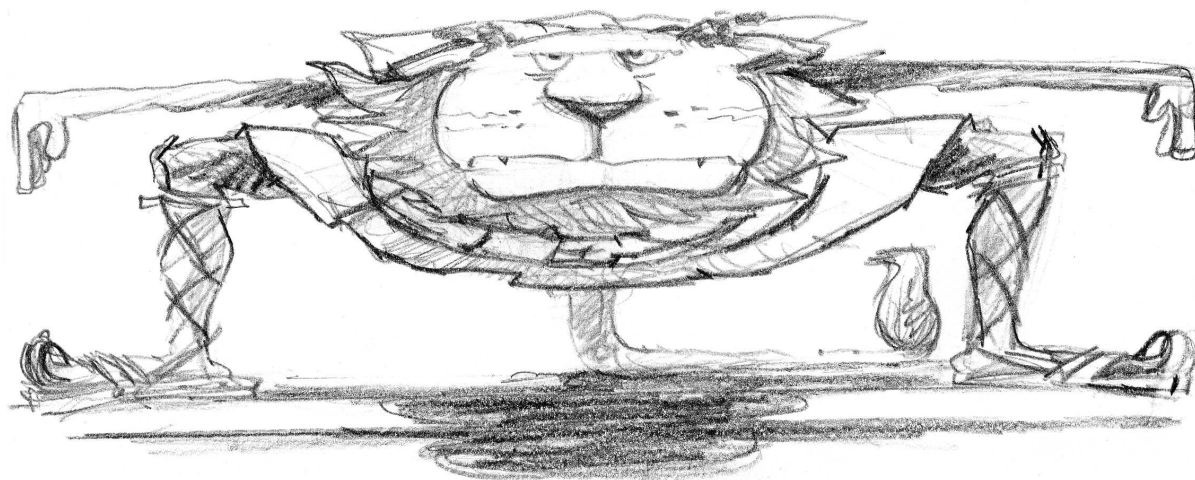
van der Poorten, Alfred J.
ceNTRe for Number Theory
Research, Killara, NSW,
Australia

Volovich, Vladimir
Voronezh State University,
Voronezh, Russia

Wetmore, Alan
Army Research Laboratory,
Adelphi, MD, USA

Wong, Wai
Hong Kong Baptist University,
Kowloon, China

Yiu, Candy L. K.
Hong Kong Baptist University,
Kowloon, China



TUG 2003 Program
the Silver Anniversary — 25 years!
The 24th Annual Meeting and Conference of the T_EX Users Group
July 20–24, 2003

Sunday, July 20, 2003

Official opening and welcome

Don Knuth / A message from the Grand Wizard
Karl Berry / Welcome from the TUG president

Keynote presentations

Nelson Beebe / A T_EX retrospective
Duane Bibby / That pesky lion: 25 years of T_EX and Meta drawings

Talks

Ross Moore / Using T_EX to manage IT for a mathematics congress
Jim Hefferon / Web services for CTAN
Thomas Esser / The t_EX distribution
Pablo Rosell-González / Return to the classics
Alun Moon / Digital illuminations
Hans Hagen / Typesetting nightmares

Monday, July 21, 2003

Talks

Oren Patashnik / B_IB_T_EX yesterday, today, and tomorrow
Patrick Daly / Abusing T_EX: custom-bib as an example
Jean-Michel Hufflen / M_IB_IB_T_EX version 1.3
James Quirk / Programming dynamic L^AT_EX documents
*Stephan Lehmk*e / T_EXPower: Dynamic presentations with L^AT_EX
Luca Padovani / MathML formatting with T_EX rules and T_EX fonts
Eitan Gurari / From L^AT_EX to MathML and beyond
Hans Hagen / eXaM_PLe project
Giuseppe Bilotta / ε - Ω : A step towards the future with a look at the past
John Plaice / Ω , OpenType and the XML world
Peter Flynn / Interfaces to structured text
Ajit Ranade / The spread of T_EX in India: The role of outsourced typesetting
Alexandre Gaudeul / A L^AT_EX case study

Panel: T_EX Heritage

Barbara Beeton / Chair

Tuesday, July 22, 2003

TUG annual meeting

Talks

Gerben Wierda / T_EX on Mac OS X using t_EX and T_EX Live
(remote live presentation from the Netherlands)
Richard Koch / New features of T_EXShop

Workshop

Francesco Costanzo, Gary Gray / The use of the MarkedObjects plug-in, along with WARMreader to create L^AT_EX-annotated figures using Adobe Illustrator under Mac OS X.

Wednesday, July 23, 2003

Talks

John Plaice / A multidimensional approach to typesetting

Candy Yiu, Wai Wong / Chinese character synthesis using METAPOST

Jin-Hwan Cho / dvipdfmx, an eXtension of dvipdfm

Kalle Saastamoinen / Creating a Web-based mathematical interface to databases using L^AT_EX, PHP, and the L^AT_EX2HTML compiler

*Stephan Lehmk*e / ERCOT_EX: Yet another database publishing application of L^AT_EX

Alan Hoenig / A new system for typesetting Hebrew with Ω

William Adams / T_EX at the end: Ω and Zapfino

Bogusław Jackowski / Accents, accents, accents: Enhancing CM fonts with “funny” characters

Vladimir Volovich / Automatic creation of efficient Type 1 fonts from METAFONT sources

Hàn Thê Thành / Making Type 1 fonts for Vietnamese

Mirko Janc / L^AT_EX in Real-World Math Typesetting

Panel: Fonts

Nelson Beebe / Chair

Thursday, July 24, 2003

Talks

Alun Moon / Literate programming meets UML

Fabrice Popineau / X_eL^AT_EX: An integrated platform for high quality scientific typesetting

Patrick Daly / Scientific publishing with pdfL^AT_EX

Gary Gray / Experiences and lessons learned teaching L^AT_EX to university students

Kaveh Bazargan / POV-ray: A 3D graphics tool for T_EX

Giuseppe Bilotta / Math in ConT_EXt: Bridging the gap with $\mathcal{A}\mathcal{M}\mathcal{S}$ -L^AT_EX

Andrzej Odyńec / A Polished T_EX story

Jerzy Ludwichowski / The future of European LUGS

Closing ceremony

Friday, July 25, 2003

Workshop: Getting to know ConT_EXt

Hans Hagen, Giuseppe Bilotta



Return to the classics

Pablo Rosell-González

Coordinación de Cómputo

Facultad de Ciencias, UNAM

México, D.F.

México

`pablo@ciencias.unam.mx`

`http://hipatia.fciencias.unam.mx/~prosell/`

Abstract

When did the width of margins in a page become less important than having a small number of pages in a book? We have been inundated by badly printed material since most publishers are now mostly interested in getting as much profit as possible, leaving the art of publishing aside. The premise of book designers should be not only to create attractive printed material but also to provide a pleasant reading experience.

There are quite a few examples of designs that are now considered classics because they have proven themselves to be aesthetically incomparable. These designs include:

Ternary canon: 2:3 page proportions where the height of the typographic box is equal to the width of the page, the left margin is half the right margin and the top margin is half the bottom margin.

Gutenberg's Göttingen Bible: same as above but in two columns with a wide column separation.

Universal scaling: the page is created when the user defines a unit box and the design takes multiples of three times the unit box; the left margin is the unit's box width, the top margin is the unit's box height, and the right and bottom margins are twice the unit's box width and height respectively.

Diagonal and double diagonal methods: the page dimensions and the left margin are provided by the user and all the other elements are defined in terms of either a diagonal or using both diagonals.

ISO 216: the typographic box is sized either A5 or A6 depending on the choice of A4 or A5 paper size, and the left and top margins are half the right and bottom margins respectively.

2–3–4–6 system: page dimensions and the margin unit are defined by the user, and the typographic box is adapted to have two, three, four, and six times the margin unit as left, top, right and bottom margins respectively.

This paper presents *classics*, the author's new class which allows the user to typeset camera-ready books using any of the above designs. This class uses the `calc` and `geometry` packages to create the design, and `crop` to generate the crop marks. If *classics* is used with Hàn Thê Thành's micro-typographic extensions, the user will get state-of-the-art printed material. Let's recover centuries of publishing tradition!

Introduction

Typography and book design are artistic expressions whose principal goal is to bring the reader the most pleasant reading experience.

Gutenberg, in the XVth century, used over 280 types, including many ligatures and expanded (or contracted) characters, to get almost even interword spaces, and protruding of some left and rightmost characters (marginal kerning). With these, he ob-

tained perfectly visual “gray” boxes when printing his Bible.

Most of these aspects are already achieved by using (L^A)T_EX, and the micro-typographic extensions made by Hàn Th^ế Thành for pdf(L^A)T_EX, explained in [7], complete the picture.

This paper is divided in two sections: the first contains a brief geometric description of some page layouts that have proven to be aesthetic and functional. Some variations obtained from them are included. The second explains the usage of `classics`, a class that allows the user to typeset books using those page layouts. Some instances of the usage of `classics` with protruding characters generated with pdf(L^A)T_EX are included.

Classic page layouts

Traditional published works present carefully studied margin proportions obeying (some of) the following four rules:

1. the diagonal of the typographic box coincides with the diagonal of the page;
2. the height of the typographic box equals the page width;
3. the outer margin is twice the inner margin;
4. the bottom margin is twice the top margin.

First approach to the classics Let $ABCD$ be a page with arbitrary proportions. Consider the following construction for the type area (figure 1) which guarantees rules 1, 3 and 4.

Construct diagonals AC and BM , where M is the midpoint of AD , and O the point of intersection. Take any point P between A and O . From P draw the parallel to AB which intersects BM in Q . From Q trace the parallel to BC which intersects AC in R . To obtain S , draw parallels from R and P to CD and AD respectively. Then the rectangle $PQRS$ will have the same proportions as $ABCD$. Moreover, O divides in the same ratio both PR and AC , i.e., $PR/PO = AC/AO = 1/3$.

Of course we want to leave wide enough margins on our page, so that the area of the typographic box covers no more than, say, 50%. But if we also want to follow rule 2 we must restrict the proportions of our page to no less than $1 : \sqrt{2}$. Table 1 shows some page ratios with the percentage of area their typographic box occupies if its height equals the paper’s width (i.e., following rule 2).

2–3–4–6 approach Take a unit u as half the inner margin of any page. If we take $2u$, $3u$, $4u$, and $6u$ dimensions for the inner, top, outer, and bottom margins respectively we will get a fairly good typographic box — this depends, of course, on the width

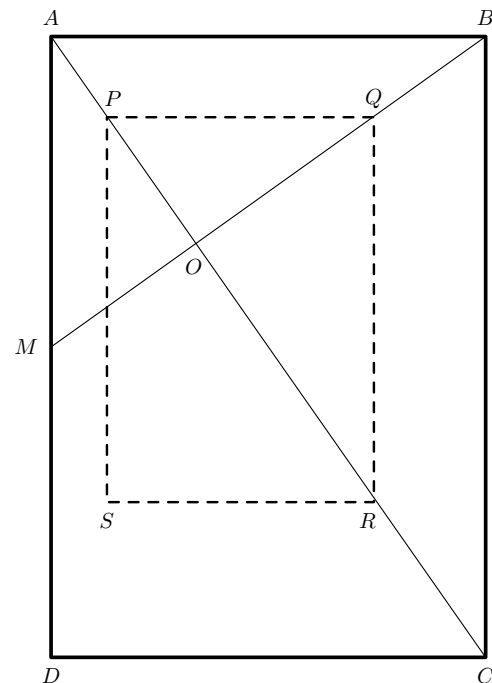


Figure 1: First approach.

Page ratios	Documents	% of area
1 : 2	—	25%
2 : 3	Gutenberg’s Bible	39.5%
2 : 3	Ternary Canon	44.4%
$1 : \sqrt{2}$	ISO 216	50%
3 : 4	Ghostscript’s archA	56%
1 : 1	—	100%

Table 1: Page ratios and typographic box area.

to height ratio of the page and on the value of u respect to the width of the page. In most cases (except for a $2 : 3$ page), the typographic box will not present the same width to height ratio as the page, because if w and h represent the width and height of the page, then the typographic box will be $w - 6u$ in width and $h - 9u$ in height. Suppose the page and type area have the same proportions; then

$$\frac{w}{h} = \frac{w - 6u}{h - 9u} = \frac{2}{3}.$$

This is, nevertheless, a good method for obtaining interesting asymmetrical results.

Ternary Canon Many medieval and Renaissance manuscripts and printed books present a $2 : 3$ width to height page ratio whose typographic box not only follows the four rules mentioned above but also presents the 2–3–4–6 progression in the margins.

Following Van der Graaf’s method, Jan Tschichold [8] published an elegant geometrical way to obtain such a layout. Take a two-page diagram (see figure 2), draw diagonals AC and BD which intersect at M , the midpoint of EF ; draw diagonals EC and ED which intersect BD and AC at G and G' respectively. Draw GG' and perpendiculars through G and G' to AB which intersect it at H and H' . The point of intersection P (P') of EG (EG') and HG' ($H'G$) is the upper left (right) corner of the typographic box. Complete the construction as in figure 1.

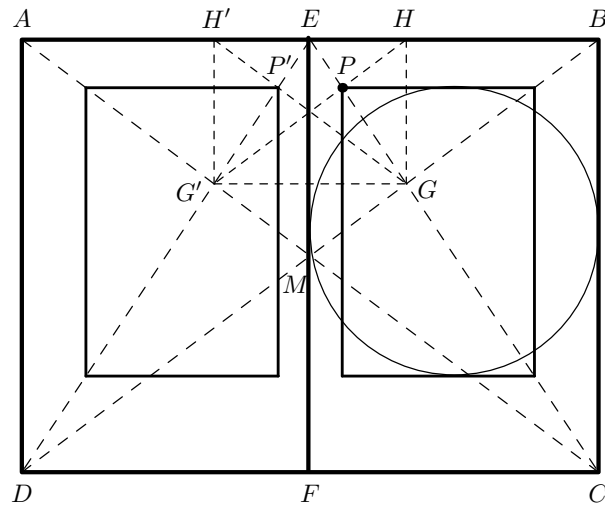


Figure 2: Tschichold’s geometrical construction for the Ternary Canon.

The circle in figure 2 simply shows that the height of the typographic box equals the width of the page. That the margins are in 2–3–4–6 progression follows from the fact that the top left corner of the box divides in $1/9$ both the height and the width of the page. If we fix a unit u which is half the inner margin then the page will have width $18u$ and height $27u$, so the inner margin is $2u$ wide, the top margin is $3u$ high. Because the height of the type area is equal to the width of the page, $18u$, the bottom margin is $27u - (3u + 18u) = 6u$. Moreover, as the typographic box is in the same ratio as the page its width is $12u$, thus the outer margin is $18u - (2u + 12u) = 4u$.

On the other hand, a much different approach is given by Raúl Rosarivo [6], who discovered that creating a 9×9 grid on a $2 : 3$ ratio page (figure 3, left), and positioning the typographic box leaving one column and one row of the grid as inner and top margins, and two columns and rows as outer and bottom margins, gives exactly the same position

and proportions for the typographic box to fulfill the rules and the 2–3–4–6 progression.

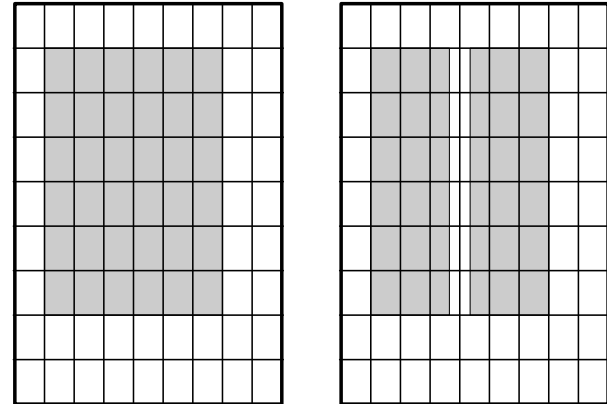


Figure 3: Rosarivo’s 9×9 grid. At the left, the ternary canon typographic box in gray. At the right, Gutenberg’s Bible two column boxes.

In the ternary canon model, the typographic box will cover only $4/9$ (44%) of the area of the paper.

Gutenberg designed his Göttingen Bible of 42 lines using the ternary canon model, but the text is written in two columns whose separation space is exactly $2/3$ the width of the grid’s cells as shown in figure 3 (right); each column has $2\frac{2}{3}$ cell width covering just (39.5%) of the area of the paper.

Universal scaling Rosarivo extended this concept to $n \times n$ grids, where n is a multiple of 3. Once divided, the column and the row correspond to inner and top margins, and two columns and rows to outer and bottom margins. The margin dimensions are inversely proportional to the number of divisions made.

Note, however, that if n is not a multiple of 3 we are still following rules 1, 3 and 4. A quite critical example is taking a square page and dividing it in a 10×10 grid. The resulting type area is a 7×7 box, filling 49% of the page (see figure 10). Nice, isn’t it?

ISO 216 The ISO A range begins with the A0 sheet, whose ratio is $1 : \sqrt{2}$ and has area 1 m^2 . If we cut the A0 sheet along the middle of the long side we obtain a sheet whose ratio is $\sqrt{2}/2 : 1 = 1/\sqrt{2}$.

This new sheet, A1, is proportional to the original one and has half A0’s area, $1/2 \text{ m}^2$. This process can be continued to obtain proportional sheets with half the area of the predecessor.

Another range that belongs to the ISO 216 family starts with the B0 sheet, whose ratio is again $1 : \sqrt{2}$ and has area $\sqrt{2} \text{ m}^2$, i.e., whose width is 1m.

Name	dimensions (mm)	area (m ²)
A0	841 × 1189	1
A1	594 × 841	1/2
A2	420 × 594	1/4
A3	297 × 420	1/8
A4	210 × 297	1/16
<i>B5</i>	176 × 250	$\sqrt{2}/32$
A5	148 × 210	1/32
A6	105 × 148	1/64

Table 2: ISO 216 A range and B5 page dimensions.

It happens that the B range consists of the geometric means of the A range. For example, the height of B5 is equal to the square root of the product of the heights of A4 and A5.

Table 2 shows the dimensions of the A range up to 6 divisions, and B5, between A4 and A5 in italics. Figure 4 shows a sketch of the A range.

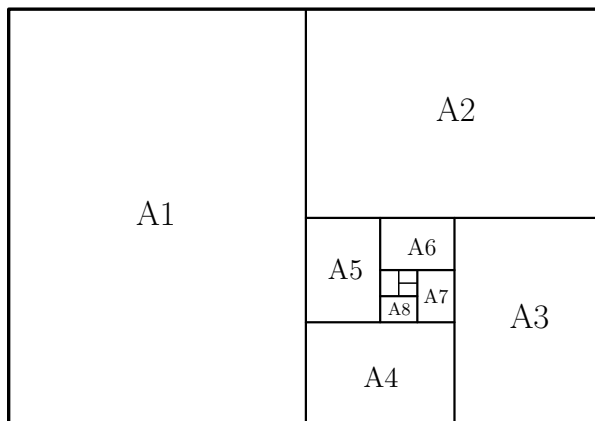


Figure 4: Subdivisions of an A0 sheet.

What is really important about this system is that it allows us two things. First, we can make a complete sheet from, say, 8 physical pages without having to make unnecessary cuts. Second, we can reduce the size of the page to its immediate successor, *retaining proportions*, using `pstops`, `psnup`, or some other tool; or by means of photographic media.

Let A_n be the page; then if $A_{(n+1)}$ is the typographic box which, as mentioned above, has 50% the area of the page (not far from the 44.4% of the ternary canon). If we follow the four rules (as shown in figure 5) the margin progression will be $2-2\sqrt{2}-4-4\sqrt{2}$ which is approximately $2-2.828-4-5.657$ which is not very far from the ternary canon's progression.

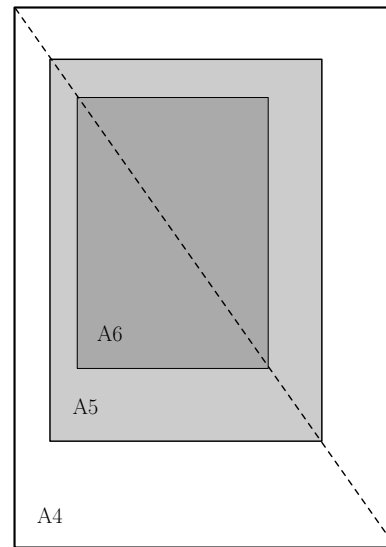


Figure 5: A5 as the typographic box of an A4 page (light gray in white), and A6 in A5 (dark in light gray).

The classics class

The `classics` class is primarily intended to typeset books using any of the designs explained in the previous section, along with some variations of them.

The user invokes `classics` like any other class:

```
\documentclass[options]{classics}
```

The description of *options* will be divided into the generic ones which are inherited from the `book` class, and the specific ones created for `classics` together with the commands (parameters) related to each page layout.

Generic options The following list shows the options inherited from `book`, which do not relate to the page design whatsoever.

`10pt|11pt|12pt` For choosing the normal type size.

The default is `10pt`.

`final|draft` Shows (`draft`) black boxes for overflow lines or not (`final`). The `final` option is the default.

`oneside|twoside` For printing on one or both sides of a page. The default is `twoside`, which produces mirrored layouts of even and odd pages. The `oneside` option makes even pages the same as odd numbered ones.

`openright|openany` For the chapters to begin only on recto pages (`openright`) or on any pages (`openany`). The default is `openright`.

`onecolumn|twocolumn` Specifies if the text box will be one or two columns per page. The default

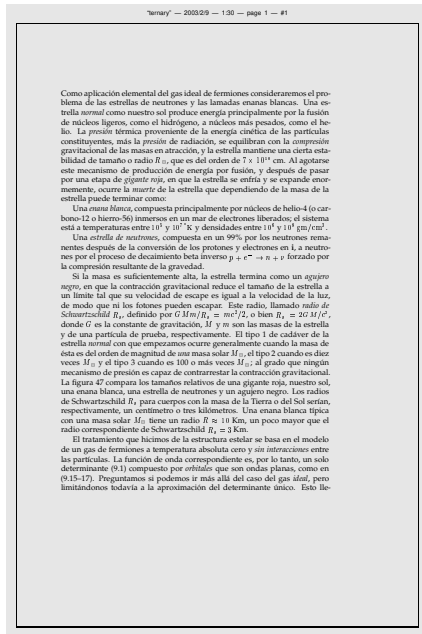
is onecolumn unless the `gutenberg` option is chosen. (See Specific options.)

`titlepage|notitlepage` If `\maketitle` is invoked, by default `classics` (`book`) will make a separate title page, and a separate abstract page, if any. `openbib` To format the bibliography in open style. `leqno` Formula numbers will be put to the left for any numbered display math environments. `fleqn` Aligns to the left displayed formulae.

As `classics` is intended specifically to produce page layouts whose dimensions are not, in general, standard, book paper size options (`letterpaper`, `legalpaper`, ...) are not recognized.

Specific options The `classics` class offers the following options for page layouts in contrast to page size options of `book`; `ternary` being the default.

`ternary` This layout builds a 2 : 3 width-to-height page with type area as described in figures 2 and 3. The only parameter users can modify is `\classicwidth` (see Parameters) because all other parameters are absolutely determined.

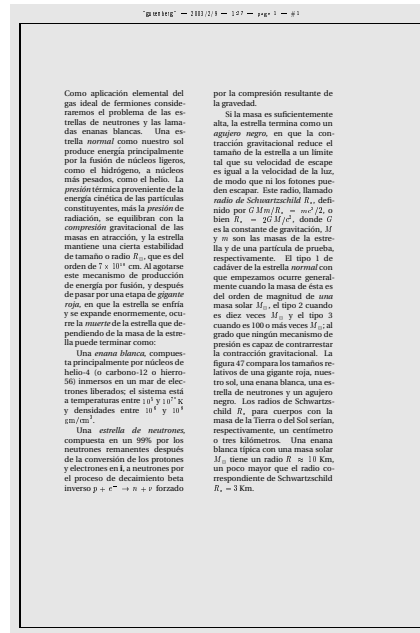


`\documentclass[palatino,frame]{classics}`

Figure 6: The default classics layout.

If the `twocolumn` option is specified, `ternary` becomes the same as `gutenberg`, but with the flexibility of changing the column separation space using `\classicscolsep`.

Figure 6 shows the `ternary` design with 42 × 63 picas width-to-height page.



`\documentclass[gutenberg,utopia,frame]{classics}`

Figure 7: Page layout for gutenberg option.

`gutenberg` Same as above but the type area is typeset in two columns having the column separation space equal to $\frac{2}{3}$ the inner margin.

Figure 7 shows the Gutenberg Bible’s layout. It is similar to `ternary` but in two columns separated by $\frac{2}{3}$ the inner margin width.

`rosarivo` Following Rosarivo’s ideas, the user can define an $n \times n$ grid of whatever page proportion are desired. By default $n = 9$, and the page proportion is 2 : 3.

The command `\cellnum` is used to change the value of n , while the width and height of the paper can be changed using `\classicwidth` and `\classicshheight` respectively.

`universal` Almost the same as `rosarivo`, but this time the user defines the cell dimensions (as `\cellwidth` and `\cellheight`) and n as above.

Note the difference: `rosarivo` divides the page once fixed to get the $n \times n$ grid, while `universal` constructs the page in terms of the cell and the grid’s dimension. Compare figures 8 and 9.

Figure 10 shows a 40 × 40 picas square page whose type area is 28 × 28 picas.

`a4|a5|b5` ISO 216 formats. These page layouts are totally predetermined, except when typesetting in two columns. If the `twocolumn` option is selected the default separation space will be equal

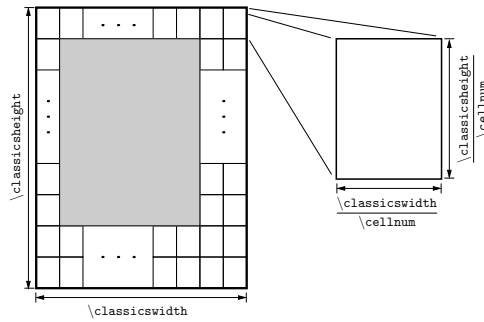


Figure 8: Page layout and relevant parameters for `rosarivo` option.

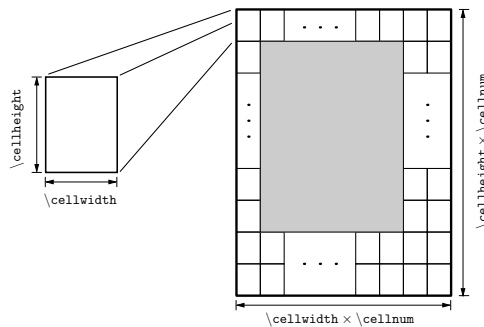


Figure 9: Page layout and relevant parameters for `universal` option.

to the inner margin, but it can be changed with the `\classicscolsep` length parameter.

Figure 11 is the A6 type area of the A5 sheet following the rules described in Classic page layouts.

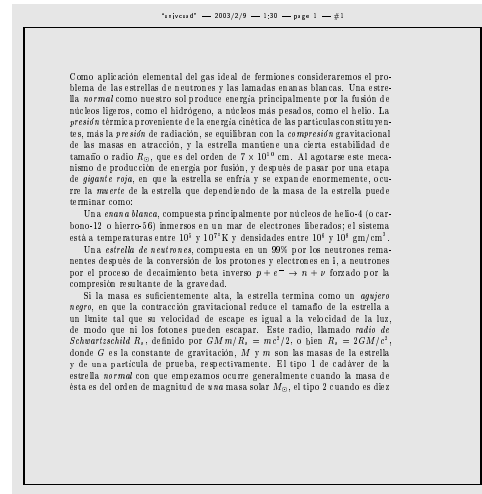
2346 The user can define the page proportions and inner margin with the `\classicsinmargin` parameter. The other margins are determined by multiplying the inner margin by 1.5, 2 and 3.

Figure 12 shows quite an extreme case; it defines a square page with 2–3–4–6 progression margins creating a wider than higher type box.

`ddiagonal` This is almost equivalent to Rosarivo's method. The user simply sets the page proportions and inner margin. It is precisely our first approach, shown in figure 1.

The main difference from `rosarivo` is that the top inner corner should not divide horizontally (vertically) the width (height) of the page.

`diagonal` Most free of all. The user can set the page proportions as well as inner and outer margin lengths. The only thing that this option preserves is similarity between the type area and the page.



```
\documentclass[universal,frame]{classics}
\cellwidth4pc
\cellheight4pc
\cellnum10
...
```

Figure 10: A square page with type area determined using `universal` option.

Cropmarks The following options define the kind of cropmarks to be printed. If any, information about the document will be printed on top of the page. It includes the file name, date, `TeX` page number, i.e., in case of `\frontmatter` material, and page index number, which starts with `#1` and is consecutively incremented.

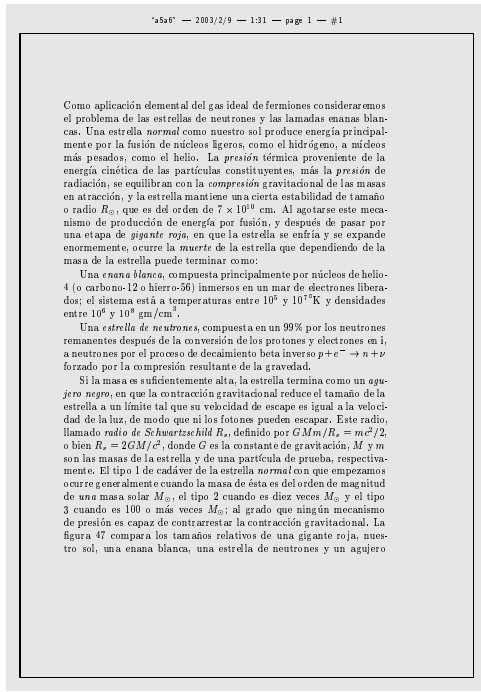
`camera|frame|nocrop` The default is `camera`. If the user is playing with the parameters, `frame` can give a better idea of the layout. It is useful to have no crop-marks (`nocrop`) if printing in a physical page whose dimensions are equal to the page defined.

Predefined fonts Finally, some shortcuts for loading *Palatino*, *Times*, and *Utopia* PostScript fonts:

`cm|palatino|times|utopia` The default option is `cm`, but believe me, with `palatino`, the ternary canon looks great; and for Gutenberg's Bible style, the compactness of `utopia` is ideal for the narrowness of the columns.

Parameters The following parameters are used by all page layout options except for `universal`, whose paper dimensions are determined by `\cellwidth`, `\cellheight`, and the grid constructed in terms of `\cellnum`.

`\classicswidth{dimen}` Sets the width of the paper. By default it is set to 42 pc, giving 42 lines



```
\documentclass[a5,frame]{classics}
```

Figure 11: A5 (148 × 210 mm) page with A6 type area obtained using a5 option.

of text in both `ternary` and `gutenberg`, if the font size is set to 10pt.

`\classicsheight<dimen>` Sets the height of the paper. If height is in terms of `\classicswidth`, i.e., if page layout `ternary` or `gutenberg` is selected, `\classicsheight` is ignored.

The following parameters are useful only for the `diagonal` and `ddiagonal` options:

`\classicsinmargin<dimen>` Sets the length of the inner margin. For the `ddiagonal` design, the outer margin is determined by the inner margin.

`\classicsoutmargin<dimen>` Sets the length of the outer margin. This parameter is only used by the `diagonal` page layout.

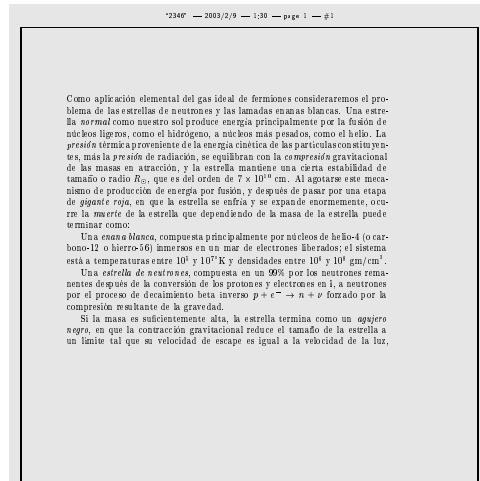
For the `universal` page layout option:

`\cellwidth<dimen>` Sets the width of the cell (see figure 9).

`\cellheight<dimen>` Sets the height of the cell.

The next parameter is needed by `rosarivo` (figure 8) and `universal`:

`\cellnum<count>` If `rosarivo` is chosen, then both `\classicswidth` and `\classicsheight` will be divided by `\cellnum` to get the grid.



```
\documentclass[2346,frame]{classics}
\classicswidth40pc
\classicsheight40pc
\classicsinmargin4pc
```

Figure 12: A 40 × 40 picas page with 2–3–4–6 margins progression and 4 picas inner margin.

If `universal` is chosen, then the page dimensions are obtained by multiplying `\cellnum` to `\cellwidth` and `\cellheight`.

Finally, for `twocolumn` or `gutenberg` options the separation space between columns is set by:

`\classicscolsep<dimen>` This command is analogous to `\columnsep` of the base classes in L^AT_EX.

The defaults If `classics` is invoked without options, i.e.,

```
\documentclass{classics}
```

then the option `ternary` will be the page layout, 10pt the font size, `cm` the font family, `camera` cropmarks, and all book inherited options as described in the ‘Generic options’ section.

Parameters are initialized in such a way that whatever page layout is selected, it will look like the `ternary` canon:

```
\classicswidth=42pc
\classicsheight=63pc
\classicswidth = 2/3\classicsheight
\classicsinmargin=56pt
\classicsoutmargin=112pt
\classicsinmargin = 1/9\classicswidth
                  = 42pc / 9 = 42 × 12pt / 9 = 56pt
\classicsoutmargin = 2\classicsinmargin
```

```

\cellwidth=56pt
\cellheight=84pt
\cellnum=9

\cellwidth = \classicsinmargin
\cellheight = 3/2\cellwidth
\cellnum × \cellwidth = \classicswidth
\cellnum × \cellheight = \classicsheight

\classicscolsep=37.33pt

\classicscolsep = 1/9(type area)
                = 2/3\classicswidth
                  9

```

No matter what page layout option is chosen, except for the ISO 216 system, if parameters are not modified, the layout will be the ternary canon, or Gutenberg's Bible if `twocolumn`.

Moral of the story

*A wider outer margin is quite useful not only to be able to hold the book comfortably, but also to be able to make notes or annotations, and not just for controversial texts.*¹

¹ Hermann Zapf, *TUGboat*, Volume 22 (2001), No. 1/2. (However, in the XVIIth century, there was not enough margin space for Fermat to write the beautiful proof of his last theorem.)

References

- [1] Jorge de Buen Unna. *Manual de diseño editorial*. Santillana, México, D.F., 2000.
- [2] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The L^AT_EX Companion*. Addison-Wesley, Reading, MA, USA, 1994.
- [3] Donald Ervin Knuth. *The T_EXbook*, volume A of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986.
- [4] Leslie Lamport. *L^AT_EX: A Document Preparation System: User's Guide and Reference Manual*. Addison-Wesley, Reading, MA, USA, second edition, 1994. Reprinted with corrections in 1996.
- [5] Douglas Martin. *An Outline of Book Design*. Blueprint, London, UK, 1989.
- [6] Raúl Mario Rosarivo. *Divina proporción tipográfica ternaria*. Buenos Aires, Argentina, 1948.
- [7] Han Thê Thành. *Micro-typographic extensions to the T_EX typesetting system*. PhD thesis, Masaryk University Brno, 2000.
- [8] Jan Tschichold. *Ausgewählte Aufsätze über Fragen der Gestalt des Buches*. Birkhäuser, Basel, Switzerland, 1987. Second printing.
- [9] Adolf Wild. La typographie de la Bible de Gutenberg. *Cahiers GUTenberg*, 22:5–16, septembre 1995.
- [10] Roberto Zavala Ruiz. *El libro y sus orillas*. UNAM, México, D.F., third edition, 1995.

Digital illumination

Dr. Alun Moon

School of Informatics

University of Northumbria

Newcastle upon Tyne, UK

alun.moon@unn.ac.uk

Abstract

Donald Knuth has given us *Digital Typography*, and through METAFONT, *Digital Calligraphy*; this paper explores how these tools can be used for *Digital Illumination*. It follows from my interest as an amateur calligrapher in Celtic artwork. Two examples of my work are in figures 1 and 2. Compare with a sketch of an element from the Lindesfarne Gospels in 3 (Bain, 1989, pg. 67), which is the destination I'm reaching for. This has been a very good exercise in learning to write macros for METAPOST.

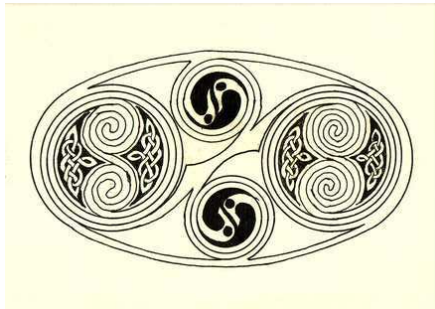


Figure 1: A cartouche with spiral inserts.

Some background

Celtic artwork in Britain covers the period from the 7th century BC through the 7th century AD. During that time examples can be found in stone carving, intricate metalwork, and, towards the end of the period, in illuminated manuscripts. The Celtic monastic scribes produced such masterpieces as *The Book of Kells* and *The Lindesfarne Gospels*. Lindesfarne itself is about 60 miles north of Newcastle; the Gospels are thought to have been produced at Jarrow on the south bank of the Tyne. These show a highly developed artistic style, with very fine, intricate detail. There are three main styles considered here: knots, keypatterns, and spirals.

Knots and keypatterns can be drawn from block elements treated as characters, and large carpet pages built from these standard elements. However, the Celtic scribes show a high degree of geometry and geometrical construction in their work.

A knot can be described as one or more strands that loop, cross and re-cross many times. Can the

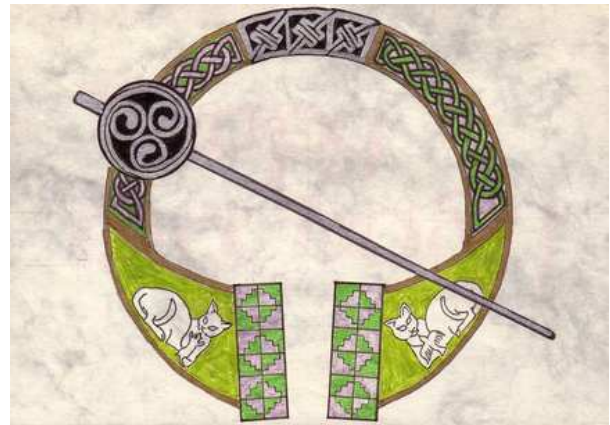




Figure 2: A brooch inspired by elements of the Tara brooch.

curves be described and then a METAFONT algorithm used to split them up to generate the under-over-under-... pattern?

A keypattern does have a base form that is then tiled to form the page. The base pattern does have a simple sequence of numbers that define it. A sequence such as (1, 1, 2, 2, 7, 2, 2, 1, 1) gives a pattern such as . This pattern can then be tiled .

Can these simple sequences be used to program METAFONT to generate larger patterns?

Similarly, spiral patterns can be constructed using a pair of compasses. How can METAFONT's geometrical programming be used as a digital pair of compasses to create these beautiful patterns?

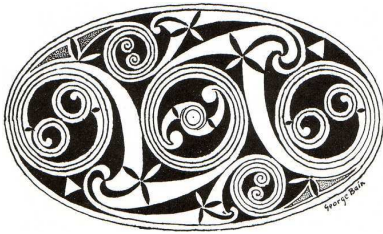


Figure 3: Sketch of a cartouche in the Lindesfarne Gospels (2 in long).

Knotwork

Interlacing patterns of weaving cords is possibly the best known and most widely recognised form of Celtic artwork.

Given a set of curves, once the intersections are known and sorted along the paths, drawing the intersections is easy. For an array of paths $p[]$, two numeric arrays are needed. One holds the times of the intersections $p[]t[]$, the other a count of intersections $p[]t\#$. This way p_1t_3 is the 3rd intersection on the 1st curve, and $p_3t\#$ is the number of intersections on the 3rd curve. The intersections can be found with the `intersectiontimes` operator in METAPOST and METAFONT.

A function `crossings` takes a suffix parameter and a text parameter. The suffix is the path for which the crossings are to be found, and the text is a list of paths to test for (see figure 4).

Intersection times The `intersectiontimes` operator tends to generate points at the beginning of the path. To iterate along the path a series of subpaths are used. Each one starts just past the last intersection (time plus `epsilon`), up to the end of the path.

There is a small problem using subpaths with the `intersectiontimes` operator; the time returned is the time for the subpath. The path $z_1..z_2..z_3..z_4$ has length 4, while the subpath $[.75, 1.25]$ has length 2. Points have been added to the beginning and end where there are not points of the original path. The intersection time on the subpath (t_s) can be converted to a time on the full path as follows:

- if $t_s < 1$, use t_s to interpolate between the beginning of the subpath (a) and the next point on the curve (ceiling of a).
- if $t_s \geq 1$, add it to the last point on the curve before the subpath (floor a).

For a simple knot the global intersection-times for one of the paths is shown in figure 5.

```

vardef crossings@#(text others) =
  save lastpt, tmp;
  p@#t[0]:=0;
  p@#t#:=0;
  forsuffices $=others:
    numeric lastpt;
    lastpt := epsilon;
    forever:
      numeric tmp;
      (tmp,whatever)=
        subpath (lastpt,length(p@#)-epsilon)
          of p@#
      intersectiontimes p$;
      exitif (tmp<=0);
      p@#t[incr p@#t#] := if(tmp<1):
        tmp[lastpt,ceil(lastpt)]
      else:
        floor(lastpt)+tmp
      fi;
      lastpt := p@#t[p@#t#]+epsilon;
    endfor
  endfor;
  sort.p@#t;
enddef;

```

Figure 4: `crossings` function.

Drawing the crossings The crossings are drawn using the `erase draw` technique, as described in *The METAFONTbook* (pg. 113). The erasing segment is drawn between the midpoints of the sections on either side of the crossing point, the line is then drawn slightly longer. This avoids gaps in the lines where the erasing began.

Examples With the crossing macros, any knotwork pattern can be drawn, as long as there are the paths $p[]$ defined. There is one caveat, each path *must* start so that its first crossing is over the path it crosses. This makes all the crossing times the odd numbers in the array of times.

The trefoil The trefoil is a simple knot using four paths (figure 6). Some people claim it symbolises the Holy Trinity, or wholeness (I like it because it is the motif used for my wedding).

Border A common theme is a knotwork border, a simple example is shown in 7 after (Bain, 1989, pg. 29). Once the paths are specified, application of the `crossings` and `drawcrossings` macros generate the knots.

Better knots Because a circular pen is used for both erasing and drawing the lines, there is a limit to how wide the line can be before the ends of the strokes become visible.

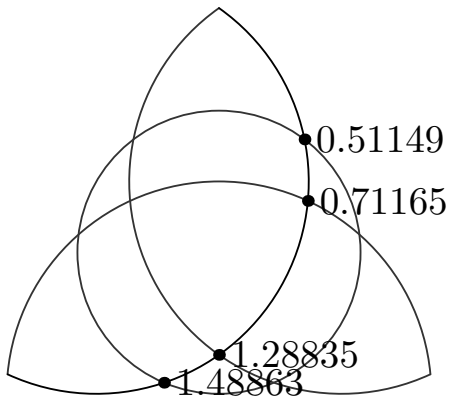


Figure 5: Times for intersection points.

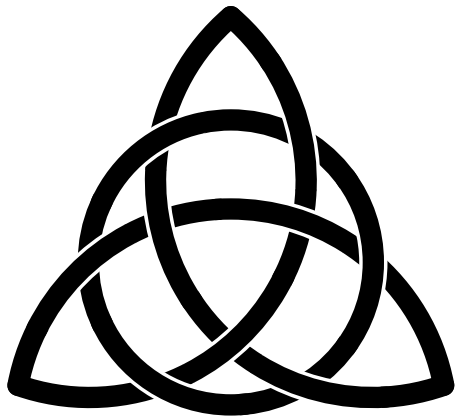


Figure 6: The trefoil.

A better method would be to generate the points that form the end of each stroke. This can be combined with the `penpos` and `penstroke` macros. This requires a little more mathematics. Once the intersection is known, a time on the path is needed to give a point a given distance from the intersection.

Keypatterns

Keypatterns are a common border or filling element. Usually the base element is a C or S spiral, which is then repeated to fill the space. The edges use a separate pattern that fills in around the basic shape.



Figure 7: A knotwork border.

Bain (1989) uses a numeric notation to describe the core patterns, the inspiration for one approach.

S-spiral generation A sequence of numbers such as (1, 2, 3, 4, 9, 4, 3, 2, 1) defines a curve; each number is the length of the segment. Each new segment is drawn at right-angles to the last. Turning anti-clockwise in the first half, where the lengths are increasing. Clockwise in the second, where the lengths decrease.

The macro is shown in figure 8. It maintains a copy of the maximum length drawn, to test whether to turn clockwise or anti-clockwise.

```
def keySspiral(text tail) :=
  begingroup
  save direct,lastpoint,maxlength;
  pair direct,lastpoint;
  direct := up rotated -90;
  lastpoint := origin;
  maxlength := 0;
  origin
  for p=tail: --
    begingroup
    direct := direct
      rotated if (maxlength<=p):
        begingroup maxlength := p;
        90 endgroup
      else:
        -90
    fi;
    lastpoint := lastpoint + direct*p;
    lastlength := p;
    lastpoint
  endgroup
endfor
endgroup
enddef;
```

Figure 8: S-spiral generator macro.

Tessellated curves The sequence (1, 2, 3, 4, 8, 4, 3, 2, 1) gives a curve that tessellates to fill a region (figure 9). Typically a blank border surrounds the keypattern; the `clip` operator in METAFONT serves well to form the border.

Programmed variation A curve that can tessellate in such a way that it interlocks with itself (figure 10) can be generated by the sequence (1, 2, 3, 4, 9, 4, 3, 2, 1).

With METAFONT we have a powerful programming tool that can vary the pattern as it is drawn, in ways that the seventh century scribes could not easily do. Figure 12 shows the border from figure 10 varying in intensity; figure 11 shows the width of the line varying.

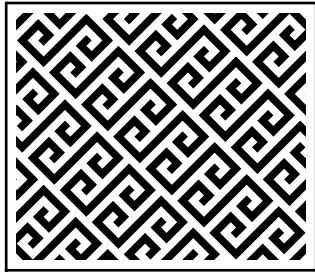


Figure 9: Space-filling keypattern.



Figure 10: Interlocking keypattern.

Figure 12 is also interesting as the scale of the lengths (and pen width) is 1 pt. It looks fine on a monitor, but may test the limits of a printer. METAFONT allows even finer detail, limited only by the resolution of the printer.

Spirals

Spirals are another signature element of Celtic artwork. Meehan (1993b) shows how the spiral elements can be drawn using two, three or four offset centres and a pair of compasses. METAFONT draws three point paths ($z_0..z_1..z_2$) as close to a circular arc as possible (Knuth, 2000, pg. 128).

Given an initial point, a pair of centres, and a number of turns, the `spiral` macro is a very simple recursive function (figure 13). Although it could be just as simple with a loop, swapping the centres over is easier to do with the recursive call.

Figure 14 shows a cartouche inspired by figure 3. Each pair of spirals is joined by a path connecting the outer points. This path really should follow a common tangent to the two curves forming the outer end of the spiral. A macro is needed to



Figure 11: Keypattern varying with line width.



Figure 12: Keypattern varying with colour.

```
def spiral(expr a,b,$)(expr turns) =
  $
  .. $ rotatedaround(a, 90)
  .. $ rotatedaround(a, 180)
  if( turns>1 ):
    & spiral(b,a,
              $ rotatedaround(a, 180))
              (turns-1)
  fi
enddef;
```

Figure 13: Spiral macro.

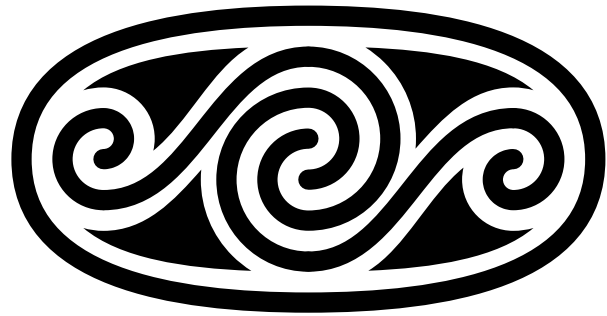


Figure 14: Cartouche inspired by figure 3.

find the points on the two curves giving the common tangent.

Some resources

Over the years I have found several books to be of use. George Bain (Bain, 1989) is often cited as the key work. He has collected a wide range of material from the Gospels, Book of Kells, jewelry, artifacts and stone carvings. It doesn't go into many of the construction techniques, but is a very good source of inspiration. Aidan Meehan has produced a series of books (Meehan, 1993a; Meehan, 1993b), which give a step-by-step approach and give some good ideas as to the construction of the geometry. Sheila Sturrock's book (Sturrock, 2001) has a different approach to building the keypatterns and clearly shows how the borders develop.

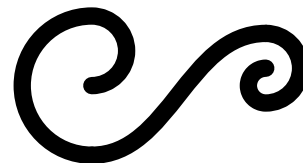


Figure 15: Joined spirals.

Andy Sloss has two books with a radically different technique (Sloss, 1997a; Sloss, 1997b). He enumerates all possible combinations of crossings, characterised by the four entry and exit directions of the two strands. These can then be laid out on a grid. This would be eminently suitable for conversion to a font (assuming enough characters). Perhaps a task for the long winter evenings.

Finally

What new Illumination can be produced by a tool as highly versatile as METAFONT? Can the transformations in Metafont implement conformal mapping and be applied to patterns generated as above? Preliminary experiments suggest it can, but the trick is finding the mapping to use. If the patterns can be described in a parametric form, can an Escher-like tiling be achieved where the pattern changes across the page? Again yes, but finding a shape that scales and still fits together is difficult.

The so-called “Dark Ages” produced a flowering of the work of Celtic scribes, culminating in the “Golden Age” of the Scribes’ art. Knuth has given us tools to usher in a Golden Age of Typesetting and Digital Illumination.

References

- Bain, George. *Celtic Art: The Methods of Construction*. Constable and Company Ltd, 1989. ISBN 0 09 461830 5.
- Knuth, Donald. *The METAFONTbook*. Addison Wesley, 2000.
- Meehan, Aidan. *A Beginner’S manual*. Celtic Design. Thames and Hudson, 1993a. ISBN 0 500 27629 3.
- Meehan, Aidan. *Spiral Patterns*. Celtic Design. Thames and Hudson, 1993b. ISBN 0 500 27705 2.
- Sloss, Andy. *How to Draw Celtic Key Patterns*. Blandford, 1997a. ISBN 0 7137 2652 0.
- Sloss, Andy. *How to Draw Celtic Knotwork*. Blandford, 1997b. ISBN 0 7137 2492 7.
- Sturrock, Sheila. *Celtic Spirals and Other Designs*. Guild of Master Craftsman Publications Ltd, 2001. ISBN 1 86108 159 6.



Scientific publishing with pdfL^AT_EX

Patrick W. Daly

Max-Planck-Institut für Aeronomie

daly@linmpi.mpg.de

Arne K. Richter

Copernicus Society

akrichter@copernicus.org

In many areas of science, 20–60% and even more authors are applying L^AT_EX when submitting their manuscripts for publication in scientific magazines, journals or book series. We therefore suggest that scientific publishers should apply L^AT_EX with pdfT_EX for

1. composing, editing and formatting articles in any required document class and lay-out;
2. making use of the easy conversion to PDF for mailing, proof-reading or uploading purposes;
3. taking advantage of PDF for e-publication on the web and CD, as well as for printing on paper by the print-on-demand or the computer-to-plate technologies of digital printing.

All these advantages of L^AT_EX+pdfT_EX, from manuscript submission to final production of a journal or a book, are demonstrated here with examples from the scientific publications produced by the non-profit publishing house “Copernicus”.

Typesetting nightmares

Hans Hagen

PRAGMA ADE

It does not take much for users (and customers) to realize that T_EX is a programming language. This often results in the perception that you can do anything you want, and make people believe that you can do better than other, less open applications. Combine this with the fact that developers seldom admit that something cannot be done, and the ingredients of a typographic programming nightmare are present.

The complication arises from the facts that:

- opposite to desktop publishing applications, T_EX sees a document as a sequence of content
- where T_EX-based macro packages tend to organize fonts and measures, designers follow a more random path
- where T_EX loves structure, authors want to put any thought on paper, being structured or not, which results in not only interfering data, but also in the wish to escape from T_EX’s machinery
- one reason for choosing T_EX is its ability to typeset math, and typesetting that often conflicts with pure text typesetting
- T_EX tries to do its best to typeset beautiful paragraphs, but frequently a (not producible by T_EX) alternative is considered more beautiful or adequate

This means that in order to fulfill the needs of authors and designers, one sometimes has to bend T_EX’s rules and cook up rather complicated macros. In this presentation I will discuss a couple of last year’s (typo-)graphical programming nightmares.

L^AT_EX in real-world math typesetting: NFSS vs. NFNF

Mirko Janc
INFORMS
mirko.janc@informatics.org

The New Font Selection Scheme (NFSS) in L^AT_EX enabled much more flexible managing of fonts. Handling of math fonts and text fragments within math was also enhanced. However, the available choice of math fonts is too limited to satisfy standard requirements of many major scientific publishers to set math in virtually arbitrary fonts.

To meet such requirements for typesetting a number of major journals and a considerable number of books that are heavy in math (from the point of view of typesetting), we developed a strategy of attacking the problem from two sides at the same time. A new font-loading scheme, based on NFSS, is employed in conjunction with custom-adapted math fonts. Within math, the main focus is on fonts intended to play the role of the `cmmi*` fonts from the Computer Modern family (italic variables plus Greek). Extensive work on kerning and spacing, as well as enriching the math font families available, was the key to allow complex math formulas to be typeset easily in various font styles and to look good.

An additional twist comes from the common requirement for special styles in section titles, table and figure captions, table body, abstract, etc. A typical example is to require the abstract to be all bold, section titles to be bold sans-serif, whereas table and figure captions should be sans-serif bold-condensed. All of those elements can contain formulas. Setting math in such circumstances can be tricky. We show that our font scheme, combined with a satisfactory solution to NFNF (**N**eed **F**or **N**ew **F**onts) can successfully accomplish the tasks.

A number of samples illustrate this font system at work. We show a number of other enhancements within this system, both on the font-loading side and in the fonts themselves. Some other real-life situations in math typesetting that tend to be neglected in academic discussions are also presented.

Using T_EX to manage IT for a mathematics congress

Ross Moore
Macquarie University
ross@maths.mq.edu.au

Just prior to this TUG conference, the ICIAM 2003 congress was held. This was a major international congress attracting ≈ 2000 delegates from all branches of mathematics—mainly applied, but a good deal of pure mathematics was also represented.

As the author was responsible for the information technology aspects of this congress, T_EX-related software served a major role throughout. It was used, for example, with

- design and production of the congress web-site;
- publicity pages for the invited speakers;
- the abstract-submission process;
- online display of abstracts, in HTML and PDF;
- printed book of abstracts, program and speaker/subject listings;
- conference volume, with full papers from the invited speakers;
- slides of abstracts for all talks and minisymposia;
- conference CD, with the above items in both HTML and fully-linked/bookmarked PDF.

In this talk we will look at some of this material and discuss the T_EXniques used in its production and maintenance over more than 2 years leading up to the congress itself. This will include aspects relevant to L^AT_EX, pdfT_EX, L^AT_EX2HTML, MathML, and CGI scripting for web pages.

BIB_TE_X yesterday, today, and tomorrow

Oren Patashnik
San Diego, CA

Abstract

This paper looks back at the last 20 years of BIB_TE_X, and also looks ahead, if not to the next 20 years, then at least to BIB_TE_X 1.0.

Introduction

BIB_TE_X is the bibliography program designed originally to accompany Leslie Lamport's L^AT_EX; it now works with other incarnations of T_EX, too. BIB_TE_X removes the tedium, and adds some flexibility, in producing a reference list.¹ When BIB_TE_X creates your reference list, it's BIB_TE_X, not you, minding the minutiae like ensuring that your reference-list entries are in the correct order, that every comma is in place, and that the information is formatted consistently across entries. Furthermore, a single, simple, change of bibliography-style name lets you convert your reference list from style A, which might order the entries alphabetically, and spell out journal names in full, and list all authors as first-name then last-name, to a completely different style B, which might order the entries according to their order of mention in the text, and abbreviate journal names, and invert just the first author's first and last names.

This paper is an updated version of a paper [7] from the 1994 TUG meeting in Santa Barbara. It gives: a history of BIB_TE_X to the present; the general goals for BIB_TE_X 1.0, which will be the frozen version of BIB_TE_X (just as T_EX 3 is the frozen version of T_EX); some specific new features for achieving those goals; and the plan for releases of BIB_TE_X leading up to BIB_TE_X 1.0. Before all that, however, primarily for those who are unfamiliar with BIB_TE_X, comes a section that explains its rudiments.

Using BIB_TE_X

To use BIB_TE_X, you'll have your bibliographic information in a bibliography database, and, to make use of that information, a few L^AT_EX² commands sprinkled throughout your L^AT_EX source file.

¹ Throughout this paper, the term 'reference list' is used generally to refer to what might also be called a 'bibliography' or a 'list of sources' or anything similar.

² The term 'L^AT_EX' is used to mean either L^AT_EX or plain (or other variations of) T_EX.

For example in a file `mybib.bib` (database file names end with `.bib`) you might have an entry like:

```
@BOOK{knuth:tex,  
  author = "Donald E. Knuth",  
  title = "The {\TeX}book",  
  publisher = "Addison-Wesley",  
  year = "1984",  
}
```

The `@BOOK` tells BIB_TE_X the entry type. (The bibliography style will instruct BIB_TE_X on how to format a `BOOK` entry type.) The `knuth:tex` is the database key, which is a sequence of characters to be used as the symbolic name for this entry. And the rest of the entry comprises four `<field> = <field-value>` pairs appropriate for a `BOOK` entry type. In general you will have many such entries in a database file; you might also have multiple database files.

And in your L^AT_EX source file you might have a citation like

```
... in the \TeX{ }book~\cite{knuth:tex} ...
```

The `\cite` command's argument `knuth:tex`, called a cite key, must match the corresponding database key. L^AT_EX might typeset this `\cite` command as

```
... in the TEXbook [41] ...
```

or

```
... in the TEXbook41 ...
```

or

```
... in the TEXbook (Knuth, 1984) ...
```

depending on the citation style. L^AT_EX's default citation style uses a number in brackets, and for that citation style, together with an appropriate bibliography style, the corresponding reference-list entry might look like:

41. Donald E. Knuth. *The T_EXbook*. Addison-Wesley, 1984.

Besides the `\cite` commands, your L^AT_EX source file will also have two BIB_TE_X-related commands:

```
\bibliography{mybib}  
\bibliographystyle{plain}
```

The `\bibliography` command does two things: It tells \LaTeX to put the reference list at that spot in your document, and it tells $\text{BIB}\TeX$ which file(s) to use for the bibliographic database, here just the single file `mybib.bib`. The `\bibliographystyle` command tells \LaTeX nothing, but tells $\text{BIB}\TeX$ which bibliography style to use, here the standard style `plain`; bibliography style file names end with `.bst`, thus the relevant file is `plain.bst` in this case.

So with your database file(s) and your \LaTeX source file structured appropriately, your citations are formatted according to the citation style, and your reference list is formatted according to the bibliography style.

To actually produce the typeset document, you run \LaTeX , $\text{BIB}\TeX$, \LaTeX , \LaTeX . The first \LaTeX run writes, to an `.aux` file, information for use by $\text{BIB}\TeX$ — which bibliography style to use, which database file(s) to use, and which database entries to include. The $\text{BIB}\TeX$ run reads all that information from the `.aux` file, reads the specified database (`.bib`) file(s), formats the reference list according to the instructions in the bibliography style (`.bst`) file, and writes its output onto a `.bbl` file. The next \LaTeX run reads the `.bbl` file and incorporates the reference list into the document. The final \LaTeX run fixes the references into the reference list. Figure 1 shows the files that $\text{BIB}\TeX$ uses. The `.blg` file is $\text{BIB}\TeX$'s log file, in which $\text{BIB}\TeX$ records any warning or error messages.

To try using $\text{BIB}\TeX$ with \LaTeX , put the six-line `BOOK` entry shown on the previous page into a file called `mybib.bib`, and then, into a file called `mypaper1.tex`, put these six lines of \LaTeX :

```
\documentclass{article}
\begin{document}
The \TeX{}book~\cite{knuth:tex} is good.
\bibliography{mybib}
\bibliographystyle{plain}
\end{document}
```

Exactly how you run \LaTeX and $\text{BIB}\TeX$ is system dependent; on my system I issue four commands:

```
latex mypaper1
bibtex mypaper1
latex mypaper1
latex mypaper1
```

To try using $\text{BIB}\TeX$ with plain \TeX , create the file `mybib.bib` as above, and then put into a file called `mypaper2.tex` these seven lines of plain \TeX :

```
\input btxmac
The \TeX{}book~\cite{knuth:tex} is good.
\medskip
\leftline{\bf References}
\bibliography{mybib}
```

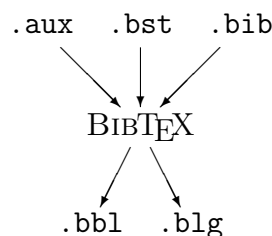


Figure 1: $\text{BIB}\TeX$'s input and output files.

```
\bibliographystyle{plain}
\bye
```

To run `mypaper2` through \TeX and $\text{BIB}\TeX$:

```
tex mypaper2
bibtex mypaper2
tex mypaper2
tex mypaper2
```

The file `btxmac.tex`, which `mypaper2.tex` `\input`s, contains the macros that make $\text{BIB}\TeX$ work with plain \TeX . Those macros are a standard part of most \TeX distributions, but if they're not a part of yours, you'll have to go fetch a copy.

That's a brief introduction to $\text{BIB}\TeX$. The following sources provide further details. Leslie Lamport's \LaTeX manual [4] explains how to use $\text{BIB}\TeX$ with \LaTeX . In particular, section B.1 describes the `.bib`-file format in detail. The file `btxmac.tex` [1] documents its own use, with or without Karl Berry's `eplain.tex` package (for which the `btxmac` macros were originally written). The “ $\text{BIB}\TeX$ ing” document [5], which is distributed along with $\text{BIB}\TeX$ itself, contains further hints for $\text{BIB}\TeX$ users. The “Designing $\text{BIB}\TeX$ Styles” document [6], also distributed with $\text{BIB}\TeX$, explains the postfix stack-based language used to write $\text{BIB}\TeX$ bibliography styles (`.bst`) files. *The \LaTeX Companion* (2nd edition) [2], by Michel Goossens, Frank Mittelbach, et al., and *Guide to \LaTeX* [3], by Helmut Kopka and Patrick W. Daly, summarize much of the information in the sources above, and describe some of the tools available for helping with $\text{BIB}\TeX$. $\text{BIB}\TeX$'s standard bibliography styles, like `plain`, are based on Mary-Claire van Leunen's *A Handbook for Scholars* [9]. That book is worthwhile reading for anyone wanting to design a bibliography style from scratch.

History

Leslie Lamport patterned \LaTeX after a document production system called `Scribe` [8], written by Brian Reid in the late 1970s at Carnegie-Mellon University. One of `Scribe`'s basic tenets was that, to the extent possible with a computer program, writers should be allowed to concentrate on content rather

than on formatting details. Or, as Reid so amusingly put it:³

Not everyone should be a typesetter.

I think of L^AT_EX as a fairly successful Scribification of T_EX — L^AT_EX is almost as easy to use as Scribe yet almost as powerful as T_EX.

In any case, Scribe had become popular in certain academic circles, and Lamport decided that, to make it easy for Scribe users to convert to L^AT_EX, he would adopt Scribe’s bibliography scheme in L^AT_EX. But T_EX macros alone were insufficient in practice to do all the things, like alphabetizing, that a bibliography processor needs to do; he decided instead to have a separate bibliography program. That program would manipulate the bibliographic information in Scribe-like database files according to the instructions programmed in a special-purpose style language. The postfix stack-based language he had in mind was to be powerful enough to program many different bibliography styles.

My own work on BIBTEX started in February 1983. Just for the fun of it, I went back and dug up the original email:

```
15-Feb-83 0908
To: OP@SU-AI, tex82@SRI-CSL
Oren,
Leslie Lamport is working on a new
macro package for TeX82, and needs
someone to write a support program
or two. We are volunteering you. His
address is TEX82@SRI-CSL. Why don't
you get in contact with him, and see
whether his requirements are within
the scope of what you're willing to do.
```

This was described as a “three-week project”, and the first time I gave this talk, at Santa Barbara in 1994, I compared the “three-week project” to the “three-hour tour” of the 1960’s American television series *Gilligan’s Island*, in which an afternoon’s harbor cruise became a shipwreck adventure lasting years. (Many of us at that TUG conference really did go on a three-hour harbor cruise.)

This year’s meeting was on the Big Island of Hawaii, though, and it struck me that there’s an even better metaphor for my work on BIBTEX than the Gilligan’s Island adventure: I started working on BIBTEX, it turns out, within days of when the current eruptive phase of Kilauea began, in early 1983. And a volcano aptly describes my work on BIBTEX — a burst of activity, followed by a dormant stretch. (2004 will be more active than dormant.)

³ — to a Bell Labs Computer Science Colloquium audience that included some troff true believers

But back to 1983. Over the course of the next year and a half I implemented Lamport’s basic design, with a few enhancements. The first working version of BIBTEX (0.41) trudged forth in the summer of 1984. Lamport wrote, and Howard Trickey modified, a bibliography style based on Mary-Claire van Leunen’s suggestions in her *Handbook for Scholars* [9]. Trickey’s modified version was to become `btxbst.doc`, the template for BIBTEX’s four standard styles `plain`, `abbrv`, `alpha`, and `unsrt`. (By the way, these are called “standard” styles not because they are supposed to be some sort of standard, but because they are in the standard release of BIBTEX.)

The first public release of BIBTEX, in March 1985, was version 0.98, for L^AT_EX version 2.08. Several upgrades, including one for L^AT_EX 2.09, followed later that year. Version 0.99, which added many new features, was released in January 1988; two minor upgrades followed the next month, but BIBTEX itself has remained unchanged since then. The standard styles have been unchanged since March 1988.

In 1990 Karl Berry wrote some macros, for use in his `eplain.tex` package, that made BIBTEX usable with `plain` (and other versions of) T_EX. He and I modified the macros and released them as `btxmac.tex` in August 1990, usable with or without the `eplain` package. Several upgrades followed, the last in March 1995.

The current versions are: 0.99c for BIBTEX itself; 0.99b for `btxbst.doc` (the standard styles’ template file — but version 0.99a for each of the four standard styles); and 0.99k for `btxmac.tex`.

That’s for the software I’ve worked on. There has, in addition, been work that others have done to support BIBTEX, especially in the last decade:

- software for the handling of database (`.bib`) files (my personal database-handling preference is the text editor Emacs’s BIBTEX mode, but I don’t recommend that others join the Emacs religion just for that);
- the amassing of many database files for public consumption, minimizing the amount of work it takes to create new entries (Nelson Beebe has been particularly prolific in that regard);
- software for generating new bibliography styles automatically, without having to know details of the somewhat arcane `.bst` bibliography-style language (Patrick Daly’s `custom-bib/makebst` package has gotten widespread use — more on this later).

That brings us up to date.

Goals

BIB_TE_X has been very stable for some time now. Software stability is nice; it helps others build tools that augment the software. As suggested above, many tools have grown up around BIB_TE_X. The popularity of (L^A)_TE_X, however, has taken BIB_TE_X into places not anticipated, necessitating changes. Here are some general goals for BIB_TE_X 1.0:

- Easier nonstandard styles: The most frequent requests I see are for new bibliography styles. Creating a new bibliography style generally entails programming in the `.bst` language, which, for most users, is not an easy task. Patrick Daly's `custom-bib/makebst` package has made it reasonably easy for ordinary user to create new bibliography styles; for BIB_TE_X 1.0 the situation will improve even more.
- More international: BIB_TE_X has spread to the non-English-speaking world. BIB_TE_X 1.0 will address associated issues.
- More fields of study: The original BIB_TE_X users were from the technical world of mathematics and computer science. The BIB_TE_X 1.0 standard styles will do more to accommodate those in other fields, like the humanities.
- Enhanced sharing capabilities: There now exist huge `.bib`-file bibliographic databases, some available to users world wide. BIB_TE_X 1.0 will make the sharing of those databases easier.
- Better documentation: The BIB_TE_X 1.0 documentation will be more extensive.
- Froz_TE_N: For even better stability, BIB_TE_X 1.0 will be frozen. As with _TE_X 3.0, it will be upgraded for bug fixes only.

Some of the features planned for implementing those goals appear in the next section.

New features

Over the years I have accumulated a list of new features and probable changes for BIB_TE_X 1.0 and its standard styles. The list below is certainly not exhaustive, but it contains the most important items. Each one listed has a high probability of existing in BIB_TE_X 1.0.

- Software for generating customized bibliography style (`.bst`) files: In Santa Barbara I had claimed that there would be a Bibsty program to do that, similar in spirit to the processing that creates the four standard styles from the current template file `btxbst.doc`, but with lots more options. That Bibsty program, however, would partly duplicate what Patrick Daly has

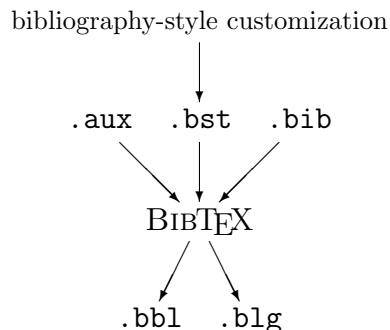


Figure 2: BIB_TE_X 1.0 input and output files.

done with the `custom-bib/makebst` package. So he and I have decided to collaborate for BIB_TE_X 1.0 on a system that will include the options of his current code, along with some things planned for the Bibsty program. This system will have, among other options, an easily changed symbolic name for each string that a bibliography style outputs to the `.bbl` file directly (such as ‘editor’ or ‘volume’), making it easier to, for example, convert bibliography styles from one language to another. Figure 2 shows how the new style customization system will fit into the scheme.

- Eight-bit input: Most current BIB_TE_X implementations can handle 8-bit input; BIB_TE_X 1.0 will guarantee, at a minimum, to support the character-set conventions of _TE_X 3. In addition, BIB_TE_X 1.0 may be able to accommodate Unicode.
- Multiple bibliographies within a single document: Many large documents contain several bibliographies—one bibliography per chapter for a book, or one per paper for a conference proceedings. Several solutions have arisen for handling such situations, but BIB_TE_X 1.0 will support multiple bibliographies directly.
- Enhanced sorting: The current BIB_TE_X does well with English, but doesn’t gracefully handle certain unusual sorting conventions of other languages, like Hungarian; BIB_TE_X 1.0 will have more powerful sorting capabilities for handling them.
- Reference-list back pointers: BIB_TE_X 1.0 will provide a direct mechanism for indicating in a reference-list entry where in the text that entry was cited. This is a useful feature that I suspect will become widespread now that our new typesetting technology makes it painless.

- An `@ALIAS` command: Suppose you have an entry in a database file that uses a different database key, say `knuth:tex`, from the cite key you prefer, say `texbook`; this might happen, for example, if that entry is in a shared database file. With BIBTEX 1.0 you will be able to keep the cite key and database key as is, as long as you put a command like

```
@ALIAS{texbook = knuth:tex}
```

somewhere in your database.

- A `@MODIFY` command: With BIBTEX 1.0 you will be able to create a slightly altered version of an entry. For example, you might want to add a `note` field to an entry in a public database file without having to repeat in your own personal database file all the information in that entry:

```
@MODIFY{laws-of-the-game,
  note = "Law~8 discusses restarts",
}
```

The `@MODIFY` command's database key should match the one from the public database.

- Distinguishing among identical database keys: If you are using two different database files that happen to use the same database key for different entries, you will be able to specify which entry you want by using a citation of the form

```
\cite{filename::database-key}
```

- A `.bib`-file extraction mode: BIBTEX 1.0 will have a mode that will let you extract just the information you need into a small `.bib` file. For example if you are submitting a paper to a journal that wants a `.bib` file in addition to a `.tex` file, but the bibliographic database you are using for the paper is huge, you can use the extraction mode to put just the entries you need for the paper into a separate `.bib` file that you can then send to the journal.
- A `\bibteoptions` (\LaTeX) command: Communication from (\LaTeX) to BIBTEX 1.0 will improve with this command.
- Extensions to the `\cite` command: Many citation styles aren't handled very gracefully by (\LaTeX)'s current `\cite` command. BIBTEX 1.0 and (\LaTeX) will more directly support more flexible `\cite` commands.
- Also appeared as: Sometimes a work appears in two different forms, for example as a journal article and then later in the author's collected works. BIBTEX 1.0 will have a mechanism to handle this.

- Name handling: BIBTEX 1.0 will have enhanced (but upward compatible) name processing, for example for complicated Spanish surnames, and for inverted-order Asian names.
- Standard-style changes: There will be lots of additions to the BIBTEX 1.0 standard styles, allowing many more options.
- New fields: The standard styles will also include a bunch of fields that are not now a part of the standard styles, including `day`, `isbn`, `issn`, `keywords`, `eprint`, `translator`, and `url`.
- New entry types: There will also be a few new entry types, including `@PERIODICAL` and perhaps `@PATENT`.
- `.bst`-language changes: There will be a few minor (but compatible) changes to the `.bst` language.
- `btmac.tex` update: These macros will be updated so that the user interfaces to BIBTEX 1.0 from \LaTeX and plain \TeX are as similar as possible.
- Documentation: The "BIBTEXing" and "Designing BIBTEX Styles" documents [5, 6] currently distributed with BIBTEX are neither as widely known nor as complete as they could be. For BIBTEX 1.0, the main documentation will be in a book, and will be much more thorough. Among other things, it will include a `.bib`-file grammar, so that those who are writing tools to manipulate the database files can make their software more robust.

Planned releases

2004 will see the release of an updated version of the `btmac.tex` macros. Probably toward the end of the year will come a bug-fix version of BIBTEX 0.99. (There are just a handful of actual bugs to BIBTEX itself, the most annoying being its mishandling of URLs, which didn't even exist when the current BIBTEX was released.)

There will be several beta releases of BIBTEX and its standard styles, leading up to 1.0; eventually one of those beta releases will be declared to be 1.0 and frozen.

Until BIBTEX 1.0 is finished, I will skim the `comp.text.tex` newsgroup for BIBTEX-related postings, so it suffices to post there anything you think I should see. In general that's a good place to post questions about BIBTEX—usually someone posts a useful reply. (Occasionally I'll reply by private email.)

References

- [1] Karl Berry and Oren Patashnik. `btxmac.tex`. Macros to make $\text{BIB}\text{T}\text{E}\text{X}$ work with plain TEX ; current version 0.99k, 13 November 1995.
- [2] Michel Goossens and Frank Mittelbach, with Johannes Braams, David Carlisle, and Chris Rowley. *The $\text{L}\text{A}\text{T}\text{E}\text{X}$ Companion*. Addison-Wesley, 2nd edition, 2004.
- [3] Helmut Kopka and Patrick W. Daly. *Guide to $\text{L}\text{A}\text{T}\text{E}\text{X}$* . Addison-Wesley, fourth edition, 2003.
- [4] Leslie Lamport. *$\text{L}\text{A}\text{T}\text{E}\text{X}$: A Document Preparation System*. Addison-Wesley, second edition, 1994.
- [5] Oren Patashnik. $\text{BIB}\text{T}\text{E}\text{X}$ ing. General documentation for $\text{BIB}\text{T}\text{E}\text{X}$ users, contained in the file `btxdoc.tex`, 8 February 1988.
- [6] Oren Patashnik. Designing $\text{BIB}\text{T}\text{E}\text{X}$ styles. Documentation for $\text{BIB}\text{T}\text{E}\text{X}$ style designers, contained in the file `btxhak.tex`, 8 February 1988.
- [7] Oren Patashnik. $\text{BIB}\text{T}\text{E}\text{X}$ 1.0. *TUGboat*, 15:269–273, 1994.
- [8] Unilogic, Ltd., Pittsburgh. *Scribe Document Production System User Manual*, fourth edition, April 1984. Chapter 12 and appendices E8 through E10 deal with bibliographies.
- [9] Mary-Claire van Leunen. *A Handbook for Scholars*. Oxford University Press, revised edition, 1992.



Abstracts — Bibliographies

Abusing \TeX : custom-bib as an example

Patrick W. Daly

Max-Planck-Institut für Aeronomie

daly@linmpi.mpg.de

Although \TeX is essentially a typesetting program, there are a number of “mis-uses” of it to accomplish what could be called off-topic programming.

The most complex example of this is no doubt the `fontinst` bundle, which creates the `.tfm` and `.vf` metric and virtual font files for PostScript fonts. Another service routine written in \TeX with no `.dvi` output is `docstrip`, which is part of the kernel \LaTeX installation, and which is vital for that installation. Originally `docstrip` was intended as a utility to remove comments from installation source files, but it now contains an even more powerful feature: it can customize the output code according to preselected options, and it can combine code from several source files.

It was this property that I employed to simplify an old problem with $\text{BIB}\TeX$: that every publisher has its own list of arbitrary formatting rules, and it is not easy to write new `.bst` files to meet these demands. Thus I wrote a generalized *master bibliography style*, or `.mbs` file, originally providing some 50 options for alternative bibliography style points, to be converted to a `.bst` file with `docstrip`. Today, my `merlin.mbs` claims well over 100 options.

The more complicated part of the `custom-bib` bundle, however, is interfacing with the user to manage the myriad choices, and to generate a `docstrip` batch file to do the actual conversion. This required yet another pseudo-program in \TeX language, `makebst`, which examines all the available options in the `.mbs` file, offers them to the user interactively, prepares the batch file, writes a protocol (for future changes of mind), and even runs the batch file. Without this, `merlin.mbs` would be totally unmanageable; it tames the wizard.

Such utilities written in the \TeX language are guaranteed to run on all systems where \TeX is installed. Any other programming language would involve problems of platform compatibilities and portability. This advantage outweighs the fact that as a programming language *per se*, \TeX is a monster.

$\text{MIBib}\TeX$ version 1.3

Jean-Michel Hufflen

Univ. of Franche-Comté

France

hufflen@lifc.univ-fcomte.fr

In October 2000, we started a new implementation of $\text{BIB}\TeX$, the bibliography program associated with \LaTeX . This implementation is so-called $\text{MIBib}\TeX$ (for “Multilingual $\text{BIB}\TeX$ ”) because it includes multilingual features. Multilingual bibliographies can be organised with respect to two approaches:

reference-dependent approach each reference of a bibliography section of a document is expressed using the language of the entry: for example, the month name of a reference to a book written in English (resp. French, German, ...) is given in English (resp. French, German, ...);

document-dependent approach all the references of a bibliography section of a document are expressed using the document’s language, as far as possible.

After the first version (1.1), Version 1.2 provided more flexibility about the specification of names within the fields `AUTHOR` and `EDITOR`. Formatting such names in a bibliography section is easier, too. These two versions use the bibliography style language (`.bst`) of $\text{BIB}\TeX$ and allowed us to define requirements for a new language for bibliography styles. The syntax of this new language is close to XSL-FO and this language will be used in Version 1.3. More precisely, the two languages will coexist in order to ease the transition between “old” styles and “new” ones. In the paper, we will:

- show how to design new styles with the “new” bibliography style language (it is completely described as an annex);
- explain how the coexistence between the two languages is organised.

(We expect to publish the full paper in the next regular issue of *TUGboat*. *Ed.*)

TeXShop in 2003

Richard Koch

Mathematics Department

University of Oregon

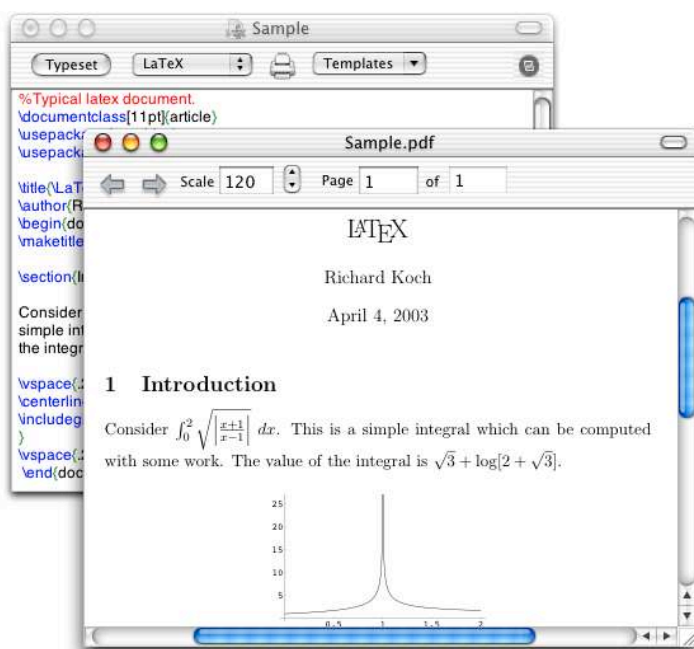
Eugene, Oregon, USA

koch@math.uoregon.edu

<http://www.uoregon.edu/~koch>

Abstract

TeXShop is a free T_EX previewer for Mac OS X, released under the GPL. A great many people have contributed code to the project. TeXShop uses t_EX and T_EX Live as an engine, typesetting primarily with pdftex and pdf_latex. The pdf output is displayed using Apple's internal pdf display code. I'll discuss recent changes in the program and plans for the future.



Introduction

I talked about TeXShop at the 2001 TUG conference. Mac OS X had just been released and still had quirks; during my talk, the Finder crashed. I typed option-shift-escape to bring up the Force Quit panel, restarted the Finder, and continued. After the talk, an audience member told me “I’m not very interested in your program. But it’s wonderful that you could restart the Finder without rebooting!”

A lot has changed since then. The Finder never crashes, the t_EX/T_EX Live distribution is stronger, lots of pdf bugs have been fixed, and TeXShop has new features.

What are we talking about here?

In case you don’t work on a Mac or haven’t used the program, I’ll start with an overview. When TeXShop first starts, a single window opens for the source code. This window is shown behind another window. The “Templates” button on the toolbar is a pulldown menu naming various pieces of text which can be added to the document; one of these pieces inserts starting code for a L^AT_EX document. The default L^AT_EX template text is shown in blue. Actually the editor colors all T_EX commands blue and these have come from the template. A beginner would choose this template and add text in the middle.

Pushing the “Typeset” button on the toolbar causes the document to be saved and typeset with `pdflatex`. A second window then appears showing the output. This window has tools to change pages, change the magnification, and rapidly move to a later page.

Next to the “Typeset” button is a pulldown menu labeled “LaTeX”. This menu contains a list of possible typesetting commands: Plain `TeX`, `LATeX`, `BibTeX`, `ConTeXt`. Also included are `MakeIndex`, `MetaPost`, and `MetaFont`. If one of these is chosen, the Typeset command will run that program.

I have described the simplest situation. Most users will open an existing `TeX` document and then both the source and preview windows will open side by side. After adding text to the source, users push “Typeset” to save and typeset the changed source. Multiple documents can be open at once.

Printing is straightforward. Select “Print” and the output will be printed on any inkjet or PostScript printer. If the printer handles color, illustrations and text can appear in color.

The Mac OS X landscape

There have been many changes in the Mac world since 2001. The most important from my point of view is that bugs in Apple’s pdf code have been quashed. I know of no outstanding bug.

My bug folder lists the following items, in order of correction:

- The Window Server Crash Bug. After one early release of OS X, users sent me illustrations and complained that TeXShop crashed when it tried to display them. When I saved these illustrations in a folder and selected one, the Finder tried to create a thumbnail preview and crashed. Apple fixed this bug very rapidly!
- The Accent Bug. Certain accented letters displayed with the accent displaced to the right rather than placed over the letter. Documents with accents printed fine on PostScript printers, but incorrectly on inkjet printers. This bug was fixed gradually, and as a result complaints from European nations gradually narrowed to just a few countries, and then disappeared.
- The Ghostscript 7 Bug. TeXShop has a second typesetting option in which a source file is typeset with `LATeX`, the dvi file is converted to PostScript with `dvips`, the PostScript file is converted to pdf by Ghostscript, and this pdf file is displayed. For a long time, Apple’s pdf routines ignored most fonts in documents created by Ghostscript 7 and the user ended

up with a page sparsely sprinkled with letters. Ghostscript 6 worked fine. This was a serious bug because NSF preprint repositories often had Ghostscript 7 pdf files. The bug was fixed in Jaguar and TeXShop users now use Ghostscript 8.

- The Large Letter Bug. Large symbols—integral signs, summation signs, parentheses—were cut off. Sometimes the top and bottom would display and the middle would be missing. Documents with such symbols would print fine on PostScript printers, but not on inkjet printers. This important bug was fixed in system 10.2.4.

Another significant change is Gerben Wierda’s rapid improvement of the `teTeX/TeX` Live installer, making it easy for users to maintain an up-to-date `TeX` distribution. Wierda’s i-Installer is completely independent of TeXShop and his distribution is used by most front ends on Mac OS X. I often receive email complaining that TeXShop doesn’t typeset correctly. If the email contains an example, I typeset it from the terminal and usually determine that the typeset output is exactly as the writer describes. So I write back “this is not my problem.” If I were more responsible, I’d look at the source code and try to find the error.

A final change is that `TeX` can now be created on Mac OS X using a great many different programs. Apple’s release of its own X window system makes X programs coexist nicely with Aqua programs, so users can easily use `emacs` and `xdvi`. At the Apple developer conference, I only saw one slide of a system running `TeX` and that fellow was using `xdvi`. Many wonderful front ends and tools have been released; these will be described by other people at the conference. I’m somewhat selfishly going to describe only TeXShop, but users should experiment before settling down with one solution.

Configuring TeXShop

In the course of this talk, I’m going to mention several people who have made enormous contributions. Don Knuth, of course, is responsible for everything we do; I’ve never met him. Two other people are here, Thomas Esser who created the `teTeX` distribution, and Hàn Thê Thành, who created `pdftex`. Salute!

TeXShop now has a reasonable help system, created by Martin Kerz (figure 1). I’ve haven’t been a great fan of Apple’s Help Viewer because it is slow, but I expect great improvements soon. Let me show you the help system because I want to point out one



Figure 1: TeXShop help system.

feature, the “How do I configure TeXShop?” section. I hope users will get in the habit of reading this section whenever they upgrade the program.

In the “Recommended Preference Changes” section, I recommend different window placement defaults. When TeXShop is first installed, it is configured to remember the last positions of the source and preview windows, and use these the next time TeXShop runs. This only makes sense if you work with one file at a time; otherwise it is likely that one of the windows was pulled out of the way before quitting, and its position may become the position TeXShop uses when it runs again. It is far more convenient to open source and preview windows side-by-side with fixed dimensions set in advance. To make this happen, typeset a small file and position the source and preview windows as you like. Then open Preferences. For “Source Window Position,” choose “All windows start at fixed position” and push the “Set with current position” button. Configure the Preview window similarly (figure 2).

While I’m dealing with preferences, I’d like to mention another one. TeXShop now starts configured to automatically convert eps files to pdf and tiff files to png during typesetting. We made this change when pdftex dropped support for tiff files. In the new configuration, the `-shell-escape` flag is set for \TeX ; the flag gives \TeX permission to run other programs during typesetting. The intended other program converts graphic files, but of course it could be any program. A few of our users worried that malicious code could be distributed as \TeX source; I suspect that those users were college teachers with dangerous students. One fix is to remove the shell-escape flag, which preferences permits. But there is also a new “Shell Escape Warning” check box in preferences. If it is set, then *the first time*

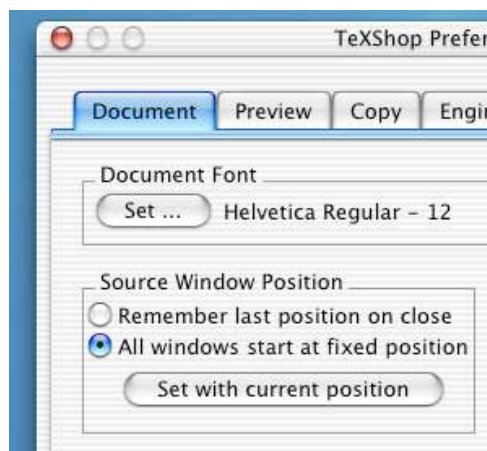


Figure 2: TeXShop window positioning.

a document is typeset during a TeXShop session, a warning dialog appears allowing the user to turn off shell escape for that particular document. This makes it easy to use graphic conversions for your own source but not for source received in email. In case you are wondering, I don’t have the flag checked on my machine. Live dangerously.

TeXShop has a few hidden preferences with no interface in the preference panel. One causes it to create a backup file every time a file is saved or typeset. The configure help section explains how to set that preference.

File encodings and the power of programmers in Japan

Internally, Cocoa editors use Unicode. But conventional \TeX cannot handle Unicode, so TeXShop converts a file to Unicode when it opens the file, and converts the Unicode back to something else when it saves. TeXShop now supports sixteen different encodings; I don’t recall how many encodings TeXShop supported in 2001 — maybe only one.

Three of these encodings are interesting. The first is utf-8, which preserves Unicode. I know a little about \TeX programs which can input Unicode, but hope to learn a lot more at the conference. To enter Unicode text, open System Preferences, choose the International module, and select the Input Menu tab. Add extra languages by checking the boxes on the left. A small flag will appear on your menu bar. To type in English, select the US flag (I suppose other flags also work); select other flags to input other characters. Notice that both Arabic and Hebrew are inserted right to left, and notice that ligatures work in Arabic, a language in which letters change shape at the end of a word. (See figure 3.)

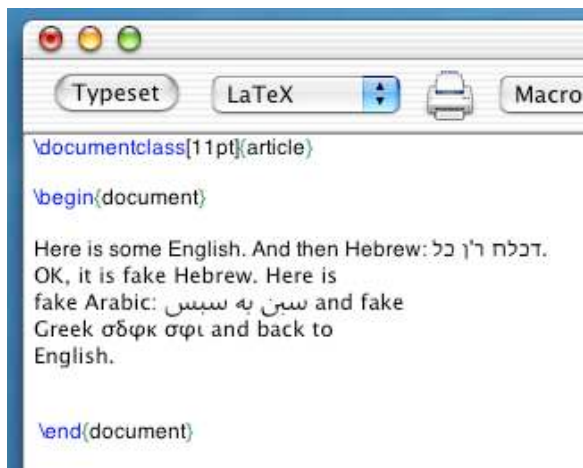


Figure 3: Unicode input.

Utf-8 encoding is slightly tricky because not all byte strings form legal input. Users sometimes open a document in utf-8 which they created in another encoding and just get a blank window because the conversion routine gave up. TeXShop never writes to files when they are opened, so the edit window should just be closed and the file opened with a different encoding. But in a panic, some users try to typeset the blank window. This is not a good move because TeXShop writes before typesetting. Users who often change file encoding should set the hidden preference to make backup files as protection against this sort of error.

A few months ago, a user asked for Russian encodings. Further email revealed that there are lots of Russian encodings in use; TeXShop now supports five such encodings. Users must deal with more than one encoding because they receive email. So in version 1.30, TeXShop supports selecting the encoding when a file is opened and when it is saved. When opening a file, the default encoding will be the one chosen in preferences, so most users can ignore the entire matter. If the encoding is changed, then the default for that one file when it is saved will be the new encoding.

Students of mine from Japan sometimes report that their friends have trouble with TeXShop. It was difficult to get a clear explanation because my students weren't TeX users themselves. A year after learning of the problem, I got a wonderful email from Seiji Zenitani with a clear explanation *and* the code to make TeXShop work. If I understand correctly, in System 9 Japanese fonts displayed the backslash character as a Yen symbol, and the Japanese keyboard had a yen symbol on the key which produced a backslash character. This created no problems for

TeX, of course. But when Unicode was introduced, it became necessary to separate the two symbols, and the Japanese keyboard began emitting a real yen symbol. The Japanese coped with this change by modifying TeX to use yen instead of backslash as the TeX control character. I mentioned this to a colleague with collaborators in Japan, and he said "yes, the damn yen problem."

At any rate, Zenitani took care of the problem and also localized TeXShop in Japanese. It is the localization I like best! Then around the time version 1.27 was to appear, Zenitani wrote me that he had heard of some smoother routines by someone else in Japan, who would write me soon. In that way I made contact with Mitsuhiro Shishikura, the most powerful programmer I know. Shishikura is a mathematician. He just told me that he has become chair of his department and I sent condolences.

Shishikura sent code to simplify TeXShop's handling of the yen problem. Then he casually mentioned that he had added a magnifying glass for the preview window (this is listed in my 2001 article as an important missing feature). So TeXShop finally had a magnifying glass. I was going to show you the glass below, but screen capture doesn't show it; apparently Shishikura's code is as mysterious to the Mac as it is to me.

No sooner was Shishikura's magnifying code in the program than he invented a Macro editor as well. I'd like to show you one nice feature of that editor. TeXShop has hidden preferences to set the background color of the source window and of the preview window, but these preferences must be set in Terminal. Suppose you wish to experiment with various backgrounds in preview documents. Create an Applescript macro in the Macro editor using the following code.

```

--AppleScript
do shell script "defaults write
    TeXShop Pdfbackground_R 0.5"
do shell script "defaults write
    TeXShop Pdfbackground_G 0.5"
do shell script "defaults write
    TeXShop Pdfbackground_B 0.5"
  
```

After executing this command, the background of the preview window will be gray the next time you typeset. Don't get too excited; the background will not print, but is useful for previewing if your document has colored text which is difficult to see against white.

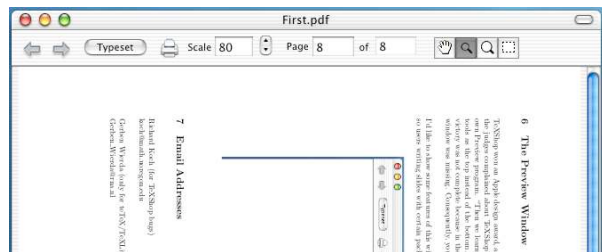
The TeXShop source code has a folder containing the email Shishikura wrote when he sent these modifications. It is a model of describing proposed

changes. Shishikura listed the changes needed in the nib files, and in each Objective C file. He described each new feature separately, so I could include some and not others. This contrasts with code changes I receive occasionally that are so massive they cannot be digested. Every source file in the project has been modified and as far as I can tell, there isn't a scrap of remaining code which I wrote myself. Moreover, the interface is completely different. What am I supposed to do with such a suggestion?

The Preview window

TeXShop won an Apple design award, and after the ceremony one of Apple's engineers told me that the judges complained about TeXShop's pdf window, whose design does not follow the design of Apple's own Preview program. "Then we learned," he said, "that Preview was redesigned in Jaguar with tools at the top instead of the bottom." Apparently that placated the judges. But our victory was incomplete—in the poster Apple put up after the ceremony, the TeXShop pdf window was missing! You couldn't tell from the poster what TeXShop actually did.

I'd like to show some features of the preview window introduced since 2001. First, the view can be rotated so users writing slides with certain packages view them in correct position.



I have often taught linear algebra. After writing the code to transform window coordinates and then testing it and discovering that the window was transformed God-knows-where, I have new humility. Any linear algebra course I give in the future will be error forgiving.

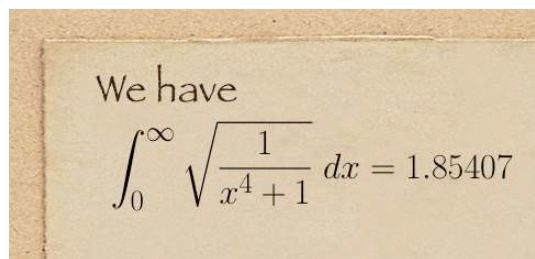
For version 1.30, just released, Shishikura revised the pdf display code, introducing new display options like scrolling through the entire document and displaying two pages side by side. He cleaned up the code tremendously, so pages are centered rather than displayed in the bottom right of the window and ... well, I don't want to think about the old code.

Indeed, there are so many new options that a couple have minor bugs which will be cleaned up

soon. The lesson is that if I always use "Single page" option and Shishikura always uses the "multipage" or "double multipage" option, then nobody is testing the "double page" option.

Copy and paste; drag and drop

Shishikura also added the ability to select a portion of the pdf output and copy it to the clipboard or drag it to another application or the desktop. This procedure can be controlled by new preferences or by menu commands. For example, the file type of the copy can be pdf, tiff, jpg, png, or pict; the foreground and background colors of the copied image can be selected in preferences for some of these types.



Above, for example, is a section of text copied into a Keynote slide in pdf format. But users working with Keynote should also consider J. McKenzie Alexander's wonderful *Equation Editor*. See the site evolve.lse.ac.uk/software/EquationEditor/.

The LaTeX panel

Geoffroy Lenglin created a great symbol panel for TeXShop. Clicking on a symbol inserts the appropriate source code into the document.

I'd like to tell a little of the story of this panel because it shows that a contributor should not interpret silence as a "no". Lenglin sent this panel while I was working on other TeXShop features, so it didn't immediately appear. I couldn't just glue it in because there were interface features to consider: hiding the panel when the preview display is active, remembering the Panel position when TeXShop quits, etc. Then tasks at the University intervened and Lenglin heard nothing from me for many months. When I finally had time, I wrote Lenglin something like "you may think I've forgotten, but now I'm ready to add the LaTeX Panel." He replied "it is good that you wrote, because I have just finished a masters degree in Aeronautical Engineering at MIT and this email address will become inactive. I'm leaving for France." Whew!

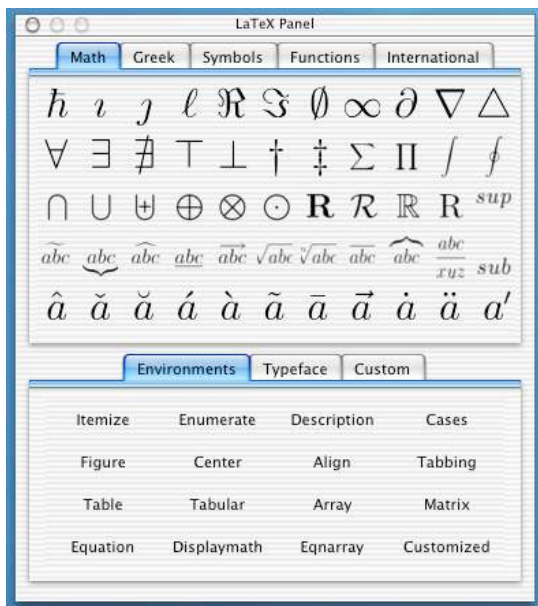


Figure 4: L^AT_EX symbol panel.

The above quotations are approximate because I can no longer find those messages in Mail. But I discovered old email from him containing a French localization of the help files. They didn't get into TeXShop! Apologies. If you write me and I ignore the email, please write again. And again.

Split window editing, syntax coloring, etc.

There have been other changes. The editing window can be split so you can view one section of source while writing another. Various methods have been added to speed typing; for instance, you can request that pressing the double quotation key insert the T_EX code for “ ” with the cursor placed between the quotation marks. Command completion is also possible; typing part of a word and pushing escape will cycle through all possible completions, and the completion can be a complicated phrase which need not contain the letters originally typed. (However, this system does not use an existing dictionary, so words must be added before they are available for command completion. Improvements to this system will appear in the future.)

For many months, users complained that when all windows in TeXShop are closed and another application is temporarily active, clicking on TeXShop in the dock created a new empty source window. My response was that the Apple Interface Guidelines require this behavior. Indeed the behavior is automatic in Cocoa. Eventually, Gerben Wierda stumbled across the “applicationShouldOpenUntitledFile” call in the API's and he wrote: “maybe the

Guidelines require it, but Apple themselves provided a way to get around the guidelines.” So a preference item now allows users to control this behavior.

When Jaguar appeared, TeXShop syntax coloring slowed way down. Users who let the program wrap lines and thus create very long paragraphs had real problems. I spend half of 2002 working on this problem . . . don't remind me.

Other changes have been made. The “root file” design was improved following a user suggestion, Zenitani fixed the tags menu so several tags can have the same text, TeXShop can display ps, dvi, jpg, tiff, pdf, and eps files, converting to pdf when appropriate. The version history file can be consulted for a full list of changes.

The TeXShop design philosophy

I want to tell you about one of my colleagues. His office is two blocks from mine, and he sometimes calls with computer questions. A typical session goes like this:

I just received email and it contains tex code. How do I typeset?

Is the code in an attachment?

No. There's a begin[document] and then lots of English.

Ah. Copy the source, paste it into a blank TeXShop window, and hit the Typeset key.

Wait a minute. Let me get a piece of paper. OK. Step one. Copy the source...

My colleague learned T_EX and he started writing books. Wonderful books. Mathematicians read his books even when they are in a different field.

You have a natural question. “He started learning T_EX? How many support calls did *that* require?”

The answer is *none*. My colleague immediately understood the connection between T_EX and mathematics; he got a few books, and he proceeded. As for software, he doesn't want it to interfere with his life. Once he learns how to do something, he doesn't want to learn a faster way, or upgrade and get more features, or find more buttons on the interface. I went to his office and discovered that he used the original TeXShop preferences and his windows opened at random spots. “I know a better way,” I said, “let me change that.” “No, no...”

I write TeXShop for colleagues like that. The highest compliment you can give is “this program is pretty simple.” I'm happy to add features for advanced users, but only if they don't cause beginners

to say “what’s this button for?” TeXShop should live in the background. It should be there if it is needed but it shouldn’t intrude.

The future

When users send suggestions, there is a temptation to pick those that are easy to do, add lots of minor features, and ignore the central issues. Let me concentrate now on two primary requests. I don’t know how to do either one, so no promises are being made.

The first request is for an editing option to provide hard line feeds when the editor wraps text. *Textures* works like that; as a user types, the lines wrap; if the window is later resized, the line feeds remain and the lines do not reformat to fit the new window size. Users who request this feature complain that when they send TeXShop files to friends, the friends have versions of T_EX which cannot deal with long lines. They also say that some utilities like diff have trouble with long lines.

People have different typing styles. I am in the habit of typing a line feed before I come to the end of a line, so I’m never bothered by these problems. When users send me files with long, long paragraphs, I’m surprised. Hard line feeds go against the basic philosophy of TeXShop because they do something behind the user’s back. And hard line feeds are certainly not required by the standard programs in t_EX/T_EX Live.

Of course, it is easy to write a three line C program which can convert a T_EX source file to a file whose lines are all short. I’ve sent such programs to people who’ve written me.

Cocoa doesn’t readily yield up the soft line feeds is inserts when it formats for the screen. I’ve read through the text API’s and I don’t see an easy way to proceed. But a few users are very insistent that they need those line feeds!

I’ve tried to postpone until the bitter end the main request by far: synchronicity. With this *Textures* feature, a user can click on a spot in the preview and immediately be taken to the corresponding spot in the source code; conversely the user can click on a letter in the source code and be taken to that letter in the typeset preview.

This feature has been implemented by other T_EX preview systems, so in the last few weeks I’ve been reading about them. I’ve read about source specials in dvi files, about the src-special flag for tex and pdftex, about dv2dt, about dvii, vpe, and srcltx. These things look helpful for a rough version of *Textures* synchronization, but I need help. Big time help.

Very early in the life of TeXShop, a user asked for synchronization using approximately the following words: “and by synchronization I do not mean that I’ll be happy if it takes me to the same paragraph; I insist to be taken to the *exact* corresponding character in the source! That or nothing.” Gosh! Intimidating. Blue Sky doesn’t charge money for nothing.

TeXShop has two typesetting modes. One produces a dvi file (and uses it to create a pdf); the other produces a pdf directly. It is possible that some of the utilities listed above would help with one method but not the other. If I must choose, I’d much rather have synchronicity using pdftex than synchronicity using tex with dvi files.

Perhaps one of you has a complete solution ready to go; it pays to ask.¹ I’d really like to see a flag in pdftex which causes it to output a separate file containing this information:

1. the name of the source file; when it changes due to input statements, the name of the new source
2. for each individual character typeset, the line number and position within that line of the character in the source, and the page number and location in pixels of that character after it is typeset.

I wouldn’t know how to handle sections of text created by macros.

Maybe that is way too much. I’m going to end with a list of questions.

- Are there utilities on the web which can decipher a dvi file and print a readable list of the contents, or of part of the contents?
- For example, could such a utility print the location of the start of each line, and the position within that line of the start of each character?
- Are there utilities on the web which can decipher a pdf file and print a readable list of the contents, or of part of the contents?
- Adobe Acrobat allows users to drag the mouse and select a portion of text. It allows users to search a pdf document. How are those tasks done?
- pdftex has a flag named `-src-specials`. Is there a utility which can read a pdf file and output these source specials?

If you have any answers or information about these matters, please talk to me during the conference or write afterwards.

¹ Since this paper was written, `pdfsync.sty` has been achieved through the efforts of many volunteers. See Jérôme Laurens’ forthcoming presentation on this at TUG 2004. *Ed.*

Creating labeled stand-alone figures in L^AT_EX using WARMreader and Adobe Illustrator under Mac OS X

Francesco Costanzo

Associate Professor

Department of Engineering Science and Mechanics

The Pennsylvania State University

212 Earth & Engineering Sciences Building

State College, PA 16802-1401

USA

costanzo@engr.psu.edu

<http://www.esm.psu.edu/faculty/costanzo/>

Gary L. Gray

Associate Professor

Department of Engineering Science and Mechanics

The Pennsylvania State University

212 Earth & Engineering Sciences Building

State College, PA 16802-1401

USA

gray@engr.psu.edu

<http://www.esm.psu.edu/faculty/gray/>

Abstract

In this paper we discuss our experience as Mac users who lived through the transition that took us from L^AT_EXing with *Textures*TM under Mac OS 9 to L^AT_EXing with t_EX and T_EXShop under Mac OS X. For us, the most difficult yet rewarding aspects of this transition concerned the creation of “stand-alone” figures containing labels and annotations prepared using L^AT_EX. By “stand-alone” figure we mean a figure in a given format (usually EPS or PDF), which can be imported by one of the many graphics import commands available in L^AT_EX.

Around April/May 2001, after our switch to Mac OS X, t_EX and T_EXShop, because of a number of issues concerning font management under OS X, translation between EPS and PDF formats, as well as issues with T_EX fonts in Adobe IllustratorTM, we could no longer use our old labeling strategy. After some experimentation, we found a solution using the WARMreader package developed by Ross Moore and Wendy McKay along with an Illustrator plug-in called **MarkedObjects**, created by Tom Ruark. In this paper we will describe why we are interested in creating stand-alone figures, why we chose to use WARMreader, and some of the techniques we have developed to create figures. The paper also describes the use of an AppleScript created by the authors to aid the figure labeling process.

Introduction

A good number of scientific journals as well as conferences now accept electronic submission of papers. Often, the instructions provided to authors require that the figures included in the paper be provided as files, typically in Encapsulated Postscript (EPS) format, distinct from the file containing the manuscript. Furthermore, depending on the journal,

there is no guarantee that the final paper will be typeset using L^AT_EX. This means that if one’s solution to including T_EX output in a figure relies on the ability to typeset the manuscript using (L^A)T_EX, then trouble may be at hand. Hence, if one wishes to annotate a figure using L^AT_EX fonts or symbols, one must be able to create the figure in question in such a way that the notes and symbols in the figure

are not generated along with the rest of the text when the main manuscript file is typeset. In other words, the annotated figures must be created so as to be a self-contained or stand-alone file.

In this paper, we will describe how we have dealt with the creation of stand-alone figures in the past and the strategies we have developed to accomplish this task with the advent of Mac OS X. In particular, the following two sections will be devoted to presenting a history of our approach to the problem, especially after we started using $\text{te}\TeX$ and $\text{T}\TeX\text{Shop}$ under Mac OS X. In the rest of the paper we will present examples of how we currently create annotated figures in the form of a tutorial.

Stand-alone figures before Mac OS X

Before the release of Mac OS X, we used *Textures*TM for $\text{L}\text{A}\text{T}\text{E}\text{X}$ ing our papers. As far as the creation of figures is concerned, Adobe Illustrator has been our application of choice for many years.

In order to include $\text{L}\text{A}\text{T}\text{E}\text{X}$ fonts into an Illustrator figure we were accustomed to simply creating a temporary $\text{L}\text{A}\text{T}\text{E}\text{X}$ document containing all of the symbols to be included in the figure. After typesetting the temporary file, we made use of a feature in *Textures* allowing us to save the typeset output in the Adobe Illustrator 88 file format, which, for all intents and purposes, is the same thing as saving the output as a PostScript file. Next, we would open the newly created Adobe Illustrator 88 file in Illustrator and simply *copy* the $\text{L}\text{A}\text{T}\text{E}\text{X}$ symbols from this file into the file containing the figure to be annotated. The precise positioning of the $\text{L}\text{A}\text{T}\text{E}\text{X}$ symbols within the figure was a trivial matter in that it was accomplished by simply using the mouse to drag the annotation objects to their proper location.

The successful outcome of this very simple procedure relies on the two applications used to carry it out, namely Adobe Illustrator and *Textures*, accessing and correctly/consistently using the same set of fonts. By ‘correct/consistent’ use of fonts we mean, as we understand it, that the applications in question use the same font map.

Switching to Mac OS X

Once we switched to Mac OS X, since *Textures* was not available under this new operating system and since we had decided to make a conscious effort not to rely on the Classic environment,* we decided to adopt the $\text{te}\TeX$ distribution as our $\text{T}\TeX$ and $\text{L}\text{A}\text{T}\text{E}\text{X}$

base, and to use $\text{T}\TeX\text{Shop}$ as our editing and previewing environment.

This transition felt rather uneventful until we started dealing with the creation of stand-alone figures. Our first instinct was to simply try the same strategy we had always used. Hence, we started by creating a temporary $\text{L}\text{A}\text{T}\text{E}\text{X}$ document with $\text{te}\TeX$ and $\text{T}\TeX\text{Shop}$, saved the output as a PostScript file, opened it in Adobe Illustrator . . . and we discovered that the $\text{L}\text{A}\text{T}\text{E}\text{X}$ fonts, as seen by Illustrator, had been translated, for the most part, into garbage. In fact, not all of the symbols were misinterpreted. Those that were misinterpreted were primarily, but not exclusively, mathematics and Greek symbols, that is, most of the symbols we use in our figures.

At first, we thought that this behavior was caused by the fact that the $\text{te}\TeX$ fonts are installed in such a way that they are not available for use by other applications. Hence, we proceeded to install, under Mac OS X, the same PostScript fonts used by *Textures* under previous versions of the operating system. These fonts were installed in a location such that they could now be seen by Illustrator. This attempt at fixing the problem did not work, thus leading us to the conclusion that (i) the font mapping used by $\text{te}\TeX$ was different from that seen by Illustrator in the fonts used by *Textures*; and (ii) that our old strategy for labeling *any* figure was to be abandoned altogether.[†]

The search for alternative figure labeling methods (which took several weeks of unsuccessful attempts) led us to a package called **WARMreader** developed by Ross Moore and Wendy McKay (McKay and Moore, 1999; Moore, 2001). The capabilities offered by this package will be outlined in the next section.

The WARMreader package: an overview

We begin this section with a disclaimer: we do not intend for this article to be exhaustive in its description of the **WARMreader** package. In particular, we will limit it to the description of those features offered by **WARMreader** that are most relevant to the type of figure labeling we do on a daily basis.

WARMreader allows one to overlay any graphics objects imported in $(\text{L}\text{A})\text{T}\text{E}\text{X}$ with labels that can be defined within the very $(\text{L}\text{A})\text{T}\text{E}\text{X}$ file in which said graphics objects are imported. In this sense, **WARMreader** can be thought of as conveniently providing and extending the facilities that are made

*In Mac OS X, the *Classic environment* refers to the running of Mac OS 9 as a process within Mac OS X so that pre-Mac OS X applications can be used.

[†]We tried several other strategies to solve the font mapping problem but with no success.

available by packages such as X_y-pic (Rose, 1991) and PSfrag (Barratt, Grant, and Carlisle, 1996).

Roughly speaking, WARMreader overlays given (L^A)T_EX expressions at specified locations over the imported graphics object. The coordinates of the (L^A)T_EX labels to be overlaid need to be stored in a text file with the same name as the graphics file but with .bb extension. The coordinates in question are expressed in points and measured with respect to the lower left corner of the graphics object's bounding box. Provided that we will come back to a more precise description of a label's placement over a figure, we think that it is important to point out how the successful and convenient use of WARMreader relies on the user's ability to generate the labels' coordinates which, in general, could be a rather time consuming task.

For Adobe Illustrator users, the process of generating the labels' coordinates is greatly facilitated by the use of the MarkedObjects plug-in, created by Thomas Ruark at Adobe (cf. McKay, Moore, and Ruark, 2001). Although we will describe the use of the plug-in later in various examples, here we simply anticipate the fact that the use of the MarkedObjects plug-in allows one to define label position markers within the figure itself along with the definition of the label. Furthermore, the use of the plug-in is such that the required .bb file is created automatically, with a complete list of all of the labels' markers and the (L^A)T_EX annotations to be overlaid onto the figure.

WARMreader and stand-alone figures

By its very nature, the WARMreader package is a tool that can only be used from within a (L^A)T_EX document. This implies that, contrary to what we stated earlier, the labeled figures one generates are not stand-alone objects but are objects embedded in a document. Hence, in order to use WARMreader to create stand-alone figures one must devise a strategy to extract the figures from the document containing them and endow each figure file with the necessary bounding box information. The strategy we have adopted is as follows:

1. we create a L^AT_EX document consisting of a single page with `\thispagestyle{empty}`;
2. we then import the graphics object, to be annotated using WARMreader, into this document;
3. after adding the labels, we typeset the document using T_EXShop with its settings as shown in Fig. 1, which displays the Engine tab within the T_EXShop preferences. This step yields several files, two of which are of PostScript and

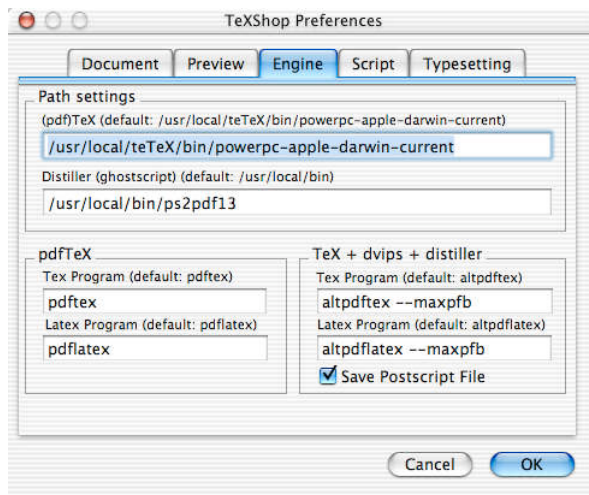


Figure 1: T_EXShop Engine tab window.

PDF type, each consisting of a single page with the annotated figure.

4. At this point we use the `ps2epsi` command made available by Ghostscript to turn the PostScript file created by `dvips` into an EPSI file (EPS file with a bitmap preview);
5. finally, we distill the EPSI file in question using the `epstopdf` command.

At the end of this operation we have two files:

- (a) an EPS file with the annotated figure and the proper bounding box information; and
- (b) a PDF file with, again, just the annotated figure with the proper bounding box information.

These files can be treated as graphic objects that can be imported in any other application supporting the import operation of images in EPS and/or PDF formats.

Remarks and a script

The procedure described in the previous section is not logically complex but it is involved. In particular, what makes it involved is the combination of having to carefully position the labels on the figure and, when everything is in place, having to go through several command-line instructions in the Unix terminal under OS X to get the final result.

To facilitate the use of this procedure, both by ourselves and by our graduate students, we have created a drag-and-drop AppleScript which makes the procedure essentially automatic. The only non-automatic part of the procedure remains the fine-level adjustment of the labels on the figure or the *nudging*, as it is referred to by Wendy McKay and Ross Moore (McKay and Moore, 1999).

Hence, to illustrate what we actually do in practice, we now present an example. This example can be thought of as a tutorial for the use of the Illustrator `MarkedObjects` plug-in, `WARMreader`, and the AppleScript we have created to help make the overall process easy to perform. The script in question has been dubbed `WARMFIGToPDF` (Costanzo and Gray, 2002).

Before presenting any examples, we feel that it is important to clearly identify the basic tools with which the examples have been created. Here is a detailed list of the operating environment that we currently use and under which we know this procedure works:

1. Operating system: Mac OS X 10.2.4;
2. `TEXShop`: version 1.28, dated January 29, 2003 (the most current information on `TEXShop` can be obtained from <http://www.uoregon.edu/~koch/texshop/texshop.html>);
3. `teTEX`: our current version of `teTEX` should be more properly referred to as `teTEX + gwTEX*` and is dated February 10, 2003 (available from <http://www.rna.nl/ii.html>);
4. `Ghostscript`: version 7.05 (available from <http://www.rna.nl/ii.html>);
5. `WARMreader`: version 1.2, dated July 5, 2001 (downloaded from the official `WARMreader` website at <http://www-texdev.mpce.mq.edu.au/WARM/WARMhome/>);
6. Adobe Illustrator: version 10.0.3;
7. `MarkedObjects` Illustrator plug-in: release date of May 6, 2002 (available at <http://www.cds.caltech.edu/~wgm/WARM/adobe/>);
8. `WARMFIGToPDF`: version 1.0 (can be downloaded at <http://lpcm.esm.psu.edu/~gray/wftpdf.sit>).

Example: Labeling the vertices and the center of a rectangle

This is a simple example in which we start by running Adobe Illustrator to create a simple rectangle. Once the rectangle is created, the file should be saved as an Illustrator EPS (EPS) file, as shown in Fig. 2. For future reference, `rectangleFig.eps` is the name we have given to the Illustrator EPS file used in this example. Figure 3, shows the content of the Illustrator window, namely a gradient filled rectangle. In addition, the figure displays the location of the `Marked Objects Tool`, which the `MarkedObjects` plug-in places among the `Pen Tools`.

*Where we understand that ‘te’ stands for Thomas Esser and ‘gw’ stands for Gerben Wierda.

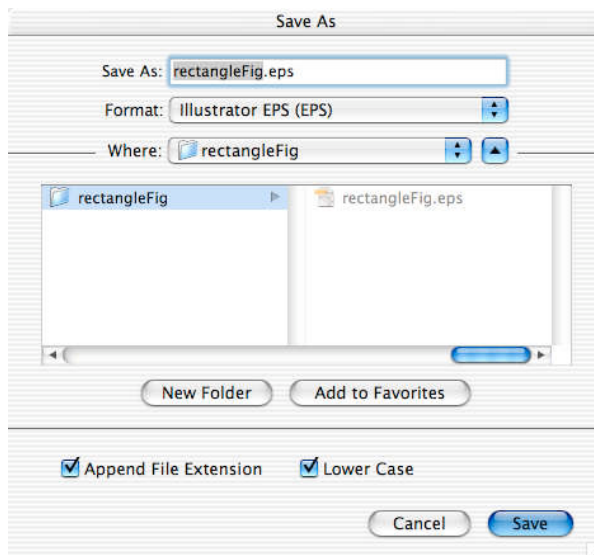


Figure 2: Illustrator ‘Save as’ dialog window.

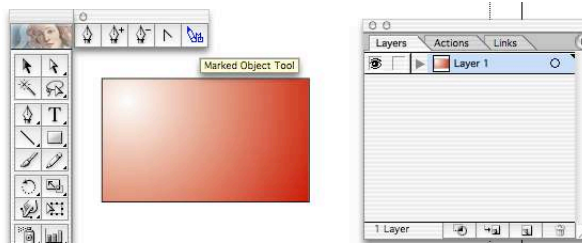


Figure 3: `MarkedObjects` Illustrator plug-in tool.

Once the figure is created, the labeling process is accomplished by selecting the `Marked Objects Tool` and creating (by clicking) as many labels as one wishes to create. In particular, we would like to create five `Marked Objects`, one for each vertex and one for the center of the rectangle. We begin by placing them in arbitrary locations, as shown in Fig. 4. Each `Marked Object` is numbered sequentially and consists of a `x`-symbol with the object’s num-

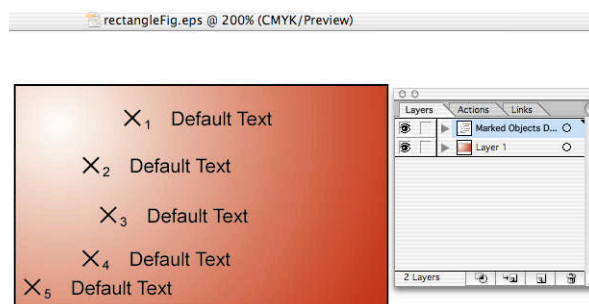


Figure 4: `MarkedObjects` objects and layer.

ber appearing as a subscript. The remaining part of each Marked Object is a string that can be edited using the Text Tool. The default appearance and text of these objects can be modified by the user via the editing of the `MarkedObjects` preferences. This can be done after opening the `MarkedObjects` dialog window. This window can be opened via: Window → SDK Dialogs → Show Marked Objects Dialog. This results in the appearance of the dialog window shown in Fig. 5. As can be seen in this figure, the

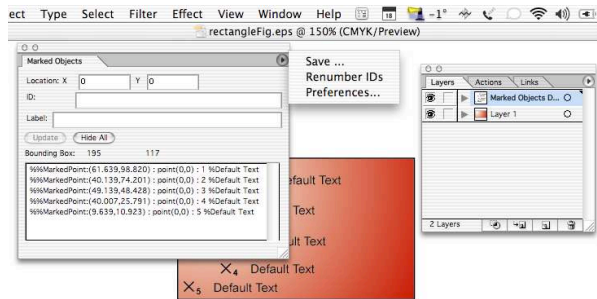


Figure 5: MarkedObjects dialog window.

`MarkedObjects` dialog window displays a variety of pieces of information, including the figure’s bounding box, as well as each label’s text and coordinates. The `MarkedObjects` preferences can be edited by clicking on the triangle-labeled radio-button placed in the upper right corner of the dialog, as shown in Fig. 5.

Going back to the description of what happens once the Marked Objects Tool is used and with reference to Fig. 4, it should be observed that using this tool *automatically* results in the creation of a new layer, called `Marked Objects DO NOT MODIFY`. Furthermore, once the layer is created, any `Save` operation performed by the user results in the creation (and subsequent updating) of a text file that is placed in the same location as the Adobe Illustrator file containing the figure. The new file in question is automatically given the same name as the Illustrator EPS file, except for the extension, which is automatically set to `.bb`. As far as the labels are concerned, their creation and editing can be done in any order desired by the user. In this example, the labels were first created (by clicking five times at arbitrary locations with the Marked Objects Tool) and then edited using the Text Tool. Figure 6 displays the Marked Objects after their text has been edited.

Now that the labels have been created, we can proceed to placing them at desired locations. In Illustrator, Marked Objects can be moved just like any other graphics object. When placing labels at de-

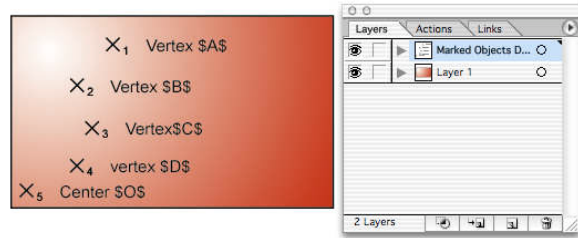


Figure 6: MarkedObjects labels.

sired locations, is it useful to keep in mind that the coordinates that the `MarkedObjects` plug-in assigns to a label are the coordinates of the center of the \times -symbol, measured (in points) with respect to the lower left corner of the bounding box of the figure. As far as the figure’s bounding box determination is concerned, by monitoring the information provided by the `MarkedObjects` dialog window, it is easy to see that this calculation disregards the position of the Marked Objects. Continuing with the labeling process, in Fig. 7 we can see the labels in their fi-

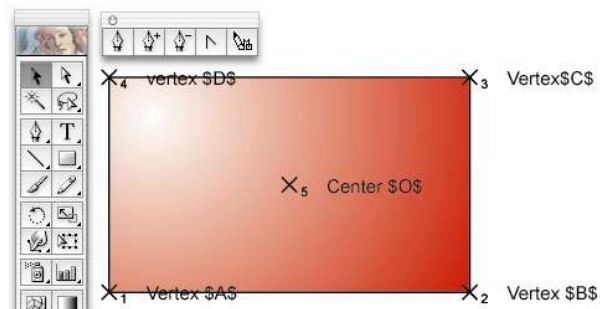


Figure 7: Marked Objects in their final position.

nal position. The labeling process is now completed by hiding the Marked Objects Layer (by clicking on the “eye” in the Layers palette), saving the resulting Illustrator file, and closing the file in question.

At this point, we have two files in our working folder: the Illustrator EPS file and its companion `.bb` file. Next, we simply drag and drop the Illustrator EPS file onto the AppleScript `WARMFIGToPDF`. The outcomes of this operation are:

1. the renaming of the Illustrator EPS file and the text `.bb` file via the prepending of the `.eps` and `.bb`, respectively, extensions with the character string ‘-AI’;* and

*If the original Illustrator file name ends in `-AI`, the renaming does not take place and the newly created `.tex` file will have the same name of the Illustrator file without the `-AI` ending.

- the creation of a \LaTeX file carrying the original name of the Illustrator EPS file (with extension `.tex`).

Before proceeding any further, a remark of practical importance must be made. When assigning a name to an Illustrator file, one needs to keep in mind that, at least with the current version of Adobe Illustrator under Mac OS X, long file names *are not* supported. The importance of this observation lies in the fact that the current version of `WARMFIGToPDF` does not check whether or not the addition of the string ‘-AI’ is compatible with the Illustrator file name length requirements. As the reader can imagine, we have lost more than one long-named Illustrator file by running `WARMFIGToPDF` while the file in question was still open in Illustrator.

In the interest of completeness, we now report the content of the `.tex` file `WARMFIGToPDF` creates.

```

%&latex
\documentclass[10pt]{article}

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PACKAGES %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\usepackage[expert]{lucidabr}          %
\usepackage{amsmath}                  %
\usepackage{amssymb}                  %
\usepackage{amsthm}                   %
\usepackage{exscale}                  %
\usepackage[mathscr]{eucal}           %
\usepackage{ifthen}                   %
\usepackage[pdftex]{graphicx}         %
\usepackage[dvipsnames]{color}        %
\DeclareGraphicsExtensions{.pdf, .jpg} %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Settings for FC, GLG, MEP books.
%\input{../../../../Settings/commands}

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% EXOTIC PACKAGES: Figure Labeling within LaTeX %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\usepackage[all,color,frame,import]{xy} %
\usepackage{warmread}                  %
\let\xyWARMprocess\xyWARMprocessMo    %
\let\WARMprocessEPS\WARMprocessMoEPS  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% USEFUL WARMreader COMMANDS %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\xyMarkedPos{##}*!D( 0.00)!L( 0.00)%   %
\txt{\rotatebox{90}{yAxis-Title}}      %
\xyMarkedPos{##}*!D( 0.00)!L( 0.00)%   %
\txt{\includegraphics[scale=x.x]{InsetGraph}} %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\begin{document}
\thispagestyle{empty}
%
\WARMprocessEPS{rectangleFig-AI}{eps}{bb}
%
\begin{xy}
\xyMarkedImport{}

\small

% Marked Point Number: 1
% MarkedPoint:(0.500,0.500) : point(0,0) : 1 %Vertex $A$
\xyMarkedTextPoints!D( 0.00)!L( 0.00){1}

```

```

% Marked Point Number: 2
% MarkedPoint:(194.314,0.500) : point(0,0) : 2 %Vertex $B$
\xyMarkedTextPoints!D( 0.00)!L( 0.00){2}

% Marked Point Number: 3
% MarkedPoint:(194.314,115.964) : point(0,0) : 3 %Vertex $C$
\xyMarkedTextPoints!D( 0.00)!L( 0.00){3}

% Marked Point Number: 4
% MarkedPoint:(0.500,115.964) : point(0,0) : 4 %vertex $D$
\xyMarkedTextPoints!D( 0.00)!L( 0.00){4}

% Marked Point Number: 5
% MarkedPoint:(97.407,58.232) : point(0,0) : 5 %Center $O$
\xyMarkedTextPoints!D( 0.00)!L( 0.00){5}

\end{xy}
%
\end{document}

```

This `.tex` file is obtained by the use of a simple template which:

- includes the `WARMreader` package (along with all the other packages one wishes to include by default*);
- properly sets up the `xy` environment, which will overlay the graphic image with the \LaTeX generated labels;
- includes the `graphics` file containing the image to be labeled;
- includes every Marked Object created in Illustrator, preceded by a summary of the information it carries by default, i.e., stored in the `.bb` file.

By default, we have chosen to include the various Marked Objects by invoking the `WARMreader` command `\xyMarkedTextPoints`. For those users with an understanding of `Xy-pic` and `WARMreader`, it is clear that this is simply a personal choice. Furthermore, it should be noted that every Marked Object is also accompanied by the syntactical elements `D` and `L`, which allow one to *nudge* the object’s position in the vertical and horizontal directions, respectively. By default the the nudging amount is set to zero.

The \LaTeX file thus created is ready to be typeset and the outcome of that typesetting will be, among other things, two files, one PostScript and one PDF file. At this point, each of these files provides a *page* containing the annotated figure. Figure 8 displays the content of the working folder after typesetting the file `rectangleFig.tex`.

The files `rectangleFig.ps` and its PDF counterpart are almost the final desired product. The only feature they lack is a bounding box that properly encapsulates the figure. As mentioned earlier, turning the file `rectangleFig.ps` into an equivalent

*The AppleScript source, found in the AppleScript Studio project, needs to be modified in order to change what is included by default.

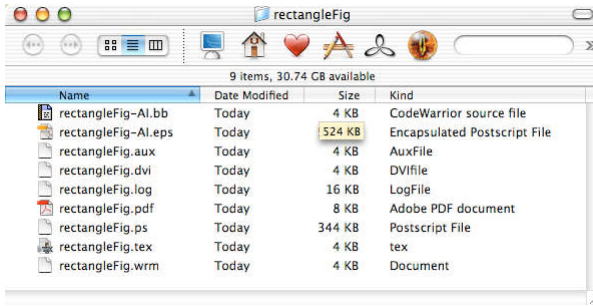


Figure 8: Content of the working folder after typesetting the `.tex` file created by WARMFIGToPDF.

EPS file is accomplished by operating on this file via the Ghostscript command `ps2epsi`. To be precise, this operation yields a file of EPSI type. As it turns out, and as will be discussed in greater detail in the next to the last section of the paper, the bounding box computed by `ps2epsi` is not entirely correct in that it often causes elements of the figure to be slightly clipped. Hence, to avoid any unwanted clipping, after creating the `rectangleFig.epsi` file, we edit its bounding box so as to enlarge it by one point in each direction. The EPSI file with the modified bounding box is given the name `rectangleFig.eps`. Finally, this file is turned into a corresponding PDF file (i.e., named `rectangleFig.pdf`) by invoking the Ghostscript command `epstopdf`.

The AppleScript application WARMFIGToPDF makes all of the operations just described automatic. Specifically, once the file `rectangleFig.tex` is created and typeset (i.e., once the working folder content is that in Fig. 8), one only needs to drop the file `rectangleFig.ps` onto WARMFIGToPDF. The outcome of this operation results in the creation of the desired EPS and PDF files, along with the deletion of all the files that are not needed for preserving the capability of future corrections to the figure. Hence, in the end, the content of the working folder is that depicted in Fig. 9. The final labeled figure is shown in Fig. 10. Clearly, the figure needs some adjusting. Hence, at this point, one can go back into the working folder and edit the `rectangleFig.tex` file to make the due corrections and adjustments. For example, editing the content of the `xy` environment as listed below makes the figure appear as depicted in Fig. 11.

```
\begin{xy}
\xyMarkedImport{} \small

% Marked Point Number: 1
% MarkedPoint:(0.500,0.500) : point(0,0) : 1 %Vertex $A$
\xyMarkedTextPoints!D(-1.50)!L( 0.00){1}

% Marked Point Number: 2
```

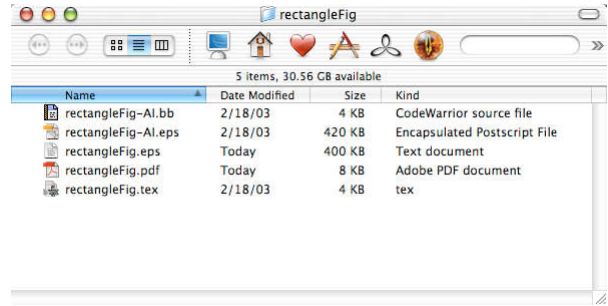


Figure 9: Content of the working folder after WARMFIGToPDF has created the final EPS and PDF files.

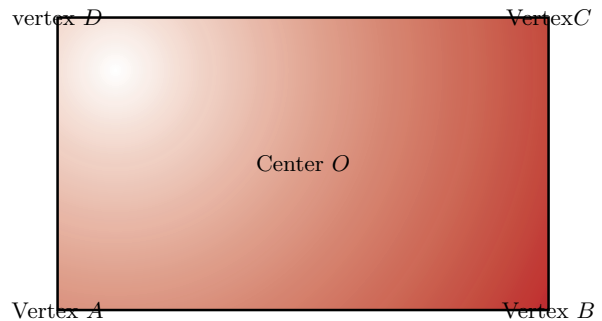


Figure 10: Appearance of the final PDF stand-alone figure.

```
% MarkedPoint:(194.314,0.500) : point(0,0) : 2 %Vertex $B$
\xyMarkedTextPoints!D(-1.50)!L( 0.00){2}

% Marked Point Number: 3
% MarkedPoint:(194.314,115.964) : point(0,0) : 3 %Vertex $C$
%\xyMarkedTextPoints!D( 0.00)!L( 0.00){3}
\xyMarkedPos{3}*!D( 1.50)!L( 0.00)\txt{Vertex $C$}

% Marked Point Number: 4
% MarkedPoint:(0.500,115.964) : point(0,0) : 4 %vertex $D$
%\xyMarkedTextPoints!D( 0.00)!L( 0.00){4}
\xyMarkedPos{4}*!D( 1.50)!L( 0.00)\txt{Vertex $D$}

% Marked Point Number: 5
% MarkedPoint:(97.407,58.232) : point(0,0) : 5 %Center $O$
%\xyMarkedTextPoints!D( 0.00)!L( 0.00){5}
\xyMarkedPos{5}*%
!D( 0.00)!L( 0.00)%
\txt{\rotatebox{45}{\textcolor{white}{Center $O$}}}

% Marked Point Number: 5
% MarkedPoint:(97.407,58.232) : point(0,0) : 5 %Center $O$
%\xyMarkedTextPoints!D( 0.00)!L( 0.00){5}
\xyMarkedPos{5}*%
!D( 0.00)!L( 0.00)%
\txt{\rotatebox{-45}{\textcolor{blue}{$O$ Center}}}

\end{xy}
```

Figure 11, along with the L^AT_EX source code used to generate it, is meant to illustrate the following points:

1. as discussed in greater detail later, the default position of the labels can be adjusted by taking advantage of the positioning directives `D` and `L`;

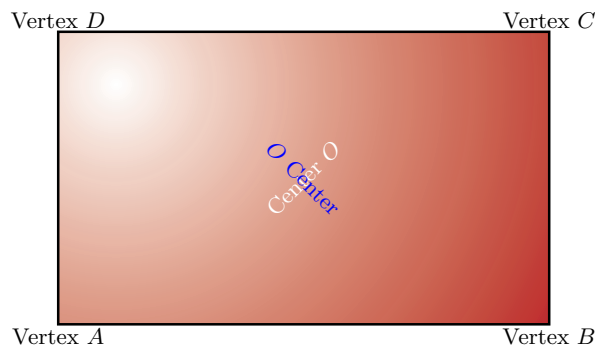


Figure 11: Modified appearance of the stand-alone `rectangleFig`.

2. one can use the command `\xyMarkedPos`, possibly the most flexible command `WARMreader` provides, to make a label out of virtually any displayable \LaTeX construct;
3. with reference to the labels originally defined as ‘VertexC’ and ‘vertex D’ and then subsequently corrected to ‘Vertex C’ and ‘Vertex D’, one can use the `\xyMarkedPos` command to correct the content of the labels directly in the `.tex` file that includes the `WARMreader` package, instead of, for example, going back to the original `Illustrator` file;
4. regardless of the command used to include a label, the same label can be included multiple times with different variations.

On nudging

Although this topic has been already discussed elsewhere (see, e.g., McKay and Moore, 1999; Moore, 2001), for the sake of completeness we will now touch upon how to accurately position labels.

The accurate placement of a label relies on understanding the exact meaning of the label’s coordinates. To this end, let us be reminded that a label, being a (\LaTeX) object, can be thought of as a *box*. Next, with reference to Fig. 12, let w and h be the

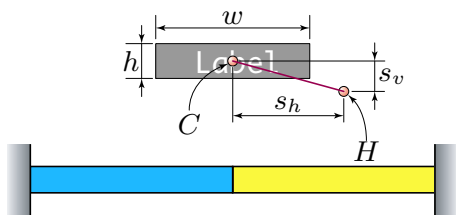


Figure 12: Elements defining the geometry of a label.

width and height of the box bounding the label, respectively. Furthermore, let C denote the center of

the label’s box and let H denote the point we will refer to as the label’s *handle*. The quantities s_h and s_v will be referred to as the horizontal and vertical shifts, respectively. Now that these geometric descriptors have been introduced, we are ready to define the meaning of the expression “the label’s coordinates”, as provided by the `.bb` file—they are the coordinates of the point H , the label’s *handle*, with respect to the lower left corner of the figure’s bounding box. As such, the label’s handle is always to be considered a *fixed point*. The quantities s_h and s_v are to be understood as the position coordinates of H with respect to C . By default, $s_h = s_v = 0$, that is, the center of the box is made to coincide with the label’s handle. `WARMreader` commands are such that one *cannot* directly specify the values of s_h and s_v in some chosen unit. In order to cause s_h and s_v to have non-zero values one actually specifies the value taken on by the ratios $2s_h/w$ and $2s_v/h$, respectively.

As an example, consider the position directives used to specify the position of the label ‘Vertex A’ shown in Fig. 11:

```
% Marked Point Number: 1
% MarkedPoint:(0.500,0.500) : point(0,0) : 1 %Vertex $A$
\xyMarkedTextPoints!D(-1.50)!L( 0.00){1}
```

The directives in question are `!D(-1.50)` and `!L(0.00)`. The first directive, namely `!D(-1.50)`, indicates that the position of H with respect to C must be 1.5 times the half-height of the box bounding the expression ‘Vertex A.’ Furthermore, the vertical position of H relative to that of C is *downward* (`!D`) and *negative* (`-1.50`). Given that the coordinates of the label’s handle are those of the lower left corner of the rectangle, as shown in Fig. 7,* these instructions result in a position of the label’s box center 1.5 times the box’s half-height *below* the box’s handle while leaving the $s_h = 0$ (since `!L(0.00)`).

As additional examples, here below we provide the directives to make one of the corners of the label’s bounding box coincide with label’s handle:

1. `!D(1.00)!L(1.00)`: the lower left corner of the label’s box is made to coincide with ‘ H ’;
2. `!D(1.00)!L(-1.00)`: the lower right corner of the label’s box is made to coincide with ‘ H ’;
3. `!D(-1.00)!L(-1.00)`: the upper right corner of the label’s box is made to coincide with ‘ H ’;
4. `!D(-1.00)!L(1.00)`: the upper left corner of the label’s box is made to coincide with ‘ H ’.

*Recall that the label’s coordinates are the coordinates of the center of the \times -symbol.

Discoveries and observations

While trying to understand the failure of our pre-Mac OS X labeling strategy and while coming up with an alternative strategy, we have run into a few interesting quirks that we would like to share.

Illustrator, PDF format, and bounding boxes

The labeling procedure we have outlined starts with the creation of a figure. As discussed earlier, when using Adobe Illustrator, this figure must be saved as an Illustrator EPS (EPS) file. Why not save the figure directly in PDF format? In other words, why not go through the labeled stand-alone figure creation process in a full PDF “environment”? The answer to this question is manifold.

First of all, most of the journals to which we submit our papers prefer to receive graphics in EPS format. In other words, it is useful to have the same figure in both EPS and PDF formats.

Second, while it is possible to save an Adobe Illustrator file in PDF format, with the current version of Illustrator the resulting figure is assigned a bounding box which, in practice, is as large as the paper media specified in the `Page Setup...` dialog. This means that when importing the PDF figure in the corresponding `.tex` document for the purpose of labeling, one is faced with the problem of determining the true bounding box of this figure.

Finally, once the labeling process is carried out, the resulting PDF file has, again, a bounding box equal to that of the page. Thus, again, one would have to find a way to determine the true bounding box of the *labeled figure*, which, in general, is not the same as that of the figure in the corresponding Illustrator file. This last problem would need to be solved even if future versions of Adobe Illustrator were to provide the possibility of saving a PDF file with a bounding box limited to that of the figure (as opposed to that of the page). However, we are not aware of any facilities (other than the `Crop Pages...` facility offered by Adobe AcrobatTM) that allows one to determine and edit bounding boxes of PDF files. Hence, at least for now, the only way for us to create a figure in PDF format with a proper bounding box is to create an EPS figure first and then distill it via Ghostscript.

To clip or not to clip While developing our labeling strategy, the calculation of the bounding box of the labeled figure was performed using the following Ghostscript command: `gs -q -dNOPAUSE -dBATCH -sDEVICE=bbbox`. The argument of the command is the PostScript file generated by typesetting the `.tex` file which contains the WARMreader commands. As strange as it may sound, in one of the Ghost-

script distributions we used, the `bbbox` device was no longer available. Not certain as to whether or not this device was going to be made available again, we decided to rely on the bounding box information contained in the EPSI file generated by running the `ps2epsi` command. What we discovered in doing so is that the bounding box computed by `ps2epsi` is often different from that computed by Ghostscript (via use of the `bbbox` device). As a matter of fact, Ghostscript computes both the bounding box and the high resolution bounding box while `ps2epsi` only computes the bounding box of the content of a PostScript file. The other behavior we observed was that whether using the bounding box information computed by `ps2epsi` or that computed by Ghostscript, the figures we were extracting from the PostScript files generated by our procedure were often clipped around the edges.

In other words, somehow the bounding box as delivered by `ps2epsi` (or, although less often, by Ghostscript) was ever so slightly too small. The problem was solved pragmatically by enlarging the `ps2epsi` generated bounding box by one point in all directions. However, we never found the time to identify the source of the `ps2epsi` and Ghostscript errors.

Summary of the figure labeling procedure

For convenience, we now summarize the steps in using WARMFIGToPDF to label figures:

1. Use Adobe Illustrator to create the figure you wish to label. Save the file, keeping in mind that:
 - (a) Illustrator 10 under Mac OS X does not yet support long file names, and
 - (b) WARMFIGToPDF will insert `-AI` into your filename.
2. Using the **Marked Objects Tool** in Illustrator, place the labels in the desired positions and change the text of those labels to the desired content using the **Text Tool**. When you are done, hide the layer containing the **Marked Objects** and then save and close the file.
3. Drag and drop the Illustrator file you have just created and marked onto WARMFIGToPDF to create a `.tex` file with the proper WARMreader commands. This file is ready to be typeset.
4. Open the resulting `.tex` file and typeset it using the ‘**T_EX + Ghostscript**’ setting in T_EXShop.
5. Iteratively adjust the positions of the **Marked Objects** by editing the `.tex` source and typesetting.

6. Once you are happy with the position of the Marked Objects, close the `.tex` file and drop the PostScript file that results from typesetting onto `WARMFIGToPDF`. This will delete all unnecessary files and will create an EPS and PDF file of your marked-up figure, each with the proper bounding box.

The Future of `WARMFIGToPDF`

We use gradients rather extensively in our Illustrator work to generate the appearance of depth. We recently discovered that there can be problems with gradients in Illustrator EPS files that have been converted to PDF using Ghostscript. Figure 13 shows a

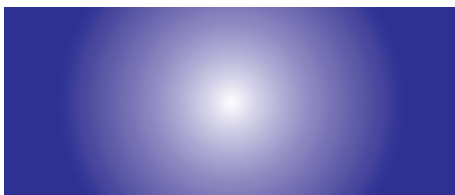


Figure 13: Smooth gradient, produced using Adobe tools.

simple rectangle that has been filled with blue and has had a simple radial gradient applied. The radial gradient starts and ends *within* the rectangle. The PDF file shown in Figure 13 was saved out of Illustrator 10 as an Adobe PDF file and then cropped using Acrobat. On the other hand, if instead we save the file as an EPS file and then use Ghostscript to convert it to PDF, we obtain the result shown in Fig. 14. Notice the incorrect color to the left and



Figure 14: Truncated gradient, erroneous result with Ghostscript.

right of where the radial gradient started and finished using the Gradient Tool. This appears to be a problem with Ghostscript and is an issue that we need corrected for the types of figures we create.

Clearly, if an image is saved out of Illustrator as a PDF file, there is no problem with the gradient. Therefore we can “work around” this problem by working with PDF files rather than with EPS files. In addition, since the future of \LaTeX seems to be heading in the direction of PDF rather

than PostScript, it is our feeling that the future of `WARMFIGToPDF` should also be in the direction of PDF. With this in mind, we have undertaken the revision of `WARMFIGToPDF` with the goal of avoiding Ghostscript to create the PDF images we wish to include in our work. In addition, since some publishers still require the submission of images for papers in EPS format, `WARMFIGToPDF` will still automatically create the appropriately marked up EPS file as part of the process. The general procedure used by the new version of `WARMFIGToPDF` will be as follows:

1. The user will create an image in Illustrator, mark it up, and save it as both an Illustrator PDF file and an Illustrator EPS file.
2. `WARMFIGToPDF` will create the `.tex` file with the marked objects embedded and ready for typesetting.
3. The user will then typeset the `.tex` file and adjust the positions of the marked objects in the usual way. In this step, the `.tex` file is typesetting using \TeX + Ghostscript and is reading in the EPS file. The outcome of this step is PostScript file as well as a PDF file that was created using Ghostscript.
4. The user will then drop the PostScript file on `WARMFIGToPDF` and then `WARMFIGToPDF` will use Ghostscript to determine the bounding box of the marked up PostScript file. This bounding box information is then used to create a corresponding final EPS file as well as a final PDF file via a typesetting process that the user never sees. The PDF is created by typesetting using `pdflatex`, reading in the original Illustrator PDF and setting the `viewport` by parsing the Illustrator PDF for the appropriate bounding box information. The bounding box used for cropping the resulting PDF file is that which was previously obtained by running the PostScript file through Ghostscript. This bounding box is incorporated into the PDF via the `\pdfpageattr` command, which has been included in the `.tex` file (which the user never sees).

We should also mention that the new version of `WARMFIGToPDF` will still process EPS files as described in this paper, but it will also have the new capability outlined above.

This new version of `WARMFIGToPDF` should be available by the time you read this.

References

- Barratt, C., M. C. Grant, and D. Carlisle. “PSfrag”. 1996. Available at <http://www.ctan.org/tex-archive/macros/latex/contrib/supported/psfrag/>.
- Costanzo, F. and G. L. Gray. “WARMFIGToPDF”. 2002. Available at <http://lpcm.esm.psu.edu/~gray/wftpdf.sit>.
- McKay, W. and R. Moore. “Convenient Labelling of Graphics, the WARMreader Way”. *TUGboat* **20**(3), 262–271, 1999. Available at <http://tug.org/TUGboat/Articles/tb20-3/tb64ross.pdf>.
- McKay, W., R. Moore, and T. Ruark. “Adobe MarkedObjects plugin for WARMreader”. *TUGboat* **22**(3), 188–196, 2001. Available at <http://tug.org/TUGboat/Articles/tb22-3/tb72moore-warm.pdf>.
- Moore, R. “What is WARMreader?” 2001. Available online at <http://www-texdev.mpce.mq.edu.au/WARM/> and <http://cds.caltech.edu/~wgm/WARM/reader2001.html>.
- Rose, K. H. “Xy-pic”. 1991. Available at <http://www.ctan.org/tex-archive/systems/mac/textures/graphics/>.

Enabling Web access to a database of calculus problems using \LaTeX , PHP and \LaTeX 2HTML

Kalle Saastamoinen, Jaakko Ketola, Tuukka Kurppa and Liisa Torikka

Laboratory of Applied Mathematics

Lappeenranta University of Technology

P.O. Box 20, FIN-53851 Lappeenranta, Finland

Internet: `kalle.saastamoinen@lut.fi`

Abstract

In this paper, we present work which we have done to provide the material used in the teaching of calculus to students at Lappeenranta University of Technology. This procedure can be divided into three separate steps: 1) creation of the \LaTeX files, 2) conversion of the base \LaTeX files to the HTML files, and 3) implementation of these HTML files in the PHP database. So far, we have stored hundreds of exercises with hints and solutions on our Web server, in addition to some theory that we felt is necessary for the students. In this paper we will also discuss the benefits and future of our functioning Web environment [1].

Introduction

At the beginning of 2002, we initiated an investigation to develop possible solutions for providing the material used in the teaching of calculus to students at Lappeenranta University of Technology. In this paper, we present the complete procedure, from creation of the \LaTeX files to a functioning Web environment [1].

We decided to write the pages in question in the \LaTeX language. We also wanted to use database and network solutions for the final version of these pages. We chose to use a network solution because the material, once available on the Internet:

1. is easy to reuse, transform and combine with other materials,
2. can be easily accessed, at least with fast connections,
3. is accessible 24 hours a day,
4. can be accessed by many users at once, for performing searches on the calculus information.

The students can then study the material that they feel is the most interesting and then hopefully begin to carry out their own research and form a view of mathematics. Therefore we have taken a constructivist approach to the learning process.

The \LaTeX files, which are entered into a database with all the necessary keywords, form the core of the solution. Based on the keywords, the database includes information on where each file is located, the specific area of mathematics, and whether the file is an exercise, solution, hint or general infor-

mation. The database alters the user's view of its contents by using the keywords that are included in the above-mentioned base \LaTeX files.

Any of our lecturers can contribute new material to the database by using a very simple interface and \LaTeX . Lecturers can also easily construct weekly tutorials and even whole examinations from the contents of the database. So far, we have stored hundreds of exercises with hints and solutions on our Web server, in addition to some theory that we felt is necessary for the students.

Our lecturers have found that this new interface saves a significant amount of time and is also flexible enough for their needs. Student satisfaction with this new material is also apparent, since the material is now always available, information is easy to find, and the material is clear.

Converting \LaTeX files to a functioning Web environment

All of our \LaTeX files are articles or exercises which are executed by a cover file. Article files include theory, exercise files include tutorials of different kinds, and cover files (`kuori.tex`) are files which call the previously mentioned document files. Cover files use `math.sty`, which includes macro definitions of mathematical formulas and all other necessary definitions. DVI files are generated by execution of a cover file. All of these \LaTeX files were originally created by a lecturer, Simo Kivelä, from Helsinki University of Technology (HUT).

Creating L^AT_EX files Base L^AT_EX files, articles and exercises include categorization which defines the final destination of the file in our Web environment. Categorization is done as follows:

- At the beginning there is `\begin{Artikkeli}` or `\begin{Tehtava}`, which defines if the file is an article or an exercise.
- `\tunniste{}`: The name of the file, for example `\tunniste{1ta179}`. (The value comes from the words Liisa, Torikka, article and the number of the article.)
- `\alue{}`: The highest categorization in our content dictionary; for example `\alue{tavdy}`, meaning ordinary differential equation.
- `\luku{}`: The next categorization area, a section; for example `\luku{2kdy}`, meaning second order differential equations.
- `\kappale{}`: The lowest categorization area, a chapter; for example `\kappale{vali}`, meaning constant linear coefficient.
- `\otsikko{}`: The title of the document, e.g. `\otsikko{20DYvaki lineaariset DY:t}`, meaning second order constant linear coefficient ODE.
- `\luonne[]{}`: Two arguments which characterize the file, for example `\luonne[Matlab]{teoria}`. Here, the first argument, ‘Matlab’, specifies that you need a computer program called Matlab to run the issued theory part; the second argument, ‘teoria’, means theory.
- `\kuvaus{}`: A short description of what the file includes.
- `\tekija{}`: The name of person who typed the file, for example `\tekija{Liisa Torikka}`.
- `\pvm{}`: The date, e.g. `\pvm{29.8.2002}`.
- `\kieli{}`: The language, with Finnish as the default.
- `\lahde{}`: The original source, for example `\lahde{LTKK/Pekka Jauhon moniste, kevät 2002}`.
- `\kayttoikeus{}`: Copyrights.
- `\lahdekoodi{}`: Source code, for example `\lahdekoodi{LaTeX}`.
- `\laitos[]{}`: The level of the material is the first argument, and the institution name is the second; for example, `\laitos[hard]{LUT}`.
- Then come keys to help database searches, for example `\avain[Calculus]{course}`; normally we use many keys to make files easy to find from our database.
- After these definitions comes the actual article or exercise.

[Add exercise or article](#)

[Modify exercise or article/Remove exercise or article](#)

File:

Figure 1: A view of the add file page.



Figure 2: A view of the starting page of calculus.

All of these files which are written are then included in the previously mentioned cover file, executed and tested.

Conversion After we have a functioning L^AT_EX file, let us say `abc.tex`, we will run a script that uses the L^AT_EX2HTML translator to install this file into our Web database. The script is written in Perl and it transfers the original L^AT_EX file into the package called `abc.tar.gz`, which includes the original file `abc.tex` along with pieces of converted HTML files.

Adding a file into the database After this, the package, here named `abc.tar.gz`, is ready to be sent into our server using a very simple form; see figure 1. Once a person pushes a button called Send a PHP-script does the following things for the file:

- it will be unpacked,
- the contents will be checked,
- the information will be added into the database and
- HTML-pages will be sent to the right directory.

All this is done based on the information included in original L^AT_EX file.

Finalization After all this is done we have our file in our Web page, where the starting page looks like figure 2.

A user can perform different searches in our Web environment. A file which was added can be found from the categorization, from its place, or by a text search on a word included in the file categorization. For example, if we pretend that our file which we added had something to do with Euler, we could try to find it by entering the word ‘Euler’



Figure 3: Searching for the word ‘Euler’.



Figure 4: Result of the search for ‘Euler’.

into the search engine as in figure 3 and the outcome of the search looks like figure 4. This works well, as long as one can think of an appropriate term for which to search.

The result from a successful search might look like figure 5.

Matematiikan virtuaalinen materiaali
Teoria: Euler-Cauchyn DY (sy033)

Huomaa:Euler-Cauchyn DY toisen kertaluvun differentiaaliyhtälöille voidaan laskea n:nnen kertaluvun DY:lle sijoittamalla yhtälöön

$$x^n y^{(n)}(x) + p_1 x^{n-1} y^{(n-1)}(x) + \dots + p_{n-1} x y'(x) + p_n y(x) = 0 \quad (1)$$

$y(x) = x^m$.

Esimerkki:Kolmannen kertaluvun Euler-Cauchy differentiaaliyhtälön

$$x^3 y'''(x) + p_1 x^2 y''(x) + p_2 x y'(x) + p_3 y(x) = 0 \quad (2)$$

ratkaisuksi saadaan sijoittamalla $y(x) = x^m$ seuraavaa: $x^m (m(m-1)(m-2) + p_1 m(m-1) + p_2 m + p_3) = 0$, koska $x^m = 0$ vain arvolla $x = 0$ voidaan se jättää huomiotta, eli saadaan yhtälö

$$m(m-1)(m-2) + p_1 m(m-1) + p_2 m + p_3 = 0 \Leftrightarrow$$

$$m^3 + (p_1 - 3)m^2 + (p_2 - p_1 + 2)m + p_3 = 0, \quad (3)$$

josta ratkaisemalla juuret saadaan kantaratkaisut samaan tapaan kuin edellä.

Figure 5: Example result document.

Conclusions and future

We have received many positive and very few negative comments from these pages that we have created. Our pages have been used in our normal basic courses and both lecturers and students have taken them as their own. We have also done both qualitative and quantitative usability testing for these pages and the results have been very promising.

In the future, we plan to automate the way in which text will be captured in the pages of, for example, examinations. This will make our pages more usable in the context of distance education, among others. We will also continue developing these pages with animations and other features.

References

- [1] Kalle Saastamoinen, Jaakko Ketola, Tuukka Kurppa and Liisa Torikka (2002). “Calculus Web pages”, <http://www.it.lut.fi/mat/virtuaali/matb>

MathML formatting with \TeX rules, \TeX fonts, and \TeX quality*

Luca Padovani

University of Bologna, Department of Computer Science

Mura Anteo Zamboni, 7

40127 Bologna

Italy

lpadovan@cs.unibo.it

<http://www.cs.unibo.it/~lpadovan/>

Abstract

In this paper we investigate the architecture of a MathML formatting engine based on an abstraction of the \TeX box primitives. This engine is carefully designed so that the \TeX -dependent formatting rules are isolated from the independent ones and is capable of achieving \TeX -comparable output quality when used in conjunction with \TeX fonts. We show how the formatting rules presented in Appendix G of the \TeX book can be easily adapted for MathML formatting, and how the semantically-rich MathML markup simplifies the rules themselves.

1 Introduction

The Mathematical Markup Language (MathML [4]) is an XML application for encoding mathematical formulae. It has two distinct sets of tags: the *presentation* tags which are used to encode what a formula looks like, and the *content* tags which are used to encode the “meaning” of a formula. The \TeX macros for math typesetting either represent operators or identifiers that cannot be typed directly on the keyboard, or they implement the most common layout schemata for the mathematics. In the context of MathML presentation, the \TeX macros of the first kind correspond to Unicode [1, 2] characters, hence they do not have dedicated markup. Macros of the second kind correspond very closely to MathML presentation tags, as Table 1 shows. We can thus say that \TeX and MathML presentation encode mathematical formulae at the same level.

Indeed, the issue of formatting MathML documents using \TeX can be addressed simply by providing a transformation from the MathML markup to the \TeX / \LaTeX markup,¹ and using a \TeX implementation for the actual rendering. This approach, however, is far from being all-satisfactory. For example, the whole rendering process is only feasible if no interactivity is required, that is, if the rendering can be performed off-line as a batch procedure

* This work has been supported by the European Project IST-2001-33562 MoWGLI.

¹ Examples of such translation tools are the one developed at the Ontario Research Centre for Computer Algebra [8], and the XSLT stylesheets by Vasil I. Yaroshevich <http://www.raleigh.ru/MathML/mmltex/>.

Layout Schemata	\TeX	MathML
Identifiers	<code>a,x,\sin,...</code>	<code>mi</code>
Numbers	<code>0,1,2,...</code>	<code>mn</code>
Operators	<code>+,(\,\oint,...</code>	<code>mo</code>
Grouping	<code>{ }</code>	<code>mrow</code>
Fractions	<code>\over, \atop</code>	<code>mfrac</code>
Radicals	<code>\sqrt, \root \of</code>	<code>msqrt, mroot</code>
Scripts	<code>-, ^</code>	<code>msub, msup, msubsup, mmultiscripts</code>
Stacked expressions, lines and braces above and below formulae	<code>\buildrel, \underline, \overline, \underbrace, \overbrace</code>	<code>munder, mover, munderover</code>
Matrices, tables	<code>\matrix, \cases</code>	<code>mtable</code>

Table 1: Correspondence of \TeX and MathML layout schemata.

on a static medium (the paper, an image). Also, requiring the large complex \TeX system whose overall purpose goes far beyond math formatting is probably unreasonable. Furthermore, not every MathML document can be faithfully converted into the corresponding \TeX markup—the conversion is more a good approximation than an accurate rendering.

An alternative approach is to implement a self-contained MathML formatter that is also capable of using \TeX fonts. However, the use of \TeX fonts at this level requires a good understanding of the conventions adopted for the metrics of the glyphs they provide and the extra parameters they have. The \TeX system relies on this knowledge in order to achieve a very high formatting quality. On the other side, MathML is a language for publishing and communicating mathematics on the Web, hence its

rather high-level formatting semantics do not make any special assumptions about the graphic capabilities of the environment in which it is rendered, e.g., whether it is a system for quality typesetting on the paper, a regular computer, or a hand-held device.

In this paper we investigate how to carefully design a MathML formatter so as to separate what is $\text{T}_{\text{E}}\text{X}$ -specific from what is common in every environment. We also show how the semantically rich MathML relieves the author from explicitly tweaking the markup, which is a potentially dangerous operation in the context of MathML and, more generally, of documents to be published on the Web, as there is no guarantee that the tweak will work successfully on a different formatter from the one the author is using.

We have chosen to proceed with our investigation by giving illustrations instead of exhaustive explanations, leaving to the reader the generalization of the cases considered here. Consequently, some knowledge of MathML and its basic concepts is highly recommended. We have assumed that the reader is also familiar with the $\text{T}_{\text{E}}\text{X}$ formatting rules for mathematics and other $\text{T}_{\text{E}}\text{X}$ concepts described in Appendix G of *The $\text{T}_{\text{E}}\text{X}$ book* [5].

2 MathML formatting

By “formatting a formula” we mean the process that transforms the formula encoded in some markup language into a lower-level representation that conveys information about the needed glyphs, their size and their relative position.

Math formatting is always done with respect to a *formatting context* which defines, at least, (a) the (relative) font size at which formatting is occurring; (b) the scripting level—that is the number of nested scripts at which formatting is occurring; (c) whether the formula is formatted at display level (in a paragraph of its own) or inline.

We define the MathML formatting context as a structure with a number of named *fields*. We use the “dot notation” to select a particular field in a formatting context, thus we will write $C.size$ to denote the value of the field *size* in the context C . Table 2 shows the main fields of a MathML formatting context.² A $\text{T}_{\text{E}}\text{X}$ nician will recognize that the MathML formatting context is just a generalization of the $\text{T}_{\text{E}}\text{X}$ notion of “style”. In fact, the development of MathML has been influenced by $\text{T}_{\text{E}}\text{X}$ in many ways.

² There can be many different definitions of a MathML formatting context. The one given here is not complete, but it suffices for the purposes of the paper.

Name	Default	Description
<i>size</i>	inherited	Font size
<i>scriptLevel</i>	0	Number of nested scripts
<i>minSize</i>	8 pt	minimum font size that a script can be reduced to
<i>displayStyle</i>	inherited	true if formulae must be formatted in display mode
<i>sizeMult</i>	0.71	Amount by which the font size is multiplied when the script level is increased by 1
<i>stretchWidth</i>	undefined	Horizontal extent the operator is asked to stretch
<i>stretchHeight</i>	undefined	Vertical extent (above the baseline) the operator is asked to stretch
<i>stretchDepth</i>	undefined	Vertical extent (below the baseline) the operator is asked to stretch

Table 2: Properties of the MathML formatting context.

Because of the structured nature of MathML documents, MathML formatting can be expressed as a recursive function. Say we have a formula encoded in MathML like $\langle t \rangle X_1 \cdots X_n \langle /t \rangle$ where t is the *type* of the root MathML element (ranging over `mrow`, `mfrac`, and so on) and the X_i its children, then the formatting of the formula in a given formatting context C , notation $\llbracket \langle t \rangle X_1 \cdots X_n \langle /t \rangle \rrbracket_C$, can be expressed as a proper combination, or rearrangement, of the formatted children X_i each in its own formatting context C_i :

$$\llbracket \langle t \rangle X_1 \cdots X_n \langle /t \rangle \rrbracket_C = \mathbf{f}_t(C', \llbracket X_1 \rrbracket_{C_1}, \dots, \llbracket X_n \rrbracket_{C_n})$$

Similarly, formatting of MathML token elements can also be expressed as a function of the current formatting context and of the Unicode characters c_1, \dots, c_n that the token is made of:

$$\llbracket \langle t \rangle c_1 \cdots c_n \langle /t \rangle \rrbracket_C = \mathbf{f}_t(C', c_1 \cdots c_n)$$

In both these cases, the type t of the element being formatted and the value of attributes may affect the formatting context C and change it into a different context C' . For instance, formatting of a token element that explicitly sets the current font size (`mathsize` attribute) will format its characters in a context C' in which the field $C'.size$ has been updated accordingly. The complete set of rules for updating the formatting context C are described in detail within the MathML specification, and they basically follow the rules for style changes in $\text{T}_{\text{E}}\text{X}$ (a concrete example will be given later).

As described in Appendix G of *The $\text{T}_{\text{E}}\text{X}$ book*, $\text{T}_{\text{E}}\text{X}$ scans and processes a logical representation of a formula (a *math list*) consisting of items, converting it into a physical representation (a horizontal list) made up of regular boxes. Items in the math list can be of different types, and in many cases they directly correspond to basic math layout schemata

(like Rad atoms for radicals, Acc atoms for accents, and generalized fractions). For every item type, Appendix G defines one or more rules describing in detail how the item is converted in one or more boxes, ultimately defining how \TeX formats mathematical formulae. Given the strict correspondence between \TeX commands for math typesetting (hence math list items) and MathML presentation elements, the same set of rules can be easily adapted for the definition of the \mathbf{f}_t functions.

In the following sections we consider in some more detail the formatting rules for the main categories of MathML elements and show how they map on \TeX rules.

2.1 Groups

Math lists result from typesetting in math mode. The content of a math list is typically formatted on a single horizontal line, with all the items aligned on their baseline. In this respect a math list is close to the MathML `mrow` element. However, `mrow` is also fundamental for the following purposes:

- all stretchable operators within the same `mrow` element should vertically stretch so as to have the same height plus depth, unless they are constrained in some way either by the markup or by some limitations of the environment;
- `mrow` is the main MathML element enabling automatic line-breaking of long formulae, when they exceed the available horizontal space for formatting. Conversely, \TeX grouping operators `{ }` freeze a sub-formula preventing any line-break in it.

Stretching of operators is done by looking at the bounding box of the child elements, once they have been formatted, and then passing adequate information through the formatting context, the idea being that operators have to take care of stretching themselves. This generalized treatment of stretchy operators relieves all the other MathML elements from taking into account vertical stretching rules. In particular, rule **13** (large operators) only applies when formatting stretchable operators (see the discussion on tokens that follows), and rule **15e** (fractions with delimiters) is never necessary because if a fraction must have delimiters, they must be explicitly encoded inside an `mrow` element along with the delimited fraction.

Automatic line-breaking is only affected by the bounding box of the formatted child elements, possibly requiring re-formatting of all or some of them. This can be considered a higher-level formatting issue which does not involve low-level \TeX rules.

2.2 Tokens

Tokens are the basic building blocks of every mathematical formula. In MathML, tokens are the only elements allowed to have actual text as content. The most important token types are `mi` for identifiers, `mn` for numbers, and `mo` for operators. The first two correspond roughly to Ord atoms in a math list, whereas the last one is refined in \TeX into different atoms, Op, Bin, Rel, Open, Close, or Punct. The most remarkable difference is that in \TeX those atoms must be made of exactly one character, whereas MathML tokens are made of arbitrary Unicode strings. From the point of view of formatting, though, the more general scheme adopted by MathML does not pose any additional problem, and it actually simplifies the encoding of non-strictly-mathematical documents in which identifiers and operators whose name is longer than one character are frequent.

The fine-grained classification of operators is needed in \TeX for mainly two reasons: (1) computing the right amount of space around operators; (2) helping the automatic line-breaking algorithm with hints on where the formula can be broken. In particular there is a distinction between unary operators (Op) and binary operators (Bin) as they typically have different spacing rules. In properly grouped MathML markup there is no need for distinguishing unary (prefix and postfix) from binary (infix) operators, as their *form* can be inferred from their position in the enclosing `mrow` element.

Rule **14** does not apply any more in general for if two characters are marked up in different tokens they should never be kerned or merged into a ligature. This behavior is rather part of the formatting semantics of the token itself.

Operators are by far the most complicated tokens to format, for they may have to stretch vertically or horizontally and are affected by a many MathML attributes. Assuming that stretching information is propagated in the formatting context (*C.stretchWidth*, *C.stretchHeight*, *C.stretchDepth*), operators can use rule **19** for determining the exact extent they should span, and rule **13** when they must be formatted in the large form.

2.3 Accents

MathML provides multiple ways of encoding an “accent”, depending on whether the accent is meant to be syntactic or semantic. A syntactic accent is simply part of a name, it has no mathematical meaning. Although it is very rare in mathematics to have identifiers with accents, the use of explicit markup

allows MathML to disambiguate the two cases. Syntactic accents are used following the Unicode rules for combining characters. Hence, a MathML identifier like

```
<mi> a&#x0307; </mi>
```

is typeset as \acute{a} (the Unicode character U+0307 represents the “combining dot above” $\acute{\circ}$).

A semantic accent usually denotes an operation, like first-order derivative in case of the “combining dot above”. As such it is marked up as

```
<mover accent="true">
  <mi> a </mi>
  <mo> &#x0307; </mo>
</mover>
```

even though the formatted result is likely to be the same as in the previous case. Note how the accent is explicitly marked up as an operator inside an `mo` element. Rule 12 (accents) can handle both accents combined with a single character and wide accents combined with arbitrary subexpressions.

Although they cannot be considered proper accents, horizontal lines extending above or under a formula are treated in MathML uniformly with all the other operators. In particular, the MathML markup corresponding to `\underline{a}` is

```
<munder>
  <mi> a </mi>
  <mo> &#x0332; </mo>
</munder>
```

where U+0332 is Unicode combining horizontal line below. The horizontal stretching rules of MathML operators require the U+0332 character to stretch to the width of the base subformula. The symmetric situation (`\overline`) is handled similarly, except that the Unicode character to be used is U+0305. These two cases are handled by rules 9 and 10.

2.4 Radicals

Formatting of root symbols deriving from `msqrt` and `mroot` elements is handled by rule 11 concerning Rad atoms.

2.5 Fractions

The `mfrac` element governs encoding of fractions. In \TeX , alignment of the numerator and the denominator can be determined by the use of `\hfill`. In MathML, it is affected by the value of the attributes `numalign` and `denomalign`. \TeX rules for formatting fractions are those from 15 through 15d. \TeX provides different commands for typesetting vertical material in a fraction-like manner, depending on whether one wants a fraction bar (`\over`, `\above`)

or not (`\atop`). MathML handles all such cases by setting the `linethickness` attribute.

2.6 Scripts

Scripts are handled in MathML by the elements `msub`, `msup`, `msubsup`, `mmultiscripts` and sometimes `munder`, `mover`, and `munderover`. The reason why `munder`, `mover`, and `munderover` elements have to do with scripting is that they implement a mechanism similar to that determined by the `\limits`, `\nolimits`, `\displaylimits` commands in \TeX .

\TeX rules governing the placement of scripts are those from 18 through 18f. Note that scripts in \TeX are represented as possibly empty fields on any atom, whereas they are uniformly marked up with elements in MathML.

2.7 Tables

MathML markup for tables does not involve any specific font dependency as it is basically a higher-level formatting problem, compared to formatting of the other elements.

3 Dealing with \TeX dependencies

If we define an *environment* as the combination of available fonts, graphic capabilities of the output medium and user requirements, \TeX rules for formatting make strong assumptions that are hard to generalize to environments other than typesetting math on the paper using a family of \TeX fonts [7]. We can summarize \TeX dependencies as follows:

- non-standard font metrics (such as, the thickness of the horizontal line in a radical is computed from the height of the radical symbol; the use of width and italic correction for the placement of scripts);
- non-standard kerning information (like using `\skewchar` for determining the horizontal displacement of accents);
- font-related quantities (the parameters σ_i and ξ_j in Appendix G) whose value cannot be otherwise inferred or computed in general;
- use of boxes of “black ink” (rules) for drawing fraction lines, root lines, joining segments in horizontal braces, formatting of Unicode characters U+0305 and U+0332;
- built-in \TeX constants (see `\delimiterfactor` and `\delimitershortfall` for stretchy operators, `\nulldelimiterspace`).

It is not feasible for any environment in which we might like to format MathML markup to provide the same set of parameters, or to address specific formatting issues the same way \TeX does. Nor is it

feasible to assume that the set of parameters we have considered is a superset of all the possible parameters that may ever affect formatting of mathematical formulae. Nonetheless the list of operations involving dependencies (formatting of tokens, radicals, scripts, fractions, accents) will be exactly the same in systems other than $\text{T}_{\text{E}}\text{X}$. This amounts to saying that the definition of each \mathbf{f}_t can be split up into two components: a \mathbf{g}_t component that does not depend on anything that is $\text{T}_{\text{E}}\text{X}$ specific, and an \mathbf{h}_t component that is strictly dependent on $\text{T}_{\text{E}}\text{X}$. By doing so we automatically identify two main parts of the MathML formatter: the set of functions \mathbf{g}_t which defines the *formatting engine* — that part which takes care of all $\text{T}_{\text{E}}\text{X}$ -independent aspects of the formatting; the set of functions \mathbf{h}_t which defines the *$\text{T}_{\text{E}}\text{X}$ device for mathematics* — that part which deals with anything which is (or may be) dependent on $\text{T}_{\text{E}}\text{X}$ fonts or $\text{T}_{\text{E}}\text{X}$ formatting rules.

Since the set of \mathbf{f}_t is finite and agreed upon *a priori* (the set of math layout schemata is fixed and relatively stable, being the result of centuries of slow evolution and convergence to modern notation), the interface to the $\text{T}_{\text{E}}\text{X}$ -dependent parts is also fixed. This way when the environment changes — say when we move to a different family of fonts that does not adopt the $\text{T}_{\text{E}}\text{X}$ conventions — we need only to re-instantiate the \mathbf{h}_t 's with those that are customized to this new environment but with the same agreed interface, still sharing the same set of \mathbf{g}_t 's.

Figure 1 summarizes graphically the entities involved in formatting a fraction element, in particular the separation of the $\mathbf{g}_{\text{mfrac}}$ and $\mathbf{h}_{\text{mfrac}}$ components.

Of course the proposed modular organization of the formatter makes sense only if the two sets of functions are both performing non-empty and non-trivial tasks. But it is clear from Appendix G of *The $\text{T}_{\text{E}}\text{X}$ book* [5] and the intricacies of math formatting detailed therein that the $\text{T}_{\text{E}}\text{X}$ -dependent part is non-trivial. As for the formatting engine part, it has the following list of non-trivial responsibilities (that we will not elaborate further here, for the sake of brevity): the construction of a data structure (typically a tree) which is suitable for formatting purposes; the implementation of the MathML mechanism for attribute inheritance and evaluation; the computation of updated formatting contexts; the algorithm for automatic line-breaking of long formulae; the algorithm for table layout.

4 An area model for MathML

The formatting functions \mathbf{f}_t take objects as arguments and produce a new object as a result. Such objects, which we will call *areas* from now on, are a

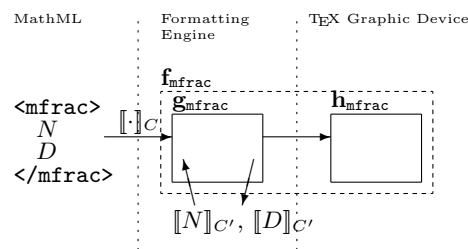


Figure 1: Modularization of the formatting function for the `mfrac` element.

low-level representation of formatted formulae, and constitute what we define as the *area model* of the $\text{T}_{\text{E}}\text{X}$ device for mathematics (or, in general, any other instance of it).

From the point of view of the formatting engine, we note that the only thing that matters is the ability to compute an area's bounding box, which is essential for updating the context with information about stretchable operators, for the automatic table layout algorithm and for the line-breaking algorithm. We summarize this by saying that areas are *opaque* to the formatting engine.

From the point of view of the \mathbf{h}_t functions, however, areas must convey more information. Many of the formatting rules presented in Appendix G have dependencies on the kind of areas being combined (for example, whether they are simple glyphs or arbitrary formatted sub-formulae), and also on the actual shape of the glyphs the areas are made of. We summarize this by saying that areas must be *transparent* to the $\text{T}_{\text{E}}\text{X}$ device for mathematics.

This neat separation of views (opaque vs. transparent) of the area model is crucial in the design of a modular architecture in that it relegates areas to the $\text{T}_{\text{E}}\text{X}$ -dependent part of the formatter, the only requirement for them being that of exporting a very limited set of operations (namely, the computation of the bounding box).

In the specific case of formatting using $\text{T}_{\text{E}}\text{X}$ rules and $\text{T}_{\text{E}}\text{X}$ fonts, the area model can be naturally synthesized as a subset of $\text{T}_{\text{E}}\text{X}$ boxes. Table 3 introduces an abstract notation for the most common area types needed for math formatting, along with their corresponding $\text{T}_{\text{E}}\text{X}$ box or primitive command that achieves the same (or a similar) formatting. The reason for not using $\text{T}_{\text{E}}\text{X}$ primitive boxes directly is that the very same abstract model can be implemented in a sensitive way depending on the environment, whereas the $\text{T}_{\text{E}}\text{X}$ box model is obviously tied to $\text{T}_{\text{E}}\text{X}$. For example, when formatting a MathML document that is meant to be interactive, the resulting areas may carry information about se-

Area	TeX box	Description
$G[\cdot]$		Glyph
K_n	<code>\kern</code>	Kern with value n ; the kern is horizontal or vertical depending on the container it is in
F	<code>\hfill</code> <code>\vfill</code>	Filler area
R_n	<code>\hrule</code> <code>\vrule</code>	Filler rule of thickness n
$S_n[\alpha]$	<code>\raisebox</code> <code>\lowerbox</code>	Shift α 's baseline by n
$H[\alpha_1, \dots, \alpha_n]$	<code>\hbox</code>	Horizontal group of areas $\alpha_1, \dots, \alpha_n$
$V_k[\alpha_1, \dots, \alpha_n]$	<code>\vbox</code>	Vertical group of areas $\alpha_1, \dots, \alpha_n$, where α_k is the reference area that determines the baseline

Table 3: Area types and their rendering semantics.

lections, or may have backward pointers towards the MathML elements that generated them. Such information is clearly unnecessary when formatting a static document for printing.

4.1 Example: mfrac formatting

We conclude this section by showing a complete example of formatting function, along with the resulting area object.

Let's suppose we are to format a subformula of the form `<mfrac> N D </mfrac>` representing the fraction

$$\frac{N}{D}$$

where we may assume, for the sake of generality, that N and D are metavariables standing for arbitrary subexpressions rather than actual identifiers. Then we have

$$\llbracket \langle \text{mfrac} \rangle N D \langle \text{mfrac} \rangle \rrbracket_C = \mathbf{f}_{\text{mfrac}}(C, \alpha, \beta)$$

where

$$\begin{aligned} \alpha &= \llbracket N \rrbracket_{C'} \\ \beta &= \llbracket D \rrbracket_{C'} \end{aligned}$$

and C' is such that if $C.\text{displayStyle} = \text{true}$ then $C'.\text{displayStyle} = \text{false}$ and all the other fields are the same as in C , whereas if $C.\text{displayStyle} = \text{false}$ then $C'.\text{scriptLevel} = C.\text{scriptLevel} + 1$, $C'.\text{size} = \max(C.\text{minSize}, C.\text{size} \times C.\text{sizeMult})$ and the other fields are the same as in C .

Depending on the value of the `numalign` and `denomalign` attributes the numerator and the denominator may be aligned to the left, to the right, or may be centered (this is the default). Assuming they are centered, $\mathbf{f}_{\text{mfrac}}$ is defined as

$$\mathbf{f}_{\text{mfrac}}(C, \alpha, \beta)$$

$$\begin{aligned} &= \mathbf{h}_{\text{mfrac}}(C, H[F, \alpha, F], H[F, \beta, F]) \\ &= S_a[V_3[H[F, \beta, F], K_d, R_h, K_n, H[F, \alpha, F]]] \end{aligned}$$

which is to be read as follows: the fraction is made of (from bottom to top) the centered denominator $H[F, \beta, F]$, a kern d , the horizontal bar of thickness h , a kern n , the centered numerator $H[F, \alpha, F]$. The reference point of the whole vertical area V coincides with the reference point of its third child (the horizontal bar). Then, the whole fraction is shifted up by a so that the horizontal bar is aligned with the axis of the expression. The quantities a , d , h , and n are computed by $\mathbf{h}_{\text{mfrac}}$ following rules 15–15d.

Note how the process is clearly split into a TeX-independent part (computation of updated formatting contexts, alignment of numerator and denominator) and a TeX-dependent one (exact positioning of the subparts).

5 How semantics helps formatting

When we speak of “semantics” in the context of MathML presentation markup, we refer not only to the presentation tags but also to the following characteristics:

- The use of explicitly encoded, although invisible, operators such as “invisible multiplication” and “function application”;
- The use of a more rigid encoding of mathematical formulae, in particular proper grouping rules. In practice this amounts to allowing within the same `mrow` element at most one kind of operator, with only a few exceptions (pluses and minuses within the same `mrow` are not considered a violation of the proper grouping rule);
- The use of a dictionary customizing the basic properties of known operators, such as their stretchability, whether they are delimiters, fences, or other kinds of operators, the amount of space that should normally be around them.

The presence of semantics in the MathML encoding of a mathematical expression has often been associated with the ability to reconstruct a more semantically-oriented representation of the same expression. However, such information can also be exploited for formatting purposes as it allows the formatter to apply context-sensitive rules that are logically related to the semantics of the entity being formatted. This eliminates the need for strange formatting rules, special cases, and other oddities that abound in the formatting rules of TeX markup, and it also permits a more general and effective formatting than TeX allows. In the sections that follow we consider three specific examples.

5.1 Invisible operators are explicit

The use of explicit markup for invisible operators has a noticeable impact on the formatter and the way it operates. As a concrete example let's consider a typical trigonometric function, say sine, for which an explicit T_EX macro has been designed. Depending on the author's taste or needs, there are two different ways to typeset the sine of x :

$$\sin x \quad \text{or} \quad \sin(x)$$

both of which are encoded using the `\sin` macro. A close look at the formatted formulae will reveal that in $\sin x$ there is a little space between the function name and the argument, whereas there is no space in $\sin(x)$. Although this is indeed very natural, and yields a nicely formatted formula, the hidden mechanism that makes this happen is anything but straightforward. The definition of the `\sin` macro is something like the following:

```
\def\sin{\mathop{\rm sin}}
```

The key point is the definition of `\sin` as an operator. By T_EX spacing rules, an operator (Op atom) followed by a variable (Ord atom) gets some space after it, but the same operator, when followed by a delimiter (Open atom), gets no extra space.

In MathML the same expressions would be encoded as

```
<mrow>
  <mi>sin</mi>
  <mo>&ApplyFunction;</mo>
  <mi>x</mi>
</mrow>
```

or

```
<mrow>
  <mi>sin</mi>
  <mo>&ApplyFunction;</mo>
  <mrow>
    <mo>(</mo> <mi>x</mi> <mo>)</mo>
  </mrow>
</mrow>
```

respectively. The fact that we have an explicit operator, function application, between the function and its argument allows the formatter to do something context-sensitive along the following lines: whenever formatting of `ApplyFunction` operator is requested, look at the next element. If it is an `mrow` whose first child is a delimiter then render `ApplyFunction` as zero-width space. Otherwise render it as some suitable constant space. The difference with T_EX is that this behavior is explicitly associated with formatting of the `ApplyFunction` character, and not part of a more general (but also less clear) scheme for spacing math atoms. Finally, note that T_EX's mechanism

relies on the `\sin` function being defined as an “operator,” whereas `sin` is correctly marked up as an identifier (for the sine function) in MathML markup.

Without going into the same level of detail, the reader can easily verify that a similar thing happens with invisible multiplication, which is in fact invisible in T_EX markup, whereas it is explicitly encoded as the `InvisibleTimes` operator in MathML. In the case of adjacent fractions, just to mention one specific case, null delimiters with non-null width are accurately placed so that the fraction bars do not join together. Correct spacing between math Ord atoms is also ensured by an accurate use of font metric information, italic corrections in particular, which guarantees a very high quality of the formatted formula, but contributes in making T_EX fonts and T_EX formatting rules mutually dependent.

5.2 Opening and closing delimiters

T_EX distinguishes delimiters as opening and closing depending on their name. `(`, `[`, `{` are examples of opening delimiters, `]`, `]`, `}` are examples of closing delimiters. The distinction is carried out at the level of atoms, where the delimiters are represented by either `Open` or `Close` atoms.

In a properly grouped MathML expression the distinction is made depending on the *position* of the operator rather than its name. In fact, a properly grouped expression must be an `mrow` element whose first and last children are the opening and closing delimiter, respectively, and the middle child is the body of the expression.

The operator dictionary that determines the default value of operator properties, spacing in particular, is addressed by both the operator's name and its form (one of prefix, infix, or postfix). Hence, the use of properly grouped markup combined with an operator dictionary provides for greater generality and flexibility in a formatter for MathML markup. As a notable side effect, it also disambiguates those cases in which the opening and closing delimiters are equal (think of `|` or `||`), which must be carefully treated in T_EX markup in order to get the spacing right.³

5.3 The strange case of the solidus symbol

Inline division is typically represented by the `/` symbol placed very close to its operands. T_EX has the oddity that the `/` symbol is *not* treated as an op-

³ T_EX also provides for a `\mid` operator which should be used for `|` when it stands as the separator in comprehensive notation for sets. This case is also simplified in MathML markup, as in properly grouped markup the `|` operator would be correctly treated as an “infix” operator.

erator Op, but rather as an ordinary symbol Ord, so that it gets no space around during the second phase of formula formatting (rule 20). The visual effect is that of rendering as $1/2$ rather than $1 / 2$. In MathML formatting this trick is no longer necessary, and the $/$ can be naturally encoded as `<mo>/</mo>`, for the spacing around it can be controlled by the `lspace` and `rspace` attributes accepted by every `mo` element (their default values can be specified in the MathML operator dictionary).

6 Conclusions

As MathML is relatively similar to \TeX , at least at some abstract level, most of the \TeX formatting rules can be easily adapted and used for MathML formatting when \TeX fonts are available. However, doing so in a modular and adaptable way, without necessarily committing to \TeX fonts, is a more delicate problem.

In this paper we have surveyed a number of issues and their possible solutions, ultimately depicting the architecture of a formatter for MathML markup which is capable of exploiting all of \TeX 's finest rules for math typesetting without being tied at the same time to the \TeX fonts. Thanks to the structurally and semantically rich MathML markup, the formatter also succeeds in cases that cannot be handled in a general way by the \TeX formatting rules without the help of the author. This aspect is particularly relevant because, if the desired rendering is not achieved by the \TeX rules, the author can tweak a \TeX formula and still be sure that the formula will be rendered the same way on every system running \TeX . On the other hand, since MathML formatters are not tied to a set of fixed formatting rules, tweaking the MathML markup can potentially compromise effective rendering of the formula.

The problem of being adaptable to the formatting environment is not just a matter of recognizing the available fonts and achieving the finest formatting with those fonts. \TeX formatting rules assume that the formulae will eventually be printed on paper, or at least displayed on a high resolution screen. There are contexts in which it is more convenient to display symbols differently, as to improve editing, interaction, or readability, especially in low-resolution display such as those used in hand-held devices.

The techniques that we have described in the paper have been successfully put into practice in two prototypes, a MathML formatting engine for a recognizer of hand-written mathematics in hand-held devices at the Ontario Research Centre for Com-

puter Algebra [9], and the `gtkmathview` widget⁴ at the University of Bologna. The latter tool, in particular, has been adopted by John Wiley & Sons, Inc., the publisher, for rendering mathematical formulae encoded in MathML markup while achieving a quality comparable to that of \TeX and using several families of \TeX fonts.

In a broader perspective, the architecture we have designed and implemented allows applications to exploit context dependencies, instead of avoiding them. As the development of the two prototypes has shown, the efforts required for implementing the techniques are negligible when compared to the potential benefits.

References

- [1] “The Unicode Standard”, Version 3.0, Addison Wesley, 2000.
- [2] “Unicode Standard Annex #28, Unicode 3.2”, 2002. <http://www.unicode.org/unicode/reports/tr28/>
- [3] “Extensible Markup Language (XML) Specification”, Version 1.0, W3C Recommendation, 10 February 1998. <http://www.w3.org/TR/REC-xml>
- [4] “Mathematical Markup Language (MathML) Version 2.0”, W3C Recommendation, 21 February 2001. <http://www.w3.org/TR/MathML2/>
- [5] D. E. Knuth, “The \TeX book”, Addison-Wesley, Reading, MA, 1998.
- [6] D. E. Knuth, “The METAFONTbook”, Addison-Wesley, Reading, MA, 1994.
- [7] U. Vieth, “Math typesetting in \TeX : The good, the bad, the ugly”, Proceedings of the Euro \TeX Conference, 2001, The Netherlands.
- [8] E. Smirnova, S. M. Watt, “MathML to \TeX Conversion: Conserving high-level semantics”, MathML International Conference, 2002. <http://www.mathmlconference.org/2002/presentations/smirnova/index.html>
- [9] L. Padovani, “A Standalone Rendering Engine for MathML”, MathML International Conference, Chicago, IL, 2002. <http://www.mathmlconference.org/2002/presentations/padovani/>
- [10] L. Padovani, “MathML Formatting”, Ph.D. Thesis, Technical Report UBLCS-2003-03, Dept. Computer Science, Bologna, Italy, 2003.

⁴ See <http://helm.cs.unibo.it/mml-widget/> and [10].

Appendix

To conclude, here is an actual example of an equation in MathML form, and the corresponding output generated by g_tkmathview.

```
<math display="block"
  xmlns="http://www.w3.org/1998/Math/MathML">
  <mrow>
    <mo>&#x222b;</mo>
    <mo>&#x2061;</mo>
    <mfrac>
      <mrow>
        <mrow>
          <mi>a</mi>
          <mo>&#x2062;</mo>
          <mi>x</mi>
        </mrow>
        <mo>+</mo>
        <mi>b</mi>
      </mrow>
      <mrow>
        <msup>
          <mi>x</mi>
          <mn>2</mn>
        </msup>
        <mo>+</mo>
        <mrow>
          <mi>p</mi>
          <mo>&#x2062;</mo>
          <mi>x</mi>
        </mrow>
        <mo>+</mo>
        <mi>q</mi>
      </mrow>
    </mfrac>
  </mrow>
  <mo mathvariant="italic">d</mo>
  <mi>x</mi>
  <mo>=</mo>
  <mrow>
    <mrow>
      <mfrac>
        <mi>a</mi>
        <mn>2</mn>
      </mfrac>
      <mo>&#x2062;</mo>
    </mrow>
    <mrow>
      <mi>ln</mi>
      <mo>&#x2061;</mo>
    </mrow>
    <mo>( </mo>
    <mrow>
      <msup>
        <mi>x</mi>
        <mn>2</mn>
      </msup>
      <mo>+</mo>
      <mrow>
        <mi>p</mi>
        <mo>&#x2062;</mo>
        <mi>x</mi>
      </mrow>
      <mo>+</mo>
    </mrow>
  </mrow>
```

```
<mi>q</mi>
</mrow>
<mo>></mo>
</mrow>
</mrow>
</mrow>
<mo>+</mo>
<mrow>
  <mfrac>
    <mrow>
      <mrow>
        <mn>2</mn>
        <mo>&#x2062;</mo>
        <mi>b</mi>
      </mrow>
      <mo>-</mo>
      <mrow>
        <mi>a</mi>
        <mo>&#x2062;</mo>
        <mi>p</mi>
      </mrow>
    </mrow>
    <msqrt>
      <mrow>
        <msup>
          <mn>4</mn>
          <mo>&#x2062;</mo>
          <mi>q</mi>
        </msup>
        <mo>-</mo>
        <msup>
          <mi>p</mi>
          <mn>2</mn>
        </msup>
      </mrow>
    </msqrt>
  </mfrac>
  <mo>&#x2062;</mo>
  <mrow>
    <mi>arctg</mi>
    <mo>&#x2061;</mo>
    <mfrac>
      <mrow>
        <mrow>
          <mn>2</mn>
          <mo>&#x2062;</mo>
          <mi>x</mi>
        </mrow>
        <mo>+</mo>
        <mi>p</mi>
      </mrow>
      <msqrt>
        <mrow>
          <msup>
            <mn>4</mn>
            <mo>&#x2062;</mo>
            <mi>q</mi>
          </msup>
          <mo>-</mo>
          <msup>
            <mi>p</mi>
            <mn>2</mn>
          </msup>
        </mrow>
      </msqrt>
    </mfrac>
  </mrow>
  <mo>+</mo>
  <mi>c</mi>
</mrow>
</math>
```

$$\int \frac{ax + b}{x^2 + px + q} dx = \frac{a}{2} \ln(x^2 + px + q) + \frac{2b - ap}{\sqrt{4q - p^2}} \arctg \frac{2x + p}{\sqrt{4q - p^2}} + c$$

From L^AT_EX to MathML and beyond

Eitan M. Gurari

Ohio State University

gurari@cis.ohio-state.edu

L^AT_EX is a high-level authoring language offering a special attention to mathematics. MathML is a low-level markup language representation for mathematics, suitable in particular for machine processing. Nemeth code, the standard used in North America for representing mathematics in braille, is a complex linear notation based on 63 symbols.

In recent years, I have been involved with the development of the T_EX4ht tool for translating L^AT_EX to hypertext, and of a tool for translating hypertext to braille. The presentation will discuss the use of L^AT_EX for authoring content for the Web, with emphasis on MathML production. Lessons learned from translating L^AT_EX to braille through MathML will also be considered.

◇

Programming dynamic L^AT_EX documents

James J. Quirk

Computer & Computational Sciences Division

Los Alamos National Laboratory

quirk@lanl.gov

This talk will present an overview of a co-operative programming model for generating dynamic L^AT_EX documents. The basic aim, at least in the area of computational science where the model was conceived, is to allow researchers to substantiate scientific articles with inline computer simulations whose code is open to hard scrutiny.

The current implementation (see <http://www.amrita-ebook.org/drink-me>) leverages off pdfL^AT_EX in a sufficiently general manner to be of interest beyond its specialist origins. And the talk will describe how T_EX is utilized to bring out its typesetting strengths, while hiding its programming weaknesses. Thus the material might serve to add a fresh perspective on the developments needed to keep T_EX relevant in the 21st century.

eXaMpLe

Hans Hagen

PRAGMA ADE

The eXaMpLe project started as an experiment to bring XML into ConT_EXt. When this was accomplished, a logical next step was to provide means to comfortably embed ConT_EXt in workflows that deal with this kind of coding. This effort resulted in the eXaMpLe framework. This framework offers ConT_EXt users the following benefits:

- remote access to a ConT_EXt server by means of a client-server applications; one can either send requests by HTTP or drop request in hot folders
- technology to hide the nasty parts of T_EX and related applications behind a user-interface, which itself is generated by ConT_EXt
- an experimental editing environment for XML documents, either combined with, or not combined with T_EX, driven by roles (authors, editors, reviewers)

The eXaMpLe framework is used by PRAGMA ADE, as well as its customers, for instance for the following:

- interfacing to T_EX, Ghostscript and other applications in the typesetting workflow
- form based editing of letters and other small documents
- typesetting on demand, based on user requirements, from XML databases
- building and maintaining resource libraries (e.g. graphics)
- generating exams and drill-and-practice documents, based on user input

In this talk I will demonstrate the tools that are part of the eXaMpLe framework. I will also discuss the (XML based) scripting environment that drives the processes.

Web services for CTAN

Jim Hefferon
 St Michael's College
ftpmaint@tug.ctan.org

The material held in the Comprehensive T_EX Archive Network sites is a great resource, but finding solutions to problems can be hard, especially for beginners or infrequent T_EX users. The emerging standards of web information services promise to help. For instance, users looking for a package with some feature can now go to CTAN and search Graham Williams's *Catalogue* for keywords. However, currently this is done by hand: you fire up a browser and click around to enter the information. This could happen behind the scenes if CTAN furnished the results in some standard format. Web information services provide that format.

This talk will look at the work done so far, at plans for the future, and will also solicit ideas for other services.

(We expect to publish the full paper in the next regular issue of *TUGboat*. *Ed.*)

ERCOT_EX: Yet another database publishing application of L^AT_EX

Stephan Lehmke
 QuinScape GmbH
 Dortmund, Germany
Stephan.Lehmke@QuinScape.de
<http://www.QuinScape.de>

In the talk, a *database publishing system* for paper-based product presentations using pdfL^AT_EX is presented which was developed to provide

1. top-quality typography;
2. completely automated document generation;
3. high flexibility for design and specification of documents from product 'specsheets' to complete catalogues;
4. multi-language support;
5. efficient production of very high volumes (number of documents, number of pages).

While some of the features (typographic excellence, multi-language support, support for high volumes) are provided by pdfL^AT_EX 'out of the box' and at most require appropriate tweaking of T_EX's parameters, to provide the optimal combination of completely automated document generation and high flexibility for document design and specification, a dedicated system consisting of several macro packages and document classes was created. The heart of the system consists of a macro package for managing a layout grid which is placed behind every page. Several pages can be constructed in parallel by placing objects (graphics, text, tables) in the grid. Grid cells are reserved according to measured dimensions of the placed objects.

It is possible to inspect the grid of a given page for free cells and continue construction accordingly. Text can flow between pages, either on a path which is calculated automatically, or through a predefined sequence of grid cells. Objects can be grouped and the group placed as a composite object (including the possibility for multi-column placement and page breaks). Dimensions of objects and groups can be measured, providing case distinctions for switching between design variants.

The automated production of a document requires a *data record* (T_EX file in key-value syntax) and a *document description* (say, for a product specsheet) formulated in a special *document design language*, where data contents are placed in the grid.

(We expect to publish the full paper in the next regular issue of *TUGboat*. *Ed.*)

Latin Modern: Enhancing Computer Modern with accents, accents, accents

Bogusław Jackowski
BOP s.c., Gdańsk, Poland
B.Jackowski@gust.org.pl

Janusz M. Nowacki
Foto-Alfa, Grudziadz, Poland
J.Nowacki@gust.org.pl

Abstract

The number of (free) fonts prepared with METAFONT is surprisingly small compared, e.g., to what is available in the commercial market in other formats. Well, perhaps not so surprisingly: METAFONT generates T_EX-oriented bitmap fonts which have not become popular outside the T_EX world.

Accepting this irksome situation as a challenge, we have prepared METATYPE1, a package for generating fonts in the PostScript Type 1 format accepted world-wide. The package makes use of METAFONT's "sister", namely METAPOST, and a few other utilities, such as `awk` and `t1utils`.

Recently, an opportunity arose to embark METATYPE1 upon enhancing the Computer Modern family of fonts with diacritical characters, thus following in Lars Engebretsen's footsteps, who also recognized the importance of the problem and created the AE (Almost EC) collection of virtual fonts. The task turned out to be fairly complex but well-suited for a fully programmable engine like METATYPE1. We here report on the outcome of the project, i.e., the Latin Modern family of fonts in the Type 1 format, and share the experiences gathered while accomplishing the task.

1 Introduction

Accented characters play the rôle of *enfants terribles* in the world of computers. Anybody who has to communicate with another computer system in a language other than English knows that using so-called "funny characters" is not fun at all.

1.1 Those pesky diacritics

A giant step towards putting some order into the chaos was the Unicode standard (ISO/IEC 10646) published ten years ago. Unicode, obviously, does not remove all the problems from the font playground, and even adds a few new ones (e.g., problems with the size of font files and with the registration of non-standard characters and languages). Nevertheless, one can believe that the world will become a bit better when Unicode turns from the standard *de nomine* to the standard *de facto*.

T_EX's 8-bit (i.e., 256 characters per font) paradigm is becoming more and more obsolescent, and enhancing it with multi-byte character codes seems inevitable. Such efforts as the Ω Project [11], developed by John Plaice and Yannis Haralambous, can-

not be overestimated from this point of view. But the typesetting system itself is only one side of the coin. The other is the collection of fonts it uses.

Originally, T_EX was equipped with the Computer Modern family of fonts (CM) which did not contain diacritical characters. Those few T_EX users who would need accented letters were supposed to employ the `\accent` primitive. The immense popularity of T_EX in countries that use lots of diacritical characters invalidated this presumption. At least three reasons can be set forth: (1) accented characters do not behave like "normal" ones, i.e., they interfere with important T_EX algorithms such as hyphenation and insertion of implicit kerns; (2) the CM fonts do not contain all necessary diacritics, e.g., an ogonek accent (used in Polish, Lithuanian, Navajo) is missing; (3) such diacritical elements as cedilla and ogonek, when treated as "accents", overlap with a letter, which precludes some applications, e.g., preparing texts for cutting plotters (see figure 1), even if outline fonts are used. The lesson is obvious—the CM family should be extended by a variety of diacritical letters.

In this paper we would like to present our approach to solving the problem, i.e., the open source family of fonts, *Latin Modern* (LM), in the PostScript Type 1 format [2], prepared using METATYPE1, a METAPOST-powered package [8] (see section 2.4). We believe that the LM family is a decent alternative to the other extensions of the CM family—we expect it to be a handy collection of fonts for typesetting in Latin-based alphabets. The fonts are also equipped with *Printer Font Metric* files (*.pfm) and therefore can be used as system fonts in GUI systems. Finally, they can be used with the CM metrics (e.g., via `psfonts.map`), so as to preserve typesetting of existing documents.

1.2 A gulf of history

Needless to say, the lack of diacritical letters in the CM family was recognized almost from the very beginning by T_EX users who had to struggle with the typesetting of languages other than English. Only in 1990, however, during the TUG meeting in Cork, Ireland, did the international T_EX community decide that fonts in the so-called Cork Encoding (EC or, in L^AT_EX lingo, T1) should be prepared for European T_EX users [6]. The work on EC fonts started soon after the Cork meeting. Norbert Schwartz designed a prototype, the so-called DC fonts. The work was then continued by a team led by Jörg Knappen. The final release of EC fonts was announced in 1997.

It was an important achievement. Nevertheless, the Cork Encoding conformed to T_EX’s 8-bit paradigm and therefore was not able to contain all characters occurring in European languages, not to mention other Latin-based alphabets, such as Vietnamese and Navajo.

For a few years, EC fonts were available only in a T_EX-specific bitmap form (`pk`). Nowadays, with the advent of electronic publishing, bitmaps are not acceptable. At least two factors can be pointed out:

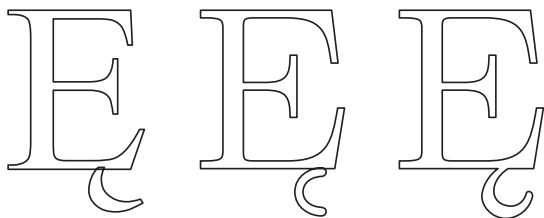


Figure 1: The letter *Eogonek* from Times New Roman for Windows XP (left), from `aer10` (middle), and from `lmr10` (right); only the latter form, with a single outline, is acceptable in professional applications.

(1) the scaling of bitmap fonts is troublesome—they look nice only if their resolution matches the resolution of the device; (2) in many cases, outline fonts turn out to display much better and, paradoxically, faster on a screen, e.g., when used in pdf. (Happily, Adobe Acrobat 6 has improved handling of bitmap fonts considerably, and non-Adobe programs with decent results are also available.)

This inspired Lars Engebretsen, who prepared a set of T_EX virtual fonts containing basic diacritical characters [4]. The virtual fonts could refer to the excellent outline version of the CM family which had appeared in the meantime. It had been created in 1988 by Blue Sky Research for the American Mathematical Society in PostScript Type 3 format, converted in 1990 by Y&Y into the hinted Type 1 format, and released in 1997 for public use by the AMS. Engebretsen called his collection AE—“Almost EC”. His virtual fonts suffer, however, from the same limitation as T_EX does, i.e., the number of characters is limited to 256. Moreover, as we have mentioned, superimposing a diacritical element on a character reveals undesirable features when the character is stroked rather than filled (see figure 1).

Only recently, automatically traced fonts in the PostScript Type 1 format, based on the EC fonts, have been published: Péter Szabó’s Tt2001, Vladimir Volovich’s CM-super (both in 2001; [14] and [15], respectively), and a newfangled CM-LGC from Alexey Kryukov (March 2003). Note, however, that Szabó courteously “recommends the wonderful CM-super package instead of his own Tt2001”. Indeed, Volovich’s collection contains many more font variations and covers a broader character set than Szabó’s. Kryukov’s collection is, in a way, a supplement to CM-super. The creation of these packages was possible thanks to a marvelous tool provided by Martin Weber, namely, `autotrace` [16].

Volovich’s accomplishment seems to bring to an end the long-lasting endeavours to introduce diacritical characters into T_EX’s realm. Do we really need yet another collection of fonts?

2 Another viewpoint

Autotraced fonts, in spite of their many advantages, have drawbacks. Objectively, the most important one is perhaps the size of a font. Such fonts are usually larger than visually similar fonts having carefully designed outlines because of a greater number of nodes in the outlines. Compare, for example, Volovich’s fairly tidy CM-super fonts with AMS CM and LM: the number of bytes per character is 260, 200, and 135, respectively. Twice is not too much, but when many magnifications are included (see sec-

tion 2.1) it makes a difference. Incidentally, the size of the CM-super fonts can be reduced by circa 10 percent by using a subroutine compression module PACKSUBR from METATYPE1 (actually, it is a short `awk` script).

For us, however, more important are arguments of a rather imponderable nature. We stand firmly by the conception underlying the \TeX and METAFONT design: *every detail, be it a typesetting or a typeface design, should be controllable and replicable.*

This is not the case with autotraced fonts. You must rely, e.g., on the nodes selected by the tracing engine. Volovich notes that the `FontLab` program (very good but commercial) was used for improving the fonts, namely, for hinting and reducing the number of nodes; therefore, the process cannot be easily repeated somewhere else. In other words, there are actually no sources for the CM-super family. The consequence is that `tfm` files have to be generated from `afm`'s (using, e.g., the `AFM2TFM` program), which adds further uncontrolled factors. For example, one cannot suppress overshoots, i.e., characters ‘o’ and ‘x’ will usually have slightly different heights, unlike the original CM fonts.

Speaking of the `AFM2TFM` converter, please note that unfortunately it cannot produce mathematical fonts. One has to use METAFONT or METAPOST (or manually edit property lists generated by `tftopl` or `vftovp`) in order to exploit such features as `charlist` or `extensible`. Ignoring this aspect would mean, in our opinion, the waste of the \TeX equipment for mathematics.

Having said this, we would like to emphasize that we highly esteem the work of Szabó, Volovich, and Kryukov. Our predilection to another solution may be regarded as a natural, if not advisable, difference of viewpoints.

2.1 Too many font sizes

There is one more issue, related indirectly to the problem of “bitmaps versus outlines”, namely, the number of font sizes for a given typeface, or more adequately — proportions. Donald E. Knuth, following the typographic praxis, implemented fonts having different proportions for different sizes (5, 6, 7, 8, 9, 10, 12 and 17 points). John Sauter attempted to go even further [13]. He prepared METAFONT programs that interpolate (and even extrapolate) Knuth’s font parameters to non-integer font sizes. We can accept Sauter’s approach as an interesting experiment, admissible for bitmap fonts. Nevertheless, using it for outline fonts is at least controversial.

We believe that, in general, four font proportions would suffice: heading (17 pt), normal (10 pt),

script (7 pt), and second-order script (5 pt, “script-script”). Because of the well-established tradition, we cannot refrain from using Knuth’s scheme, but we would strongly discourage extending it.

For these reasons, we accept with difficulty the enormous number of different sizes and proportions present in both the EC and CM-super font families. This is apparently the inheritance of Knuth’s and Sauter’s ideas. We would gladly discard most of the fourteen renditions of a single typeface (in sum, font sizes 5, 6, 7, 8, 9, 10, 10.95, 12, 14.4, 17.28, 20.74, 24.88, 29.86, and 35.83 points). The series proposed by Knuth plus the \TeX `scaled` and `at` operations provide sufficient means to deal with font scaling in most applications.

2.2 Too few typefaces

If anything, completely new typefaces are needed. The number of fonts prepared with METAFONT is surprisingly small compared, e.g., to what is available on the commercial market. Well, perhaps not so surprisingly. As we have already mentioned, METAFONT generates \TeX -oriented `pk` bitmap fonts which have not become popular outside the \TeX world. In principle, the conversion of `pk` bitmaps into PostScript Type 1 form is possible, as Szabó and others have proven. Which does not mean that looking for alternative tools is impractical.

2.3 Alternative tools

In general, computer tools fall into two classes: visual (interactive) and logical (programmable). Perhaps someday the classes will converge and “visual-and-logical” tools will prevail, but at present, without doubt, interactive tools are in vogue. The majority of contemporary visual typographic programs are commercial products. Fortunately, George Williams launched (in 2000) an impressive open source project, `FontForge` [17] (originally named `PfaEdit`). This font editor is already a powerful tool and, being extensively developed, it promises even more for the future, providing an alternative to the proprietary products. Another interesting visual tool for generating PostScript Type 1 fonts is Richard Kinch’s `MetaFog` [9], which enables visual tuning of METAPOST-generated PostScript files.

Programming tools are not so popular. Are they to go extinct some day? We hope they will not. It would be a pity, because in some applications programmability is better. Fortunately, there exist people who share our point of view. One of them is Włodek Bzyl, who found a plausible application for the logical approach in typography. His amazing colour PostScript Type 3 fonts are no mean

challenge for those who use visual tools [3].

Fonts are very complex structures. They are governed by a large set of interdependent parameters, such as character dimensions, font-specific parameters (italic angle, x-height, typical stems), characteristic shapes (serifs and arcs), not to mention such technicalities as hints or subroutines. And here an important aspect of programmability enters. By definition, programmable tools require sources in a human-readable text form. A plethora of standard text processing utilities (`awk`, `perl`, `grep`, `diff`) can therefore be employed to crosscheck the consistency of the data describing the font. This can hardly be achieved with purely interactive programs — although it should be noted that some interactive typographic programs have implemented limited programmability.

2.4 METATYPE1

We prefer unlimited programmability. Provoked by the irksome scarcity of fonts prepared using METAFONT, we contrived another font generating package, METATYPE1 [8], based on METAPOST, which produces results in the widely accepted PostScript Type 1 format. The package makes use of two sets of METAPOST macros (the general purpose `plain_ex` and the task-oriented `fontbase`) and a few other utilities, such as `awk` (for processing METAPOST output), `t1utils` (for converting text data into a binary form), and `mft` (for neat proofing). Originally, METATYPE1 was developed for DOS; thanks to Włodek Bzyl, it is also available for Linux.

Some of the first results obtained with METATYPE1 was Donald E. Knuth’s `logo` font and an electronic replica of a traditional Polish font, `Antykwa Półtawskiego` [7]. (Available from `ftp://ftp.GUST.org.pl/pub/TeX/GUST/contrib/fonts/replicas`). We also used METATYPE1 for auditing and enhancing selected fonts from the URW++ collection distributed with `Ghostscript`.

In 2002, during the \TeX meeting in Bachotek, Poland, representatives of the European \TeX user groups, having discussed matters via email, devised a proposal for converting the AE virtual fonts into a more universal PostScript Type 1 format and also augmenting them with a set of necessary diacritical characters. Thus the opportunity arose to try METATYPE1 on a new, unconventional task. We took up the gauntlet without hesitation.

3 The Latin Modern family of fonts; or details, details, details

Our intention was to preserve the AE name, as we wanted to emphasize the rôle of Engebretsen’s idea

<code>lmb10</code>	<code>lmr17</code>	<code>lmss10</code>	<code>lmssqbo8</code>
<code>lmb010</code>	<code>lmr5</code>	<code>lmss12</code>	<code>lmssqbx8</code>
<code>lmbx10</code>	<code>lmr6</code>	<code>lmss17</code>	<code>lmssqo8</code>
<code>lmbx12</code>	<code>lmr7</code>	<code>lmss8</code>	<code>lmtcsc10</code>
<code>lmbx5</code>	<code>lmr8</code>	<code>lmss9</code>	<code>lmtt10</code>
<code>lmbx6</code>	<code>lmr9</code>	<code>lmssbo10</code>	<code>lmtt12</code>
<code>lmbx7</code>	<code>lmri10</code>	<code>lmssbx10</code>	<code>lmtt8</code>
<code>lmbx8</code>	<code>lmri12</code>	<code>lmssdc10</code>	<code>lmtt9</code>
<code>lmbx9</code>	<code>lmri7</code>	<code>lmssdo10</code>	<code>lmtti10</code>
<code>lmbxi10</code>	<code>lmri8</code>	<code>lmss010</code>	<code>lmtto10</code>
<code>lmbxo10</code>	<code>lmri9</code>	<code>lmss012</code>	<code>lmvtt10</code>
<code>lmcsc10</code>	<code>lmro10</code>	<code>lmss017</code>	<code>lmvtto10</code>
<code>lmcsc010</code>	<code>lmro12</code>	<code>lmss08</code>	
<code>lmr10</code>	<code>lmro8</code>	<code>lmss09</code>	
<code>lmr12</code>	<code>lmro9</code>	<code>lmssq8</code>	

Figure 2: The Latin Modern collection of fonts.

in this enterprise. Soon it became clear, however, that the differences would be fundamental and that the change of the name would be necessary in order to avoid confusion. Therefore, we coined the name “Latin Modern” to foreshadow further development — we would like the final version of LM to comprise as many Latin-based alphabets as possible, e.g., Vietnamese (which regretfully is not included yet).

The collection of AE fonts consisted of 50 fonts, reasonably selected from the abundance of Computer Modern. We decided to add a variable-width typewriter font and a few oblique derivatives, arriving finally at 57 fonts (see figure 2). Observe two details:

1. We adopted a more regular (although unorthodox) font naming convention with respect to slanted/italic variants: we have used the letter ‘o’ as a suffix for oblique (slanted) fonts and the letter ‘i’ as a suffix for truly italic fonts. The 8-character limit is preserved.
2. The LM family contains the font `lmssqbx8` (i.e., the bold version of `lmssq8`); a corresponding font occurs neither in CM nor in EC. Actually, the respective AE fonts (`aessq8`, `aessqi8`, and `aessqb8`) refer to the fonts `lcmss8` `lcmssi8`, and `lcmssb8`. These fonts, added by Pierre A. MacKay, were meant to be used with `SL \TeX` . Their regular variants are nearly identical with Knuth’s `cmssq8` and `cmssqi8`. The only difference is the capital ‘I’ (see figure 3).

The issue of font names was triggered by the slanted fonts that we decided to add: what name should we assign to the oblique variant of `lmvtt10`? The name `lmvtt110` did not conform to the Knuthian 8-character scheme, while the name `lmvtti10` did not tell the truth. After thinking the problem over, we could not find the reason why oblique fonts,



Figure 3: The letter *I* from Knuth’s `cmssq8` (left) and MacKay’s `1cmss8` (right).

i.e., the mechanically skewed ones, received the designator ‘i’ in some cases (e.g., `cmssi10`) and the designator ‘sl’ in other (e.g., `cmbxs110`), and why the designator appeared either at the end of the kernel of the name, as in the mentioned examples, or—in some cases—immediately after the prefix ‘cm’ (`cmslitt10`, `cmitt10`).¹ We could either uphold traditional Knuth’s terminology (but what then should we call oblique `lmttt10`?) or take an opportunity and introduce some regularity in font naming at the risk of commencing an incompatibility mess. We have chosen the latter solution. . .

The issue of an alternative letter ‘I’ necessitated, besides undertaking a decision whether to introduce it or not (we decided to introduce it as a variant letter), some extra work due to the addition of variant accented characters and a variant ligature *IJ*. The `lmssq*` fonts became thus somewhat exceptional. This is usually undesirable but sometimes cannot be avoided.

The reader may wonder why we dwell on such trifles? The answer is simple: it was the mass of details of this kind that made the work on the LM family laborious, although individual tasks were relatively simple. In other words, the problem with details is that each of them, even the tiniest one, has to be handled *somehow*—as the amount of details grows, the job becomes more complex.

Enumerating all dilemmas, technicalities, subtleties or even puzzles with which we had to struggle is obviously pointless. On the other hand, our work consisted nearly exclusively of such details—how to describe such a work? Perhaps the best method is to let the reader perceive the scent of the battleground by showing representative examples. Two such examples we have already indicated. The rest of the paper presents a few more.

3.1 From PostScript to METATYPE1 sources

The process of conversion of fonts from PostScript

¹ The reason turns out to be that on the original SAIL development computer, the file name limitation was 6+3, even worse than 8+3, and the shorter names were generated by taking the first 3 and last 3 characters from the longer. The names for Computer Modern were chosen to be unique after applying this procedure. *Ed.*



Figure 4: The optical axis of a glyph does not necessarily coincide with the geometric center of the glyph. Compare the corrected placement of the accent in *gcommaaccent* (left) with the default one (right).

Type 1 form into METATYPE1 sources is only moderately relevant since the potential users of the LM fonts are not expected to repeat this operation any more. The METATYPE1 sources are legible and can easily be modified, if necessary.

We used a stand-alone utility PF2MT1 (belonging to the METATYPE1 package) for the translation of `pfb+afm` pairs from CM fonts into METATYPE1 code. The virtual AE fonts provided the necessary information for merging the results of the conversion. `awk` turned out to be a very convenient tool for such operations. Thanks to it, the framework of the LM sources was ready after a few hours; amending the LM sources took a few months.

3.2 Tuning and augmenting the METATYPE1 sources of the LM fonts

The main part of the job, although also the simplest one, was adding accents. METATYPE1 provides a `use_accent` operation, similar to the `TeX \accent` primitive, that can conveniently be used for this purpose. By default, `use_accent` aligns the centre of an accent with the centre of its accentee and raises the accent by $x - h$, where x is the value of the x -height parameter, and h is the height of the character. This is the procedure used by `TeX` for accenting. Such an algorithm is not always appropriate. Occasionally, the position of an accent may have to be adjusted. The command `use_accent` enables an arbitrary shift of both accent and accentee. Moreover, a supplementary glyph axis parameter can optionally be specified for each character (see figure 4).

All in all, adding accented letters was child’s play. Somewhat more difficult was adding extra characters.

In the AE family, the characters were brought together from several different sources. For example:

- `aer10`: *arrow left hook* (i.e., faked *ogonek*);
- `cmmi10`: *less*, *greater*, *bar*, *backslash*, *braceleft*, *braceright*, and *section*.

Figure 5: There are two acute accents in LM fonts: a flattened variant is used for capital letters. This idea was implemented in PL fonts and then in EC. In general, the flattening is neither a slanting nor a rotation.

- `cmu10`: *sterling*.
- Some characters were drawn using rules: *visiblespace*; missing characters were marked by a rule having width and height equal to $\frac{1}{2}$ em.
- Others were assembled from components: *Aogonek*, *aogonek*, *Eogonek*, *eogonek*.

For Latin Modern, we went even further, and “borrowed” the characters *asciicircum* and *asciitilde* from `cmex10`; *mu* — from `cmmi10`; *dagger*, *daggerdbl*, and *paragraph* — from `cmsy10`.

It is debatable whether borrowing characters is acceptable. The *section* sign from `cmsy10` is certainly an alien in a sans serif font. Therefore, characters that seemed to us sufficiently important (*section*, *sterling*) were programmed from scratch. We used, of course, appropriate parameters from the CM driver files, but we did not follow Knuth’s recipe rigorously. This might have been done (see the comments on the *Euro* symbol below). We preferred, however, our shapes of glyphs. This may evoke some compatibility-related issues but, anyway, full compatibility among CM, EC, and AE fonts cannot be achieved (see section 3.3).

Actually, some characters were borrowed not from CM fonts but from their PL counterparts (i.e., CM fonts equipped with Polish diacritical letters; the relevant METAFONT code from the PL fonts was incorporated into the EC sources). The acute and grave accents over capital and small letters in PL fonts differ, namely, accents over capital letters are flattened — we applied the same approach in the LM fonts (see figure 5) which is consistent with EC and inconsistent with CM.

Besides the accented, borrowed and newly programmed variant characters, a few glyphs had to be programmed from scratch as consistently as possible with the CM typeface design. A notable example is a *Euro* currency symbol. It looks as though it became so important recently that Adobe even assigned it a name beginning with a capital letter (cf. *dollar*,

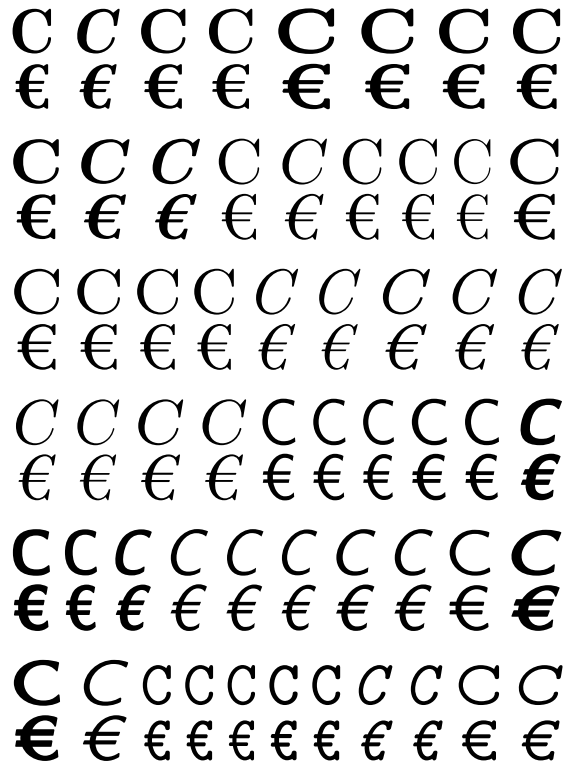


Figure 6: *Euro* symbols from the LM fonts; observe that a *Euro* symbol is narrower than the corresponding letter *C* (above), but that the stem sizes are preserved. Unfortunately, there is no slot for a *Euro* symbol in the *Cork Encoding*.

yen, *sterling*, etc., in the *Adobe Glyph List For New Fonts* [1]). We attempted to exploit the METAFONT code for the letter *C* — and it worked (see figure 6). The *Euro* design is philosophically based on a script *E*, not a *C*; therefore, our design has a bottom serif to be more distinguishable from *C* (many thanks to Werner Lemberg).

The LM fonts also contain a few idiosyncratic symbols. We wanted, for example, to have a ligature *f-k* in the repertoire of characters (see figure 7) because there are several words in Polish containing the sequence ‘fk.’ They are less numerous than words with ‘fi’ and ‘fl’ but more than words with ‘ffi’ and ‘ffl’ (which occur exclusively in words of foreign origin).

Of course there are more candidates for non-standard ligatures, e.g., ‘fb’, ‘fh’, ‘fj’, ‘ffb’, and ‘ffh’. These groups of letters occur sporadically in English and German (they are absent from Polish), and may be included in a future release.

3.3 Compatibility issues

The answer to the question of whether the LM fonts

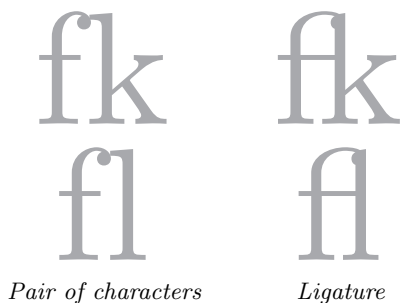


Figure 7: There are several words in the Polish language that contain the digraph ‘fk’; therefore, we included the ligature *f.k* (top-right) in the LM character set for the sake of consistency with native CM ligatures, such as *fl* (bottom-right).

can serve as a replacement for CM or EC ones is obviously ‘no’. First of all, the collection of fonts is different—LM is a subset of CM (except `lmssqb8` and a few oblique derivatives), not to mention EC. Therefore, not every text typeset with CM or EC fonts can be re-typeset using LM ones.

On the other hand, it should be noted that LM fonts are based on the data taken from CM driver files. Therefore, all relevant dimensions are (or at least should be) the same in LM and CM fonts within the accuracy of rounding errors. It is thus possible, for example, to use existing LM fonts as a replacement for CM in the `dvips` file `psfonts.map`—it suffices to prepare appropriate encoding (`*.enc`) files.

In order to reach this level of compatibility, we had to add two more characters, namely *arrowup* and *arrowdown* which, somewhat surprisingly, are present in `cmr5`, but not in other fonts in the `cmr*` series. At the same time, we resisted the temptation to include a full quiver of other arrows. The main reason was that arrows are absent from the basic *Cork Encoding* (they appear only in the *Text Companion Encoding*—see, e.g., the file `dcdoc.tex` distributed with the EC sources); moreover, since PostScript is already involved, various transformations can easily be applied, if necessary. In the future, however, we may change our opinion.

The METATYPE1 programs for the arrows are based on METAFONT sources contained in `sym.mf`. While adapting the code, we encountered a quandary which is a good example of a seemingly trivial yet embarrassing detail. It turns out that the arrow programs produce questionable results for certain driver files; namely, the sidebearings disappear! The arrow programs were perhaps never tested with all driver files. One could live with this; nevertheless, we decided to preserve minimal space at both

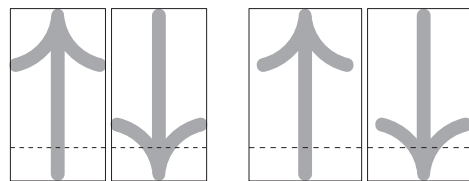


Figure 8: The METAFONT program for arrows (in `sym.mf`) would produce glyphs stripped of sidebearings for parameters from `cmssdc10` (left); arrows in LM fonts always have sidebearings (right).



Figure 9: The caron alias hacek accent (the leftmost box) is slightly lowered in the CM fonts; in the LM fonts, all accents are aligned horizontally.

sides—the result is certainly more palatable (see figure 8).

Another quandary is related to accents. For some inexplicable reason, the caron accent in CM fonts is lowered in relation to the other accents (see figure 9). We considered it a fault and decided to raise all carons appropriately. We thus relinquished full compatibility between CM and LM families—although we hope this visual “incompatibility” will be seen as an improvement.

3.4 The game of names

Among many technicalities related to the representation of PostScript fonts, we would like to comment upon only one—the particularly upsetting problem of character names.

There exists a standard of glyph naming worked out by Adobe [1], contained in the documents *Adobe Glyph List 2.0* (of 20th September 2002) and *Adobe Glyph List for New Fonts 1.1* (of 17th April 2003). Regrettably, the standard contains numerous entries that are at best dubious. We have already scoffed at the name of the *Euro* symbol that singularly begins with a capital letter. But this is nothing. The excerpt from the *Adobe Glyph List For New Fonts* concerning characters with *commaaccent* is really astounding (see figure 10). Even more astounding is a part of this story pertaining to *Tcedilla* and *tcedilla*:

- Version 1.1 of *Adobe Glyph List* mentioned the characters described as ‘T with cedilla’ and ‘t with cedilla’ and assigned them names *Tcommaaccent* and *tcommaaccent*, respectively; the


```
Gcommaaccent; LATIN CAPITAL LETTER G WITH CEDILLA
Kcommaaccent; LATIN CAPITAL LETTER K WITH CEDILLA
Lcommaaccent; LATIN CAPITAL LETTER L WITH CEDILLA
Ncommaaccent; LATIN CAPITAL LETTER N WITH CEDILLA
Rcommaaccent; LATIN CAPITAL LETTER R WITH CEDILLA
Scommaaccent; LATIN CAPITAL LETTER S WITH COMMA BELOW
gcommaaccent; LATIN SMALL LETTER G WITH CEDILLA
kcommaaccent; LATIN SMALL LETTER K WITH CEDILLA
lcommaaccent; LATIN SMALL LETTER L WITH CEDILLA
ncommaaccent; LATIN SMALL LETTER N WITH CEDILLA
rcommaaccent; LATIN SMALL LETTER R WITH CEDILLA
scommaaccent; LATIN SMALL LETTER S WITH COMMA BELOW
```

Figure 10: An excerpt from the up-to-date *Adobe Glyph List For New Fonts* [1]. How sweet. . .

characters that could be described as ‘T with comma below’ or ‘t with comma below’ were simply ignored.

- In version 1.2 of the *Adobe Glyph List*, the names *Tcommaaccent* and *tcommaaccent* were now assigned both to characters described as ‘T or t with cedilla’ and ‘T or t with comma below’.
- The up-to-date *Adobe Glyph List for New Fonts* says that the most recent change was renaming “[Tt]cedilla back to [Tt]commaaccent”; the previous version was derived from *Adobe Glyph List 2.0* and one of a few changes was “renaming *tcommaaccent* to *tcedilla* and *Tcommaaccent* to *Tcedilla*”. Note that in the current version both *Tcommaaccent* and *tcommaaccent* are described as “letter with cedilla” . . .

To untangle the “commaaccent” story a little bit, we would like to quote a more reliable opinion from Michael Everson’s web site devoted to European alphabets [5]:

- Concerning Latvian: “The [accented] characters g, k, l, n, r, G, K, L, N, and R must always be drawn with a *comma below*, although these characters are identified in ISO standards as *letters with cedilla*. Note particularly the reverse comma accent used with the *latin small letter g with cedilla*.” (Cf. figure 4.)
- Concerning Romanian: “Note that Romanian uses the characters *s with comma below* and *t with comma below*. In inferior Romanian typography, the glyphs for these characters are sometimes drawn with *cedillas*, but it is strongly recommended to avoid this practice.”

There were more pitfalls of this kind, not as ridiculous as the case of the *commaaccent*, but sufficiently confusing to make this part of the job quite arduous.

Given such a state of the art, we decided to copy some glyphs under different names—just in case.

We repeated, e.g., the glyphs *scommaaccent*, *tcommaaccent*, *Scommaaccent*, and *Tcommaaccent* under the names *scedilla*, *tcedilla*, *Scedilla*, and *Tcedilla*, respectively. Altogether, there are approximately 10 duplicated characters per 400-character font.

The duplication of glyphs does *not* lead to an enormous inflation of the size of font files because of a very efficient subroutine packing mechanism (cf. section 2, p. 66). Actually, a duplicated character only increases the size of a font by 30–40 bytes. This means that 10 duplicated characters would increase a font size by less than 1 percent, as the average size of an LM font (pfb) is 60 KB.

3.5 Beware of your friends

The basic tools we used (`awk`, `METAPOST`, `tftopl`, `vftovp`, `tiutils`) worked nearly infallibly. Only once did we meet a truly intricate problem. It was a bug persistently offered by our friend, `METAPOST`.

One of the important operations in the process of font generation is determining the orientation of a path: anticlockwise-oriented paths are used for filling, and clockwise-oriented for unfilling. The function `turningnumber` in `METAFONT` and `METAPOST` returns +1 and –1 for anticlockwise-oriented and clockwise-oriented paths, respectively. In `METAFONT` it works correctly; in `METAPOST`, unfortunately, it does not. The bug manifests its presence even in such trivial cases as the following (see the top element in figure 11):

```
path p;
p=(0,10)..controls (5,10) and (10,5)
  ..(10,0)..controls (10,-5) and (5,-10)
  ..(0,-10)..controls (-5,-10) and (-10,-5)
  ..(-10,0)..controls (-10,5) and (-5,10)
  ..cycle;
```

This nearly circular 4-node path is evidently clockwise-oriented. Nevertheless, `METAPOST` maintains that `turningnumber p = 0`.

We did not analyse the `METAPOST` source code as we were not going to fix the bug, but circumventing it was crucial. The only method that proved to work was the “straightening” of a path prior to the application of the `turningnumber` function; in other words, each Bézier segment of a path was changed to a straight line and then the `turningnumber` function was applied to the modified path. It works well enough so far, although the method is not general (see the bottom two pairs in figure 11) and, moreover, frequently used straightening slows down the process of generating fonts.

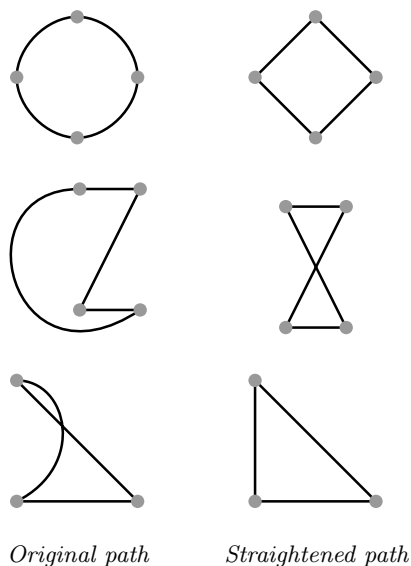


Figure 11: The operation of straightening a path typically does not change the orientation of a path (top); this works around a bug in METAPOST. In general, however, this may happen — the middle and bottom pictures show how a non-zero turning number can be changed to zero and vice versa. The latter situations, fortunately, are rather unlikely in fonts.

3.6 Encodings

In olden days, there was a one-to-one correspondence between a *font name* and the name of a *font metric file* (`tfm`). This is not possible any longer. If there are more characters in a font than 256, as in CM-super and LM, one has to select a subset of characters and assign codes to every character. Even not knowing the precise results of combinatorial analysis, one may fancy how many such encodings may coexist. It seems that there is no choice — *metric files must not use the same name as the basic font*, otherwise a mess is bound to ensue.

One could think of a distinguished (main) encoding that would inherit the basic name, but we would rather equate all encodings. At present, we supply the *Cork*, *QX*, and *texnansi* text encodings in the official distribution of the LM fonts.

The Cork encoding does not need further explanation. The QX encoding is actually a “double” encoding, i.e., there is a fixed collection of characters and two numberings — one to be used with \TeX and one to be used with GUI systems [12]. It was worked out a few years ago by the members of the Polish \TeX Users Group GUST as a difficult compromise between needs and abilities. In a nutshell: the *QX Encoding* for \TeX is a variant of the *Cork Encoding* with a few characters exchanged (e.g., *gbreve*,

Gbreve, *uring*, and *Uring* are replaced by Lithuanian *iogonek*, *Iogonek*, *uogonek*, and *Uogonek*, respectively); the *QX Encoding* for GUI systems is a variant of the Code Page 1250 (and also includes Lithuanian characters with ogonek).

Recall that the complete list of the LM font names is shown in figure 2. The respective `tfm` file names are derived by adding prefixes, e.g., `cork-` for the *Cork Encoding* and the prefix `qx-` for the *QX Encoding*. For instance, `lmr10` with the *Cork Encoding* has the name `cork-lmr10` and with the *QX Encoding* the name `qx-lmr10`.

This protocol is admittedly immature. Nevertheless, we do insist on recommending either this naming scheme or a similar one as a guideline for \TeX users as long as \TeX is not capable of handling multi-byte character codes — or even longer.

3.7 Availability

One final detail: the LM fonts are freely available at <http://www.ctan.org/tex-archive/fonts/lm>. METATYPE1 is available at <http://www.ctan.org/tex-archive/fonts/utilities/metatype1>.

4 Concluding remarks

We would like to emphasize once again that our aim was not only to provide a new family of fonts, but to provide it with METATYPE1 sources that can be maintained — adjusted, augmented, improved, etc. While it is rather difficult to write a font program from scratch, it is relatively simple to modify existing sources; e.g., as we have mentioned, adding accented letters is straightforward.

As concerns our plans regarding the LM family, we would like to enhance fonts: to extend the repertoire of characters (first of all by the *Text Companion* for the EC fonts²), to improve kerning, hinting and shapes of certain glyphs, and, last but not least, to provide OpenType versions of the LM fonts for XP trailblazers. We consider, moreover, converting a few more CM programs from METAFONT to METATYPE1, as we would like to eventually dismiss the borrowed characters (see section 3.1, p. 69).

Before bringing the curtain down, we would like to draw the reader’s attention to a weak point of our approach: the CM parameterization has been lost. The METATYPE1 sources can be enhanced, but they cannot be used for producing, say, light or condensed versions of sans serif fonts. An experiment with the programming of the *Euro* symbol and the arrows has shown that converting METAFONT sources

² This has been accomplished as this *TUGboat* issue goes to press.

to METATYPE1 ones without losing the parameterization is, in general, possible but rather time-consuming. It is an open question whether such a venture, while extremely attractive, is practical.

5 Acknowledgments

The project is supported by European T_EX user groups, in particular by the German-speaking T_EX users group DANTE e.V., the French-speaking T_EX users group GUTenberg, and the Dutch-speaking T_EX users group NTG; and also by the T_EX Users Group — very many thanks to all. We are also grateful to Volker Schaa and Stefan Sokółowski for their valuable comments concerning the draft version of the paper.

References

- [1] *Adobe Solutions Network: Type Technology — Unicode and Glyph Names*, <http://partners.adobe.com/asn/tech/type/unicodegn.html>
- [2] *Adobe Type 1 Font Format*. Addison-Wesley, 1990, <http://partners.adobe.com/asn/developer/pdfs/tn/T1.SPEC.PDF>
- [3] Włodzimierz Bzyl, *The Tao of Fonts*. Proc. of TUG 2002, 4th–7th September, 2002, Trivandrum, India, *TUGboat* 23(1), March 2003, pp. 27–40. <http://tug.org/TUGboat/Articles/tb23-1/bzyl.pdf>
- [4] Lars Engbrechtsen, *AE fonts*, <http://ctan.org/tex-archive/fonts/ae/>
- [5] Michael Everson, *The Alphabets of Europe* (ver. 3.0), <http://www.evertype.com/alphabets/>
- [6] Michael Ferguson, *Report on multilingual activities*, *TUGboat* 11(4), November 1990, p. 514.
- [7] Bogusław Jackowski, Janusz M. Nowacki, Piotr Strzelczyk, *Antykwa Półtawskiego: A Parameterized Outline Font*. Proc. of EuroT_EX 1999, 20th–24th September, 1999, Heidelberg, Germany, pp. 109–141.
- [8] Bogusław Jackowski, Janusz M. Nowacki, Piotr Strzelczyk, *METATYPE1: A METAPOST-based Engine for Generating Type 1 Fonts*. Proc. of EuroT_EX 2001, 27th–27th September, 2001, Kerkrade, the Netherlands, pp. 111–119, <http://www.ntg.nl/eurotex/metatyp1.pdf> and <http://www.ntg.nl/eurotex/JackowskiMT.pdf>
- [9] Richard J. Kinch, *MetaFog: Converting METAFONT Shapes to Contours*. *TUGboat* 16(3), pp. 233–243, 1995. <http://tug.org/TUGboat/Articles/tb16-3/tb48kinc.pdf>
- [10] Han-Wen Nienhuys, *MFTrace — Scalable Fonts for METAFONT*, <http://www.xs4all.nl/~hanwen/mftrace/>
- [11] John Plaice and Yannis Haralambous, *Omega System*, <http://sourceforge.net/projects/omega-system/>
- [12] *QX encoding tables for T_EX and for window systems*, <http://www.gust.org.pl/fonty/qx-table1.html>, <http://www.gust.org.pl/fonty/qx-table2.html>
- [13] John Sauter, *Building Computer Modern Fonts*, *TUGboat* 7(3), October 1986, p. 151.
- [14] Péter Szabó, *T_EXtrace*, <http://www.inf.bme.hu/~pts/texttrace/>
- [15] Vladimir Volovich, *CM-super Font Package*, <ftp://ftp.vsu.ru/pub/tex/font-packs/cm-super/>
- [16] Martin Weber, *Autotrace*, <http://autotrace.sourceforge.net/>
- [17] George Williams, *FontForge: A PostScript Font Editor*, <http://fontforge.sourceforge.net/>

A The contents of the Latin Modern family of fonts, version 0.92

For meticulous readers, we enclose below the complete list of LM glyph names in alphabetic order. Note that some characters do not occur in all fonts, e.g. there are no *f*-ligatures in the typewriter fonts. In all, there are five classes of character sets:

1. The basic class (527 glyphs); this class consists of `lmb10`, `lmb010`, `lmbx10`, `lmbx12`, `lmbx5`, `lmbx6`, `lmbx7`, `lmbx8`, `lmbx9`, `lmbxi10`, `lmbxo10`, `lmr10`, `lmr12`, `lmr17`, `lmr5`, `lmr6`, `lmr7`, `lmr8`, `lmr9`, `lmri10`, `lmri12`, `lmri7`, `lmri8`, `lmri9`, `lmro10`, `lmro12`, `lmro8`, `lmro9`, `lmss10`, `lmss12`, `lmss17`, `lmss8`, `lmss9`, `lmssbo10`, `lmssbx10`, `lmssdc10`, `lmssdo10`, `lmssso10`, `lmssso12`, `lmssso17`, `lmssso8`, `lmssso9`, `lmvtt10`, and `lmvtt010`.
2. The class ‘`ssq`’ (538 glyphs); besides the characters present in the basic class, it contains *varI*, *varIacute*, *varIcircumflex*, *varIdieresis*, *varIdotaccent*, *varIgrave*, *varIJ*, *varImacron*, *varIogonek*, *varItilde*, and *varIvardieresis*. The following fonts belong to this family: `lmssq8`, `lmssqbo8`, `lmssqbx8`, and `lmssqo8` (cf. figure 3 and the relevant comments in section 2).
3. The class ‘`typewriter`’ (512 glyphs); the following glyphs are missing in comparison with

the basic class: *f.k, ff, ffi, ffl, fi, fl, Germandbls, IJ, ij, permyriad, servicemark, suppress, trademark, varcopyright, and varregistered*. The class consists of *lmtt10, lmtt12, lmtt8, lmtt9, lmtti10, and lmtto10*.

4. The class for only *lmcsc10* and *lmcsc010* (519 glyphs); the following glyphs are missing in comparison with the basic class: *dquoteright, f.k, ff, ffi, ffl, fi, fl, and tquoteright*.
5. The class for only *lmtcsc10* (510 glyphs); the set of missing characters is as in class 3 plus *dquoteright* and *tquoteright*.

A.1 Alphabetic list of glyphs in the Latin Modern family

A a Aacute aacute Abreve abreve Acircumflex acircumflex Acute acute acute.dup acute.ts1 Adieresis adieresis AE ae AE.dup ae.dup Agrave agrave althyphen Amacron amacron ampersand anglearc angleleft angleright Aogonek aogonek Aring aring arrowdown arrowleft arrowright arrowup asciicircum asciitilde asterisk asteriskmath at Atilde atilde Avardieresis avardieresis

B b Backslash baht bar bigcircle blanksymbol born braceleft braceright bracketleft bracketright breve breve.ts1 brokenbar bullet

C c Cacute cacute caron caron.ts1 Ccaron ccaron Ccedilla ccedilla Ccircumflex ccircumflex Cdotaccent cdotaccent cedilla cedilla.dup cent centigrade centoldstyle circumflex circumflex.dup colon colonmonetary comma commaaccent copyleft copyright csquotedblbase csquotedblright currency cwm cwmascender cwmcapital

D d dagger daggerdbl dbar dblbracketleft dblbracketright dblgrave.ts1 dblverticalbar Dcaron dcaron Dcroat dcroat degree Delta diameter diedieresis dieresis.dup dieresis.ts1 discount divide divorced dmacron dollar dollaroldstyle dong dotaccent dotlessi dotlessj dquoteright

E e Eacute eacute Ebreve ebreve Ecaron ecaron Ecircumflex ecircumflex Edieresis edieresis Edotaccent edotaccent Egrave egrave eight eightoldstyle ellipsis Emacron emacron emdash endash Eng eng Eogonek eogonek equal estimated Eth eth Euro euro Evardieresis evardieresis exclam exclamdown

F f f.k ff ffi ffl fi five fiveoldstyle fl florin four fouroldstyle fraction

G g Gacute gacute Gamma Gbreve gbreve Gcaron gcaron Gcedilla Gcircumflex gcircumflex Gcommaaccent gcommaaccent Gdotaccent gdotaccent Germandbls germandbls germandbls.dup gaborretni Grave grave grave.ts1 greater guarani guillemotleft guillemotright guilsinglleft guilsinglright

H h Hbar hbar Hcircumflex hcircumflex hungarumlaut hungarumlaut.ts1 hyphen hyphenchar hyphendbl hyphendbl.alt

I i Iacute iacute Icircumflex icircumflex Idieresis idieresis Idotaccent Igrave igrave IJ ij Imacron

imacron interrobang Iogonek iogonek Itilde itilde Ivardieresis ivardieresis
 J j Jcircumflex jcircumflex
 K k Kcedilla kcedilla Kcommaaccent kcommaaccent
 L l Lacute lacute Lambda Lcaron lcaron Lcedilla lcedilla Lcommaaccent lcommaaccent Ldotaccent ldotaccent leaf less lira logicalnot Lquoteright lquoteright Lslash lslash
 M m macron macron.dup macron.ts1 married mho minus mu multiply musicalnote
 N n Nacute nacute naira nbspace Ncaron ncaron Ncedilla ncedilla Ncommaaccent ncommaaccent nine nineoldstyle numero Ntilde ntilde numbersign
 O o Oacute oacute Obreve obreve Ocircumflex ocircumflex Odieresis odieresis OE oe OE.dup oe.dup ogonek Ograve ograve ohm Ohungarumlaut ohungarumlaut Omacron omacron Omega one onehalf oneoldstyle onequarter onesuperior Oogonek oogonek openbullet ordfeminine ordmasculine Oslash oslash Oslash.dup oslash.dup Otilde otilde Ovardieresis ovardieresis
 P p paragraph paragraph.alt parenleft parenright percent period periodcentered permyriad perthousand perthousandzero peso Phi Pi plus plusminus Psi published
 Q q question questiondown quillbracketleft quillbracketright quotedbl quotedbl.alt quotedblbase quotedblbase.alt quotedblbase.ts1 quotedblleft quotedblleft.alt quotedblright quotedblright.alt quoteleft quoteleft.alt quoteleft.dup quoteright quoteright.alt quoteright.dup quotesinglbase quotesinglbase.alt quotesinglbase.ts1 quotesingle quotesingle.alt quotesingle.ts1
 R r Racute racute radical Rcaron rcaron Rcedilla rcedilla Rcommaaccent rcommaaccent recipe referencemark registered registered.alt ring
 S s Sacute sacute Scaron scaron Scedilla scedilla Scircumflex scircumflex Scommaaccent scommaaccent section semicolon servicemark seven sevenoldstyle sftthyphen Sigma six sixoldstyle slash space sterling suppress
 T t Tcaron tcaron Tcedilla tcedilla Tcommaaccent tcommaaccent Theta Thorn thorn three threeoldstyle threequarters threequartersemdash threesuperior tieaccentcapital tieaccentcapital.new tieaccentlowercase tieaccentlowercase.new tilde tilde.dup tildelow tquoteright trademark twelveudash two twooldstyle twosuperior
 U u Uacute uacute Ubreve ubreve Ucircumflex ucircumflex Udieresis udieresis Ugrave ugrave Uhungarumlaut uhungarumlaut Umacron umacron underscore Uogonek uogonek Upsilon Uring uring Utilde utilde Uvardieresis uvardieresis
 V v varcopyright vardieresis vardotaccent varl varlacute varlcircumflex varldieresis varldotaccent varlgrave varlIJ varlmacron varllogonek varltilde varlvardieresis varregistered visiblespace
 W w Wacute wacute Wcircumflex wcircumflex Wdieresis wdieresis Wgrave wgrave won Wvardieresis wvardieresis
 X x Xi
 Y y Yacute yacute Ycircumflex ycircumflex Ydieresis ydieresis yen Ygrave ygrave Yvardieresis yvardieresis
 Z z Zacute zacute Zcaron zcaron Zdotaccent zdotaccent zero zerooldstyle

CM-Super: Automatic creation of efficient Type 1 fonts from METAFONT fonts

Vladimir Volovich

Voronezh State University

Moskovsky prosp. 109/1, kv. 75, Voronezh 304077 Russia

vvv@vsu.ru

Abstract

In this article I describe making the CM-Super fonts: Type 1 fonts converted from METAFONT sources of various Computer Modern font families. The fonts contain a large number of glyphs covering writing in dozens of languages (Latin-based, Cyrillic-based, etc.) and provide outline replacements for the original METAFONT fonts. The CM-Super fonts were produced by tracing the high resolution bitmaps generated by METAFONT with the help of \TeX trace, optimizing and hinting the fonts with FontLab, and applying cleanups and optimizations with Perl scripts.

1 The idea behind the CM-Super fonts

The Computer Modern (CM) fonts are the default and most commonly used text fonts with \TeX . Originally, CM fonts contained only basic Latin letters, and thus covered only the English language. There are however a number of Computer Modern look-alike METAFONT fonts developed which cover other languages and scripts. Just to name a few:

- EC and TC fonts, developed by Jörg Knappen, which are the default \LaTeX fonts for the T1 and TS1 font encodings and cover many Latin-based scripts (mainly European).
- EC and TC Concrete and Bright fonts, developed by Walter Schmidt, which are additional font families containing the same glyphs as EC and TC fonts.
- LH Cyrillic fonts, developed by Olga Lapko, which support the family of T2 font encodings: T2A, T2B, T2C, X2, and others. They support the same font families as the original EC fonts, EC Concrete and EC Bright fonts.
- TIPA (International Phonetic Alphabet) fonts, developed by Rei Fukui, which support the T3 font encoding. There exist Concrete (CIPA) and Bright (BIPA) families of the TIPA fonts too.
- FC fonts, developed by Jörg Knappen, which support the T4 font encoding for African languages.
- VNR fonts, developed by Cuong Nguyen, Werner Lemberg and Hàn Thế Thành, which support the T5 font encoding for Vietnamese.
- CBgreek fonts, developed by Claudio Beccari, which support the Greek font encoding (LGR).

There exist free Type 1 versions of the original CM fonts, provided by Blue Sky Research, Elsevier Science, IBM Corporation, the Society for Industrial and Applied Mathematics (SIAM), Springer-Verlag, Y&Y and the American Mathematical Society, but until not long ago there were no free Type 1 versions of other “CM look-alike” fonts available, which limited their usage in PDF and PostScript target document formats. The CM-Super fonts were developed to cover this gap.

Such a conversion from METAFONT to Type 1 fall into one of two general categories. First, base the conversion on analytic study of the METAFONT output (which may include “patching” the METAFONT program, analyzing the output of METAPOST, or similar approaches). Such an approach can give (potentially) the most accurate results, but I did not choose to use it, since it is much harder to develop. (There are some commercial translators of METAFONT to Type 1 which I did not evaluate, mainly due to their “closedness”, which I wanted to avoid.) The second, straightforward, approach is based on tracing the high-resolution bitmaps generated by METAFONT, and thus obtaining outline versions of the fonts.

I was considering several approaches to converting the METAFONT fonts to Type 1 format — it seems that the GNU fontutils package developed by Karl Berry is capable of this, but it had some problems with glyph positioning. Since the appearance of \TeX trace, it became very easy to do this. \TeX trace is a free automatic converter of METAFONT fonts to Type 1 format, developed by Péter Szabó. It is based on Autotrace by Martin Weber.

And, as far as I know, some code was taken from the GNU fontutils package.

It would be possible to simply generate Type 1 variants of all needed METAFONT fonts (each containing no more than 256 glyphs), but this will have disadvantages: the total number of files will be truly vast (several thousands), and also the total size of these fonts will be much bigger than necessary, because of glyph duplication: Latin letters (and some other characters and glyphs) will be present in more than one font, e.g., the fonts `ecrm1000` (T1 encoding), `larm1000` (T2A), `lbrm1000` (T2B), `lcrm1000` (T2C), `fcr10` (T4), and `vnmr1000` (T5) are all of the same family and shape (Computer Modern Roman) and design size (10pt), and will thus duplicate at least the basic Latin alphabet. Given the total number of font files (several thousands), such duplication will give significant overhead.

Thus, it seems natural to try to combine all fonts which have the same font family, shape and design size, and differ only by font encoding (set of supported glyphs) into *one* SuperFont (the name comes from Karl Berry’s Fontname package), which will contain all the *unique* glyphs from these fonts just once. Then it is possible to create map files for `dvips`, `pdftex` or other programs which will re-encode the big Super-fonts into specific 256-character font encodings, selecting the glyphs needed for a particular font encoding.

Such an approach is undertaken in the CM-Super font package: each Type 1 SuperFont includes all the glyphs from several METAFONT fonts which have the same font family, font shape and design size. Currently, only text font encodings are supported. The name “super” does not imply that this font collection contains Type 1 variants of all existing CM look-alike fonts (e.g., there are no glyphs from math fonts in CM-Super), but that each font in this collection is a SuperFont supporting many glyphs, languages, and scripts. These SuperFonts can be used with other applications as well as \TeX .

2 Font encodings, families, shapes, and names

The New Font Selection Scheme (NFSS) in \LaTeX serves as a very good classifying mechanism in the chaos of various fonts: each font is classified by its encoding, family, shape and design size. The font encoding defines the set (and order) of glyphs which are contained in a font. Currently, the CM-Super font collection supports glyphs from the following \LaTeX font encodings: T1, TS1, T2A, T2B, T2C, X2. In the near future I’ll try to add support for these encodings as well: T3, T4, T5, LGR, and others (e.g.,

additional Cyrillic and Latin glyphs which are not already present in supported encodings).

The original CM fonts used file names such as `cmr10`, `cmbx12`, etc., while their EC analogues use font names consisting of four letters, the first two of which are always “ec” while the second two denote font family and shape: `ecrm1000`, `ecbx1200`, etc. Design sizes are specified differently in the two families: EC fonts use a 4-digit scheme, while CM fonts use two digits. I’ve chosen to use font names of SuperFonts which follow the scheme used in EC fonts (with “sf” instead of “ec”, “tc”, etc.), since the majority of METAFONT fonts included into the CM-Super package follow this naming scheme (EC, TC, LH).

Families and shapes currently supported (the README file in the CM-Super distribution contains a more detailed list):

1. 29 font shapes supported by EC/TC fonts. Each font shape comes in 14 font design sizes ranging from 5pt to 35.83pt (or 11 design sizes for type-writer font shapes ranging from 8pt to 35.83pt), giving $23 \cdot 14 + 6 \cdot 11 = 388$ font files;
2. 13 font shapes for `Sl \TeX` (each comes in one design size);
3. 14 fonts from Computer Modern Concrete family (font file names correspond to the scheme used in EC Concrete fonts, again with “sf” instead of “ec”);
4. 19 fonts from Computer Modern Bright family (font file names correspond to the scheme used in European Computer Modern Bright fonts).

The total number of Type 1 font files included in the CM-Super package is 434.

Each Type 1 font contains glyphs from several METAFONT fonts with the same font shape and design size. For example, `sfrm1000.pfb` combines unique glyphs from the following METAFONT fonts: `ecrm1000.mf`, `tcrm1000.mf`, `larm1000.mf`, `lbrm1000.mf`, `lcrm1000.mf`, `rxrm1000.mf` (for encodings T1, TS1, T2A, T2B, T2C, X2, respectively). In a future version, this font will also include glyphs from `tipa10.mf`, `fcr10.mf`, `vnmr1000.mf`, `grmn1000` (for font encodings T3, T4, T5, LGR).

Caps and small caps fonts do not include glyphs from TC fonts (TS1 font encoding), since there are no small caps TC fonts (but it may make sense to include glyphs from the TS1 encoded fonts into these fonts, for completeness, by duplicating glyphs from the corresponding non-smallcaps font shapes).

Our approach provides big savings: if we were making separate Type 1 fonts for each of the above mentioned METAFONT fonts, we would have 256 ·

5 + 128 = 1408 glyphs from the five METAFONT sources mentioned; but `sfrm1000.pfb` contains only 585 unique glyphs, which includes some glyphs which were added for the sake of completeness.

The CM-Super fonts come with AFM and INF files and are thus usable with non- \TeX -related applications as multilingual fonts.

3 Details on making the CM-Super fonts

As mentioned above, CM-Super fonts were made by tracing the high resolution bitmaps created by METAFONT. Below is a more detailed description of this process.

First, I created map files which contain each font shape per line corresponding to each METAFONT font which is to be traced. Each map file contains fonts for some particular font encoding (T1, TS1, T2A, T2B, T2C, X2). Also there is a map file for additional glyphs (ellipsis and alternative variant of the sharp “s”).

Also, for each supported font encoding, I created the encoding vector with standard glyph names following the Adobe Glyph List (AGL) conventions (a few glyphs absent from the AGL and even from Unicode were named arbitrarily).

Now we can run the script `traceall.sh` from \TeX trace to make Type 1 fonts from each METAFONT font, using the map files and encoding vectors made in the previous step.

Now we need to combine these small Type 1 font files into Super-fonts. First, we “disassemble” them using `t1disasm` from the `t1utils` package, to convert them into plain text format, convenient for processing. Then we can run a script which combines unique glyphs from the fonts with the same font shape and design size into one (disassembled) Type 1 font. The script not only check for unique names, but also checks that the glyphs with the same names from different fonts (e.g., the Latin letters) are represented identically.

Since we combined several Type 1 fonts into one big font, the `FontBBox` parameter needs to be fixed. This is done using the `pf2afm` PostScript program from the Ghostscript distribution.

We also fix the `isFixedPitch`, `ItalicAngle`, and `Weight` values using a script, since \TeX trace doesn’t set them right. The value of `ItalicAngle` is extracted from the TFM file using the `Font::TFM` perl module.

To make the resulting fonts smaller, we make some cleanups, like removing redundant “0 hmoveto” from glyph charstrings dictionary, setting the default font encoding to `StandardEncoding`, setting the value of `lenIV` parameter in the private dictionary to 0, etc.

At this point, we have “raw” Type 1 fonts which should be optimized (which is done later).

Now we gather information contained in the TFM files (which are generated by METAFONT), and apply it to PFB files, and also create AFM font metric files. This in itself involves several steps:

- First, we extract kern values from each of the TFM files, using the script based on the `Font::TFM` perl module, and generate the textual representation of kerns which is used in AFM file format.

Then we combine the kern values from individual TFM files which correspond to the same font shape and design size (but differ by font encoding) into one big kern table. While doing such combining, we always check that there no inconsistent kerns (for the same glyph pairs) in different fonts. A few such inconsistencies were indeed found.

- Now we’d like to make the glyph widths in the PFB files use precise (non-integer) values which better match the values in the TFM files. These widths are generated using the best approximation (based on continued fractions) with the denominator not exceeding 107 to fit in 1 byte in CharStrings (giving space economy), and are stored using the `div` operator in CharStrings. Apparently, such a subtle technique was used first in the BSR/Y&Y CM fonts.

Again, we combine the exact glyph widths obtained from different METAFONT fonts into glyph widths for SuperFonts. And finally, we fix the `hsbw` operators in the PFB files to use the calculated precise glyph widths.

- We extract font parameters (`fontdimen` values) from individual METAFONT fonts. They are converted to the corresponding values like `Ascender`, `Descender`, `XHeight`, `CapHeight` which are stored in the AFM files.
- The ligatures contained in the TFM files are also extracted and put into the AFM files.
- Finally, glyph bounding boxes are extracted from the PFB files using `pf2afm`, and all the pieces obtained in previous steps are combined into final AFM files for the CM-Super fonts.

Now it’s time to optimize the PFB files, since they contain a lot of “junk” control points and do not follow the rules which should be obeyed in good Type 1 fonts. This is the only step which was performed using commercial software: FontLab, but now it is also possible to use `PFAedit`¹ which may give comparable

¹ This program has since been renamed to `FontForge` and is available from <http://fontforge.sourceforge.net/>. *Ed.*

results. The optimization consists of adding nodes at extremes, removing overlaps in contours, optimizing contours (removing unnecessary control points, simplifying the contours), and also autohinting. We intentionally used only automatic optimization (in packet mode, without human interaction). The aim was to use *totally* automatic conversion of METAFONT fonts to Type 1 format, automatic optimization and hinting, with the best achievable quality of final Type 1 fonts, to be able to re-generate the fonts if necessary (e.g., when a new version of original METAFONT fonts is released). Undoubtedly, there is room for improvement of this approach, which we will attempt in future versions of the fonts. After passing the fonts through FontLab, we perform some cleanup again by disassembling, processing and re-assembling back the fonts.

Finally, we create the INF files using a simple script.

The above description may seem a bit complicated at the first glance, but all steps are performed by running a few simple Perl scripts. Some of them may appear to have wider application, so I'll put them into the distribution at some point.

4 Related works

Some packages have been developed which may be useful in conjunction with the CM-Super font package. First, the `type1ec` package which is analogous to the existing `type1cm` package and makes the EC-based fonts available at any size (as opposed to the set of the standard font sizes defined in the default \LaTeX font definition files). This will work efficiently since the CM-Super fonts are vector fonts and will be preloaded only at the few design sizes scaled appropriately.

The second package, recently developed, is the `cmap` package, to be used with `pdf \LaTeX` , which “hooks” into the low-level \LaTeX font pickup command to preload the CMap resources for the fonts which are used in the document. This adds searching and copying capabilities to the PDF files, by defining the “meaning” (Unicode values) for the

glyphs used in the document. At the moment this works only for Type 1 fonts, since `pdf \TeX` had been ignoring the `\pdffontattr` command for the Type 3 fonts, but this was fixed recently and thus it will soon be possible to make the files “searchable” even if the document uses some bitmap fonts.

Both these packages are available on CTAN.

5 Future work

There are many possibilities for improving and extending the CM-Super package. Unfortunately, I haven't had much time recently to work on it, but hopefully I'll move forward soon. Some of the ideas are written in the `TODO` file in the distribution. Most important are:

- cover more fonts—support some other \LaTeX font encodings: T3 (TIPA), T4 (African writings, FC fonts), T5 (Vietnamese fonts), LGR Greek font encoding (CB-Greek fonts), . . . ;
- make the fonts even more efficient (smaller) by using techniques similar to the ones described in Thanh's paper (putting glyphs and accents into subroutines, and constructing accented glyphs from them instead of putting the whole definition of accented glyph into the font);
- make an ultimate step forward to improve the quality of glyph shapes by using analytic transformation rather than tracing.

6 Acknowledgements

I am grateful to the following people who made this work possible: Hàn Thế Thành who inspired me to make a joint talk at TUG 2003 and who gave me many ideas for improving the package and helped to understand some mechanisms (in particular, how to add the CMap entries into font dictionaries into PDF files); Peter Szabo and Martin Weber and Karl Berry who made it possible to easily generate Type 1 fonts from bitmaps; Yuri Yarmola from FontLab for providing a copy of FontLab; William Adams, Wendy McKay, Robin Laakso and Patricia Monohon for the help with corrections of this article and friendly help before and during this conference.

Making Type 1 fonts for Vietnamese

Hàn Thế Thành
 University of Education
 Ho Chi Minh City
 Vietnam
 hanthethanh@gmx.net

Abstract

In this article I describe how I made the VNR fonts and converted them to Type 1 format. VNR is the Vietnamese version of the CMR fonts, written in METAFONT based on CMR and EC font sources. Conversion to Type 1 format was done based on the Type 1 version of CMR produced by Blue Sky, with the help of MetaFog, FMP, FontLab and a lot of hacking in VNR sources and Bash and Perl scripting. The result is a set of Type 1 fonts that is similar to the Blue Sky fonts, but also provide Vietnamese letters with the same quality of outlines and hints.

Vietnamese letters and VNR fonts

Vietnamese is written with Latin letters and a few more accents in a system called

Quốc Ngữ

(which can be translated to English as “national language”) developed by the Portuguese missionary Alexander Rhodes. What separates Vietnamese from other languages typeset with Latin characters is that some letters in Vietnamese can have two accents. The total number of accented letters in Vietnamese (including uppercase and lowercase letters) is 134. Table 1 lists all lowercase Vietnamese letters.

a	á	ạ	à	ả	ã	o	ó	ọ	ò	ỏ	õ
ă	ắ	ặ	ằ	ẳ	ẫ	ô	ố	ộ	ồ	ỗ	ỗ
â	ấ	ậ	ầ	ẩ	ẫ	ơ	ớ	ợ	ờ	ở	ỡ
e	é	ẹ	è	ẻ	ễ	u	ú	ụ	ù	ủ	ũ
ê	ế	ệ	ề	ể	ễ	ư	ứ	ự	ừ	ử	ữ
ì	í	ị	ì	ỉ	ĩ	y	ý	ỵ	ỳ	ỷ	ỹ
						đ					

Table 1: List of all Vietnamese lowercase letters.

Vietnamese accents can be divided into three kinds of diacritic marks: tone, vowel and consonant. Table 2 shows them all with examples.

As Vietnamese letters are identical to Latin letters, it is natural to write VNR as a set of META-

Diacritic mark	Example
<i>Vowel</i>	
breve	băn khoăn
circumflex	hôm nay
horn	Qui Nhơn
<i>Tone</i>	
acute	Lái Thiêu
grave	Bình Dương
hook above	Thủ Đức
tilde	dĩ vãng
dot below	học tập
<i>Consonant</i>	
stroke	đã đời

Table 2: List of all Vietnamese diacritic marks.

FONT files which compose the Vietnamese letters from English letters in CMR sources and appropriate accents. There were several works on this topic prior to VNR. The best among them was the package `vncmr` by Werner Lemberg, who also created some basic macro support for typesetting Vietnamese in \LaTeX and plain \TeX .

As I learnt more about \TeX and METAFONT, I set out to create new VNR fonts, as I was not entirely happy with the accent shapes and positioning in previous packages. I borrowed many ideas from the `vncmr` package and other CMR-based fonts, like the Czech and Polish version of CMR fonts. It took about 2 years until the first version was released and used in practice.

In the beginning I wrote VNR fonts based on CMR sources. Later, Werner Lemberg and Vladimir

Volovich convinced me that it would be better to switch to EC sources instead of CMR sources. The two main reasons:

1. it would be easier to include Vietnamese letters in CM-Super fonts;
2. it would fix some problems in VNR (mainly with encoding and some missing glyphs).

So I changed VNR to be based on EC, which of course introduced new problems. The most noticeable is with naming: Should VNR fonts be named with an EC-like naming convention (`vnrm1000.mf`) or a CMR-like naming convention (`vnr10.mf`)? In the end I decided to support both, but the sources (and thus the glyph shapes) are EC-based. Why not drop one of them? Because:

1. I wanted to support EC naming, so Vladimir could include Vietnamese letters in CM-Super;
2. I wanted to support CMR naming, because I wanted to use the Type 1 version of CMR fonts. Also, there are many packages which depend on CMR-style names (such as `Texinfo`).

I would like to point out that I am not a type designer and therefore the aesthetic aspect of Vietnamese letters (regarding accent shapes and positioning) is open to discussion. What I have done is heavily based on what I learnt from other Vietnamese fonts available to me, from the typography of other languages (mainly Czech and Polish), and from comments from various people. It's not to say that I take no responsibility for VNR fonts. If something looks bad, it is of course my fault.

It's good that I can use Vietnamese with \TeX , but I want PDF

PK fonts just look ugly in Acrobat Reader. So I needed Type 1 fonts. My first attempt was to create a set of virtual fonts for Vietnamese, which refer to CMR fonts. Unfortunately, the output simply doesn't look good, as many needed accents are not available in CMR and must be substituted by some other glyphs.

I also tried to use \TeX trace to generate Type 1 versions of VNR fonts (using both EC and CMR naming). The result is useable, but the size is too large and the quality of auto-generated outlines and hints cannot be compared to Blue Sky fonts. It seemed a pity to me that I could not use the high-quality outlines and hints of English letters in the Blue Sky fonts. So I looked for another method to convert VNR fonts to Type 1 format.

Reuse is a good idea

The main idea of efficient generating Type 1 format for VNR fonts is simple:

1. Take the accents from METAFONT sources and convert them to Type 1 format;
2. compose those accents and the English letters from Blue Sky fonts to get accented letters.

Thus nothing is new; each step is just a reuse of existing data.

Use the right tool for each task

The two steps mentioned above can be done properly, given that we have the right tool for each.

Converting the accents to Type 1 format This can be done *very quickly* using \TeX trace, or *nearly perfectly* using METAPOST and MetaFog. As I had an evaluation copy of MetaFog and I wanted the result to be as good as possible, I chose the latter.

In this step, some glyphs that are needed in T5 (the Vietnamese \TeX encoding) but are missing in the Blue Sky fonts must also be converted. The number of glyphs needed for each font is 47, of which 22 are Vietnamese accents and letters. Table 3 shows all the glyphs that need to be converted to Type 1 format. The `v1` and `vu` prefixes stand for "Vietnamese lowercase" resp. "Vietnamese uppercase". At the moment they look identical; however, they may be changed.

Many of the additional glyphs are available in the Blue Sky fonts already, but their availability is not consistent. Each of the additional glyphs listed here is missing at least in some Blue Sky font. To get rid of this headache, I chose to convert them all.

47 glyphs is quite a large number for each font, as the total number of glyphs to be converted is 2585 (47×55). Fortunately, not all of them required manual correction, and the additional glyphs need to be converted only once, given that EC sources will not be changed. So, if I change the VNR sources, I will have to re-convert only those 22 Vietnamese accents and letters.

When MetaFog finished and we had the outlines of the needed glyphs, hints for accents were auto-generated using FontLab.

Several papers have been published on this topic so I will not repeat it all here. (*See References, Ed.*)

In short, using METAPOST and MetaFog gives the best result, but a lot of manual work is required. That's one reason why I didn't convert all Vietnamese glyphs but only those that are *truly* necessary. Why regenerate the English letters when they already exist in Blue Sky fonts at the high quality?

Vietnamese accents and letters	Additional glyphs
dotbelow	quotesinglbase
hookabove	guilsinglleft
vlgrave	guilsinglright
vlacute	quotedblleft
vlcircumflex	quotedblright
vtilde	quotedblbase
vldotbelow	guillemotleft
vlbreve	guillemotright
vlhookabove	endash
vugrave	emdash
vuacute	cwm
vucircumflex	zeroinferior
vutilde	uni2423
vudotbelow	quotedbl
vubreve	dollar
vuhookabove	less
Ohorn	greater
Uhorn	backslash
ohorn	asciicircum
uhorn	underscore
Dcroat	braceleft
dcroat	bar
	braceright
	asciitilde
	sfthyphen

Table 3: List of glyphs that need to be converted from METAFONT to Type 1 using MetaFog.

Composing the accented letters Composing the base letters and accents to create accented letters has two advantages:

1. the base part of the letter will have the outlines and hints at the same quality as the Blue Sky fonts;
2. the final size of fonts will be considerably reduced, as base letters and accents will be put into subroutines in the final Type 1 fonts. This way each glyph is included only once (again, reuse is good).

The tool used in this step was FMP (Font Manipulation Package) from Y&Y. This package contains a tool that can create composite glyphs from existing glyphs in a font. The question is how to

place an accent over a letter *exactly* like the VNR sources do. The solution is simple: I added some hooks into VNR sources, so information about accent positioning is written to a log file. Then I wrote some Perl scripts to extract the data from the log file and use them with the composite tool from FMP.

What about the result?

A rough comparison on average font size gave the following result:

Blue Sky	25 KB
Type 1 VNR (T _E Xtrace)	70 KB
Type 1 VNR (as described herein)	40 KB

Compactness is gained thanks to FMP, which puts all accents and English letters into subroutines so they can be reused without duplication, as mentioned above.

Regarding the quality of outlines and hints of each accented letter:

- The base part has the the same outlines and hints as in Blue Sky fonts;
- the accent part has the outlines produced by METAPOST (and MetaFog), with hints auto-generated by FontLab.

As the accent shapes are quite simple, the hints auto-generated by FontLab are quite reasonable. It's very hard to do better without a great deal of experience in manual hinting.

A sample of font `vnr10` is shown in figure 1.

Any known problems?

Yes: The VNR sources are EC-based, while the English letters from Blue Sky are Computer Modern-based. So it is possible that the Type 1 version of VNR fonts will have slightly different glyph shapes from their counterpart generated by METAFONT. I did some quick comparisons: The difference is only visible at very high resolution, and can be tolerated in my opinion. Hopefully nobody will mind, or even notice.

Related works

I applied a similar method to add Vietnamese letters into the URW fonts. The result is a package I called URWVN. Technically, the URWVN fonts are very similar to VNR:

1. they are very compact; the average size of the URWVN fonts is 35 KB;
2. the base character of accented letters has the same outlines and hints as in URW fonts;

Chí Phèo

Hắn vừa đi vừa chửi. Bao giờ cũng thế, cứ rượu xong là hắn chửi. Bắt đầu chửi trời. Có hề gì? Trời có của riêng nhà nào? Rồi hắn chửi đời. Thế cũng chẳng sao: đời là tất cả nhưng chẳng là ai. Tức mình hắn chửi ngay tất cả làng Vũ Đại. Nhưng cả làng Vũ Đại ai cũng nhủ, “*Chắc nó trừ mình ra!*” Không ai lên tiếng cả. Tức thật! Ồ! Thế này thì tức thật! Tức chết đi được mất! Đã thế, hắn phải chửi cha đứa nào không chửi nhau với hắn. Nhưng cũng không ai ra điều. Mẹ kiếp! Thế thì có phí rượu không? Thế thì có khổ hắn không? Không biết đứa chết mẹ nào đẻ ra thân hắn cho hắn khổ đến nông nổi này? A ha! Phải đấy, hắn cứ thế mà chửi, hắn chửi đứa chết mẹ nào đẻ ra thân hắn, đẻ ra cái thằng Chí Phèo! Hắn nghiến răng vào mà chửi cái đứa đã đẻ ra Chí Phèo. Nhưng mà biết đứa nào đã đẻ ra Chí Phèo? Có trời mà biết! Hắn không biết, cả làng Vũ Đại cũng không ai biết. . .

Figure 1: A text sample of vnr10.

3. the accent part of accented letters was drawn manually,¹ with hints auto-generated by FontLab.

The aesthetic aspect is open to discussion, as I am not a type designer. Comments or suggestions are very welcome.

A sample of Times Roman is shown in figure 2.

Future works

At the moment, VNR and URWVN have only one version for each accent. Therefore uppercase and lowercase forms of a letter have the same accent, e.g. `aacute` and `Aacute` have the same shape for the accent. This should be improved by introducing two separate versions for each accent, one for uppercase and one for lowercase. This is important for Vietnamese, as many letters have double accents, which causes such letters to be very tall. Uppercase letters should have wider and lower accents than their lowercase counterparts.

The METAFONT sources of VNR could also be improved to make the accents look better, especially for sans serif fonts.

¹ More precisely, they were derived from existing glyphs in URW fonts with manual modification.

Files

- The VNR fonts and `vntex` are available at <http://vinux.sourceforge.net/vntex>
- The URWVN fonts are available at <http://vinux.sourceforge.net/urwvn>
- Samples of VNR fonts are available at <http://vinux.sourceforge.net/vntex/vnfontsample.pdf.gz>
- Samples of URWVN fonts are available at <http://vinux.sourceforge.net/urwvn/urwvnsample.pdf.bz2>

Acknowledgments

I would like to thank the following people, without whom my work would never be finished:

- Werner Lemberg for creating the `vncmr` package from which I learnt a lot;
- Tom Kacvinsky for donating a copy of FMP;
- Richard Kinch for creating MetaFog and donating an evaluation copy;
- Paul Watry for donating a copy of FontLab;
- and William Adams for correcting this paper.

Chí Phèo

Hắn vừa đi vừa chửi. Bao giờ cũng thế, cứ rượu xong là hắn chửi. Bắt đầu chửi trời. Có hề gì? Trời có của riêng nhà nào? Rồi hắn chửi đời. Thế cũng chẳng sao: đời là tất cả nhưng chẳng là ai. Tức mình hắn chửi ngay tất cả làng Vũ Đại. Nhưng cả làng Vũ Đại ai cũng nhủ, “*Chắc nó trừ mình ra!*” Không ai lên tiếng cả. Tức thật! Ô! Thế này thì tức thật! Tức chết đi được mất! Đã thế, hắn phải chửi cha đứa nào không chửi nhau với hắn. Nhưng cũng không ai ra điều. Mẹ kiếp! Thế thì có phí rượu không? Thế thì có khổ hắn không? Không biết đứa chết mẹ nào để ra thân hắn cho hắn khổ đến nông nỗi này? A ha! Phải đấy, hắn cứ thế mà chửi, hắn chửi đứa chết mẹ nào để ra thân hắn, để ra cái thằng Chí Phèo! Hắn nghiến răng vào mà chửi cái đứa đã đẻ ra Chí Phèo. Nhưng mà biết đứa nào đã đẻ ra Chí Phèo? Có trời mà biết! Hắn không biết, cả làng Vũ Đại cũng không ai biết. . .

Figure 2: A text sample of Vn Nimbus Roman No9 L Regular.

A Step-by-step description of generating Type 1 VNR fonts

As an example, `vnr10.pfb` was generated as follows:

1. Run METAPOST on `vnr10.mf` to write information about accent positioning into a log file; the result of this step is `vnr10.log`, containing precise positioning of accents.
2. Run Perl scripts to extract the needed information from the log file and convert it to a format suitable for use with FMP. The result are two files, `vnr10.ac1` and `vnr10.ac2`.

The file `vnr10.ac1` contains accent positioning instructions for letters with single accent. Those instructions are similar to the `CC` commands often found in AFM files; they look like:

```
CC acute 2 ; PCC a 0 0 ;
                                PCC vlcute 146 495
CC breve 2 ; PCC a 0 0 ;
                                PCC vlbreve 85 495
```

`vnr10.ac2` contains accent positioning instructions for letters with double accents. As the composite tool from FMP does not allow compositing a letter with two accents, this must be accomplished in two passes. `vnr10.ac1` is used in the first pass and `vnr10.ac2` in the second pass. The instructions in `vnr10.ac2` look like:

```
CC abreveacute 2 ; PCC abreve 0 0 ;
                                PCC vlcute 205 631
```

To make it clear, a character with double accents is constructed as follows: In the first pass, the base letter and the first accent is composed to create a new glyph. In the second pass, this new glyph is composed again with the second accent to get the double-accented letter.

3. Run METAPOST on `vnr10.mf` to generate PS outlines of glyphs that need to be converted to Type 1 format; those are listed in table 3.
4. Run MetaFog on the result of the previous step to convert them to Type 1 format (without hinting). If some glyphs were incorrectly converted, manual intervention is needed in this step.
5. Autohint the font generated by MetaFog in the previous step, using FontLab. The result is a font named `vnr10-t5supp.pfa`.
6. Process the file `cmr10.pfb` from the Blue Sky fonts with a Perl script which removes the `div` operator from glyph descriptions. This is necessary because some tools from the FMP package don't like fonts containing this operator. The result is `cmr10.pfa` (FMP tools work with PFA format only).
7. Run a Bash script that uses the FMP tools to compose `vnr10.pfb` from the constituents

`cmr10.pfa`, `vnr10-t5supp.pfa`, `vnr10.ac1` and `vnr10.ac2`. This script does the following:

- remove unwanted glyphs from `cmr10.pfa`;
 - merge `cmr10.pfa` and `vnr10-t5supp.pfa` to get all the glyphs needed for composition in a single font (FMP requires this);
 - compose all letters with single accents, according to instructions in `vnr10.ac1`;
 - compose all letters with double accents, according to instructions in `vnr10.ac2`.
8. Post-process the resulting `vnr10.pfa` to add a few final things, such as the UniqueID, encoding, font name and the like.

B Step-by-step description of generating URWVN fonts

The process is similar to the case of VNR fonts. However, there are two main differences:

1. the accents were drawn manually instead of being generated from METAFONT sources;
2. accent positioning was also done “manually” instead of being generated from the METAFONT sources.

As an example, `utmr8v.pfb` (`8v` is the abbreviation suggested by `fontname` for the Vietnamese encoding) was generated as follows.

1. Open `utmr8a.pfa` in FontLab and add the Vietnamese accents, plus some other missing letters: `uhorn`, `Uhorn`, `ohorn`, `Ohorn`.
2. Use the program `a2ac` (written by Petr Olšák, with some modifications to fit my needs) to create a “rich” AFM file where composite instructions look like:

```
CC 0circumflex 2 ; PCC 0 0 0 ;
      PCC vucircumflex 194 181 ;
CC 0circumflexacute 3 ; PCC 0 0 0 ;
      PCC vucircumflex 194 181 ;
      PCC vuacute 116 339 ;
```

The reason to use `a2ac` instead of writing these instructions from scratch is that `a2ac`

allows generating CC instructions in a clean, systematic and efficient way; it also allows automatic generation of kerning information for newly composed glyphs. The functionality of `a2ac` is very similar to the famous `fontinst` (with some limitations).

3. Run a script that takes the above AFM file, generate a virtual font and run `TeX` (`pdfTeX`) on a test file to display how composited glyphs look. The purpose of this step is to check quickly whether accent positioning is already good, or if it needs further improvements. If so, we come back to the previous step. This allows a very efficient edit-compile-test-edit cycle.

Why not do accent positioning in FontLab instead of using `a2ac` and all this hacking? Because the latter defines accent positioning in a precise, consistent and more efficient way. If I did it in FontLab, there would be some disadvantages:

- (a) it would take longer;
 - (b) accents could not be placed consistently;²
 - (c) if an accent is changed, all must be done again.
4. When the accent positioning is reasonable, run a script that uses the FMP tools to compose accented letters, as in case of the VNR fonts.
 5. Post-process the result to fix the same administrative things.

C References

Richard Kinch, *TUGboat* 16(3) (1995), “MetaFog: Converting METAFONT Shapes to Contours”. <http://www.tug.org/TUGboat/Articles/tb16-3/tb48kinc.pdf>

Taco Hoekwater, *Bijlage* 26, MAPS 20 (1995), “Generating Type 1 Fonts from METAFONT Sources”. http://www.ntg.nl/maps/pdf/20_39.pdf

² Well, they could, but at a very high price.

Chinese character synthesis using METAPOST

Candy L.K. Yiu, Wai Wong
Department of Computer Science,
Hong Kong Baptist University
[candyiu,wwong]@comp.hkbu.edu.hk

Abstract

A serious problem in Chinese information exchange in this rapidly advancing Internet time is the sheer quantity of characters. Commonly used character encoding systems cannot include all characters, and often fonts do not contain all characters either. In professional and scholarly documents, these unencoded characters are quite common. This situation hinders the development of information exchange because special care has to be taken to handle these characters, such as embedding the character as an image. This paper describes our attempt towards solving the problem. Our approach utilizes the intrinsic characteristic of Chinese characters, that is, each character is formed by combining strokes and radicals. We defined a Chinese character description language named *HanGlyph*, to capture the topological relation of the strokes in a character. We are developing a Chinese Character Synthesis System CCSS which transforms *HanGlyph* descriptions into graphical representations. A large part of the CCSS is implemented in METAPOST.

1 Introduction

The rapid advancement of the Internet and the Web provides an effective means of information exchange. However, there is a very serious problem in exchanging Chinese documents: the number of Chinese characters that now exist or have ever existed is unknown. Furthermore, new characters are continually being created. Therefore, no character set can encode *all* Chinese characters.

Even if a character set could encode all Chinese characters, it is very expensive to create Chinese fonts using typical methods and a fairly large number of Chinese characters would be so rarely used that the expense would be very difficult to justify.

One possible solution to this problem is to create an unencoded character according to its composition of strokes and radicals. Several experiments along this line were attempted in the past, but none were very successful. The key reason is that the composition of the strokes and radicals is very complex, and the previous attempts did not effectively divide and resolve the complexity. The section on related works gives a brief survey of some previous attempts.

Our approach to Chinese character synthesis resolves the complexity in two ways. First, we defined a high-level Chinese character description language, *HanGlyph*. It captures the abstract and topological relation of the strokes. Thus, the character descrip-

tion is compact and can be targeted to a variety of rendering styles. The section on the *HanGlyph* Chinese character description language describes the language in more detail. Secondly, we use METAPOST as our rendering engine to take advantage of its meta-ness and the ability of specifying paths and solving linear equations.

The *HanGlyph* language is defined based on many studies of Chinese characters. The section on the structure of Chinese characters explains the basic structure of Chinese characters for the benefit of readers who are not familiar with them. *HanGlyph* defines 41 basic strokes, 5 operators and a set of relations. A character is built by combining strokes using the operators recursively. *HanGlyph* allows the user to define macros to represent a stroke cluster which can then be re-used in building more complex characters.

The CCSS (stands for Chinese Character Synthesis System) takes *HanGlyph* expressions and renders the characters. It can be divided into three parts: a front-end to translate *HanGlyph* expressions into METAPOST programs, a set of primitive strokes, and a library of METAPOST macros to implement the operators and relations. By varying the parameters to these macros, or redefining the basic stroke macros, Chinese characters in different styles can be formed. Thus, it can create a variety of different fonts from the same *HanGlyph* description.

2 The structure of Chinese characters

Chinese characters, or *hanzi*, have their roots in a very long history. A large body of literature on the study of the written form of Chinese language, dating from as early as more than 2000 years ago (during the *Han* dynasty) up to now, is available. The written form of Indo-European languages consists of around 30 characters. Words are formed using these characters in a linear fashion. In contrast, written Chinese language is denoted by tens of thousands of *hanzi*. The exact number of *hanzi* that have ever existed can never be known.

Many studies have pointed out that each Chinese character is composed from strokes. The number of strokes in a character varies from one for the simplest, up to around 50 for the most complex. Unlike the linear composition of words from characters in Indo-European languages, the arrangement of the strokes in *hanzi* is two-dimensional.

According to the convention of writing Chinese characters, a stroke is a continuous movement of the brush over the writing surface without being lifted up. It is commonly agreed that there are five basic strokes: (橫 *héng*¹), (豎 *shù*), (撇 *piě*), (捺 *nà*) and (點 *diǎn*).

In practice, each of these basic strokes has some variations depending on the position in a character. For example, the stroke 撇 can have two variations: (平撇 *píngpiě*) (as the top stroke in 千) and (豎撇 *shùpiě*) (as the leftmost stroke in 月). In addition, a number of combinations of these basic movements are considered as strokes because they are connected in a natural way in writing. For example, a (橫) followed by a (撇) is a single stroke called (橫折撇 *hèngzhépiě*). Modern studies of Chinese characters [1, 9] identified a small set of around 40 strokes as the basic elements of *hanzi*.

Although the arrangements of strokes to form a *hanzi* is very complex, there are some rules that guide the formation of characters. Further, some stroke arrangements are relatively stable and appear in many characters. Some of these arrangements are themselves *hanzi*, for example, 日月; some of them are known as *radicals* which are used in Chinese dictionaries to index characters, for example, 彳 匚 卩. There are some relatively stable arrangements that are not *hanzi* themselves, nor radicals, but appear in many characters. We will use the term *components* to refer to all these kinds of stroke arrangements,

while we use a more general term *stroke clusters* to refer to any arrangements of several strokes.

Except for a small number of very simple characters, such as 人, 二, 十, which cannot be divided into component parts, all *hanzi* can be considered as compositions of certain components. The ways of composing *hanzi* from components are known as the *structure* of the character. Many studies, such as [2] and [8], have identified around 10 different types of structures if one considers how to compose a character from only two components. This does not place serious restrictions, because the composition process can be performed recursively. Figure 1 illustrates the commonly used structures.

3 Related works

Based on the studies of Chinese characters, several attempts have been carried out to create *hanzi* from a structural composition approach.

Toshiyuki *et al* [10] proposed a way of describing Chinese characters using sub-patterns. In principle, their method is similar to the approach of *Han-Glyph* because the underlying theory of character structure is intrinsic to all Chinese characters.

Dong [11] and Fan [5] reported their work on the development of a Chinese character design system which took a parametric approach to create characters in different styles. Lim and Kim [7] developed a system for designing Oriental character fonts by composing stroke elements.

Inspired by the success of METAFONT [6] in creating latin character fonts, Hobby and Gu [3] attempted to generate Chinese characters of different styles using METAFONT. A small set of strokes were defined in METAFONT. A small set of radicals were then defined as METAFONT macros by using the strokes. Characters can then be specified as METAFONT programs using these macros as building blocks. By varying some parameters governing the shapes of the strokes, fonts of different styles can be generated. However, the research was not conclusive because they only generated fonts with a very small character set (128 characters).

Another attempt similar to Hobby and Gu was done by Hosek [4] who aimed at generating *hanzi* from a small sets of components.

A common theme of the works mentioned is the difficulty of handling the complexity of the structures and the numerousness of characters. Our approach handles the complexity by using an abstract description and a layered CCSS to decompose the complexity into several sub-problems. On the *Han-Glyph* level, we consider strokes as abstract objects.

¹ The word *héng* following the *hanzi* name of the stroke is in *pinyin*, a phonetic transcription of Chinese characters. We hope these *pinyin* transcriptions can help readers who do not know Chinese to pronounce the names of the strokes.

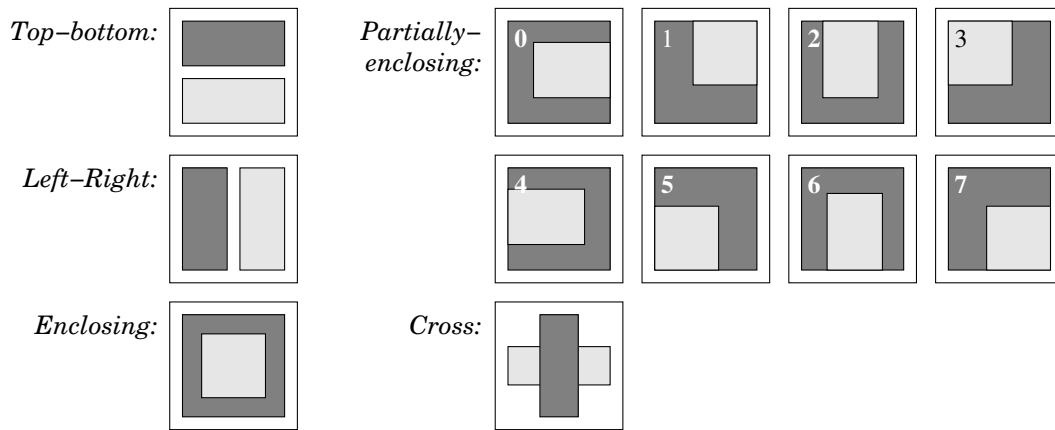


Figure 1: The basic structure of *hanzi*.

We need to specify only the relative positions between these abstract objects. On a lower level, we can work out the outline of the strokes and fine tune the positions.

Another theme of the works mentioned is that they are mainly aimed at the design and generation of character fonts. Our approach can certainly be applied in font generation. However, a very important application area, namely the exchange of Chinese character information, is made possible with our character description language *HanGlyph*.

4 The *HanGlyph* language

Based on the analysis described in previous sections, we defined a Chinese character description language, named *HanGlyph*. The most crucial characteristic of this language is that it is abstract and it captures only the topological relation of the strokes that form a character.

The essential information needed to distinguish a Chinese character is the arrangement of strokes. The precise location of each stroke can vary in a large extent up to a certain threshold, and the character can still be recognized correctly. For example, the following two characters, \pm and \pm , comprise exactly the same strokes and in exactly the same arrangement. The only difference between them is the relative length of the two horizontal strokes. Exactly how much longer a horizontal stroke is in these characters is unimportant for distinguishing between them. To recognize the character \pm , the threshold is that the upper horizontal stroke must be shorter than the lower one. Therefore, the *HanGlyph* language does not describe the precise geometric information of the characters.

4.1 The strokes

After studying a number of Chinese linguistic and graphological works, we selected a set of 41 strokes as the primitives of *HanGlyph*. Each primitive stroke is assigned a Latin letter as its code so that users can easily write *HanGlyph* expressions using a standard qwerty keyboard. Table 1 lists these primitive strokes.

4.2 The operators and relations

To form a Chinese character, one combines primitive strokes using operators. Five operators are defined as listed in Table 2. Figure 1 illustrates the composition performed by these operators. Each operator combines two operands to form a stroke cluster. This operation continues recursively until the desired character is formed. For example, to describe the character \pm , one may first combine two horizontal strokes using the top-bottom operator, then use the cross operator to add a vertical stroke. The *HanGlyph* expression for this character (written in ASCII characters) is `h h=s+`. (*Note*: the expression is in postfix notation.)

However, with only these operators, some characters, like \pm and \pm mentioned above, cannot be distinguished. To resolve situations like this, we can augment the operator with a number of *relation specifiers* to describe the operation in more specific terms. For our sample character \pm , the proper *HanGlyph* expression should be `h h=< s+_` where the symbol `<` denotes the relation that the length (i.e., the horizontal dimension) of the upper horizontal strokes must be shorter than the lower one, and the symbol `_` denotes the relation that the two operands of the cross operator, namely — and — , are aligned at the bottom.

Table 1: *HanGlyph* primitive strokes

Stroke	Name	Code	Examples	Stroke	Name	Code	Examples
	點	d	衣主沙		撇	p	大人少
	左點	D	心快熱		平撇	P	看千毛
	長點	f	不		豎撇	q	用月兒
	撇點	g	女好巡		撇折	r	么絲去
	橫	h	二三		捺	n	人大丈
	橫折	i	口四國		平捺	v	走趕
	橫折鉤	j	勾狗月		提	t	刁打地
	橫折撇	k	又水冬		點提	U	冰清
	橫折彎	l	沿船般		橫折折	N	凹
	橫折彎鉤	m	乙吃		橫折折折	L	凸
	橫鉤	a	冠皮軍		橫折折鉤	J	万仍
	豎	s	十中用		橫折提	E	计
	豎折	b	山區忙		橫撇彎鉤	K	隊
	豎彎	c	四酒		豎折折	B	鼎亞
	豎彎鉤	w	見兒		豎折折鉤	C	馬弓
	豎提	e	衣根		豎折撇	Q	专
	豎鉤	S	丁寸利		橫斜鉤	M	風颱
	彎鉤	X	狗狼家		橫折折撇	R	廷建及
	斜鉤	Y	我氣代		撇點	z	学
	臥鉤	W	心感		橫折斜	F	子魚矛
	橫豎彎鉤	o	九几				

Table 2: *HanGlyph* operators

Name	Symbol	Example
top-bottom 上下	=	昌早李
left-right 左右		明他你
fully enclosing 全包	@	回國困
half enclosing 半包 [†]	^	凶問區
cross 穿插	+	十半木

[†]A digit ranging from 0 to 7 should suffix the half enclose operator to indicate the direction of the opening.

The following four kinds of relations are defined:

1. Dimension — The relations in this group specify the relative dimension of the operands, i.e., comparing the width and height of their bounding boxes. There are four boolean relations: less than (<), greater than (>), not less than (!<), not greater than (!>).
2. Alignment — This specifies how the operands are aligned. The possible alignments are at top (‘), at bottom (⏟), at left (⌈), at right (⌋) and centered (#). More than one alignment can be

added to an operation, for example, to align at bottom right (⏟⌋).

3. Touching — This specifies whether the operands can touch each other. The possible relations are touching (~) or not touching (!~). When combining two elements to form a new character, the strokes next to the interface of the two elements may or may not touch each other. In general, if the strokes on either side of the interface have the same direction, they will not touch each other, for example, 昌 晶. Otherwise, the strokes may touch each other, for example, 香 相.
4. Scale (/) — This is used to adjust the width and height of the resulting character after the operation.

4.3 The *HanGlyph* macros

It would be very tedious if every character is described down to all its primitive strokes. It can be seen that certain arrangements of strokes are very common, such as 日 月, and so on. They are used to build up characters. We call them *components*. *HanGlyph* allows *macros* to be defined to stand for a component. For example, the component 日 is a macro with the name `ri_4`. It is defined in terms of

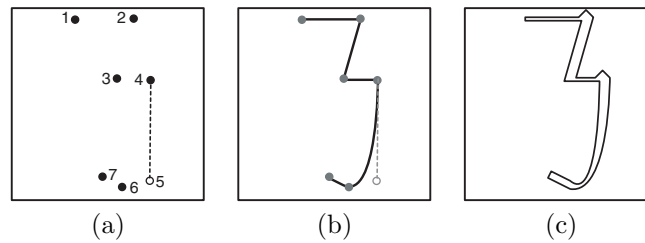


Figure 2: Basic stroke macros for the stroke 𠃉.

another macro `sih` representing the component 𠃉. The actual definitions are as below:

```
let(sih){s i|/h=/>
let(ri_4){sih h@}
```

With a small number of operators and relations, and using a postfix notation, the syntax of the *HanGlyph* language is very simple. This facilitates the development of simple language processors. Figure 5.4 shows the concrete syntax of *HanGlyph*.

After defining the *HanGlyph* language, we have written descriptions of more than 3755 Chinese characters (the first level characters in the GB2312-80 character set). We found that the *HanGlyph* language is adequate for its purpose, to capture the topological relation of the strokes.

5 The CCSS

The purpose of the Chinese Character Synthesis System (CCSS) is to render the *HanGlyph* expressions into a visual representation. For example, the *HanGlyph* expression `h h = <` only specifies that there are two horizontal strokes, one above another, and the upper stroke should be shorter than the lower one. It does not tell us about the exact distance between two strokes. In addition, it does not tell us exactly how much shorter is the upper stroke than the lower one.

The task of the CCSS is to determine and calculate the precise geometric information for each stroke so that a good rendering of characters can be generated.

The CCSS consists of three modules: basic strokes, composition operations and a *HanGlyph*-to-METAPOST translator. The first two modules are implemented as METAPOST macros. The translator is a C program.

5.1 Basic strokes

Each of the 41 basic strokes listed in Table 1 is implemented as a set of three METAPOST macros:

- A *Control-point* macro defines the control points, handle points and their properties. For

example, the stroke 𠃉 (横折折钩 *héngzhézhé*gōu) has six control points and a handle point as shown in Figure 2(a). The locations of the points are specified relative to a reference point. The properties of a control point indicate whether it is a beginning, an end, or a turning point. These properties will be used in creating the outline since its shape at different types of points will be different, for example, the second control point is a turning point, the outline at this point will have a serif shape. The properties will also be used in the composition operations to determine whether certain transformation and positioning operations are required.

- A *Skeleton* macro specifies a path passing through the control points. This path is very important. Given two points, the path can be straight or curvy; therefore, this macro traces out the exact stroke skeleton. Figure 2(b) shows the skeletal path of the stroke 𠃉.
- An *Outline* macro creates the outline for the stroke. It is defined relative to the control points and the skeletal path. Figure 2(c) shows the outline for the stroke 𠃉.

The first reason for organizing the stroke composition into three macros is to avoid distortion. In composition operations, each stroke will be transformed several times before the whole character is formed. If the stroke including the outline is represented in one macro, the transformation will distort the stroke thickness and even the direction in certain slant strokes.

Another reason is to provide meta-ness and flexibility. This organization provides several levels of style changes. The first level is to vary the parameters of the outline macros. For instance, changing the stroke thickness parameter will result in characters of varying stroke thickness. If we change the set of outline macros, we can create completely different font styles, but they may still be recognised as a family because the locations of control points are unchanged. More variations can be achieved if the

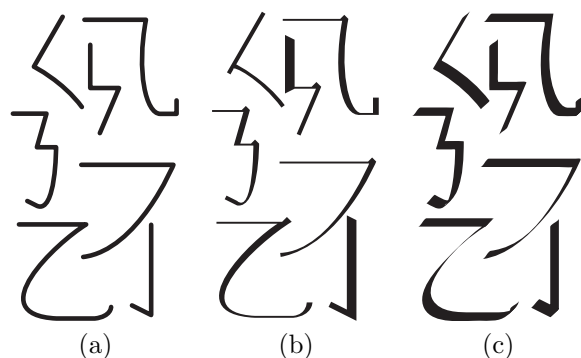


Figure 3: Variations of strokes having the same skeleton.

control point macros and the skeleton macros are also changed. The result will be a completely different font family. Figure 3 shows three variations of the same skeleton. Figure 3(a) is the simple skeletal path of the strokes. Figure 3(b) is the outline with serif. Figure 3(c) is generated by stroking the skeleton with a pen angled at 25 degrees.

5.2 Composition operations

CCSS implements five operations corresponding to the five operators defined in *HanGlyph*. These operations are implemented as METAPOST macros. Again, the operators in *HanGlyph* represent abstract operations, like the *Top-bottom* operator (=) only means ‘put an operand on top of another’. It carries no precise geometric information. Given this abstract instruction, the macro implementing this operation has to calculate the exact location and dimension of each operand. The resulting rendering should be a well-balanced and well-positioned arrangement of strokes.

One important task of the composition operation is to estimate the relative sizes and positions for its operands so that the result is visually well-balanced. For example, Figure 4 illustrates two characters having the same radical 木 (mù) on their left side. The width of this radical in the first character 林(lín) is larger than that in the second character 樹(shù) because the right-hand side of 樹 has many more strokes. We have found that the ratio of the widths of the two components is proportional to the ratio of the sums of the lengths of the strokes and the number of strokes of the components.

HanGlyph expressions may include a number of relations to augment the operators. The composition operations need to calculate the exact dimension and transformation to apply to each operand. For example, the character 人(rén) is composed of two strokes where the right-hand one is shorter and

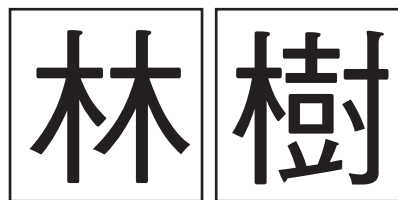


Figure 4: The same radical having different widths.

the two are aligned at the bottom. The composition operation will first scale the right-hand stroke down to a default size, and then translate it so that the bottom lines of the two strokes are aligned to the bottom of the character box as shown in Figure 5.

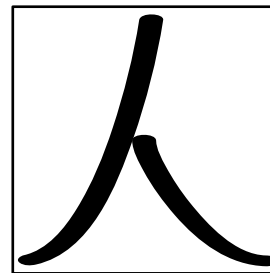


Figure 5: A character composed of two strokes aligned to bottom.

While we are talking about transforming the strokes, in fact, only the control points and the skeletal path are transformed. After all strokes forming a character have been put at the right position, the outline is drawn. This avoids the outline being distorted by the transformations.

5.3 *HanGlyph* to METAPOST translation

The front-end of the CCSS is the translator that converts *HanGlyph* expressions into METAPOST programs. The current implementation of the translator puts each *HanGlyph* expression into a METAPOST figure. Within each figure, the appropriate sequence of composition operation macros is called to render the character. The output of the system is a set of PostScript files.

This implementation provides a simple way to render the *HanGlyph* expressions and obtain previews of the characters. It facilitates the fine-tuning of the composition operations. Future implementations can streamline the process according to the requirements of the target application. For instance, a back-end processor can be added to convert the PostScript output into a particular format, such as a PostScript Type 3 font.

5.4 The syntax of *HanGlyph*

Figure 6 shows the concrete syntax of *HanGlyph* in an augmented BNF notation.

6 Conclusion

This paper describes an attempt to synthesize Chinese characters from an abstract description. A Chinese character description language known as *HanGlyph* has been defined. A Chinese character synthesis system is being developed. It is implemented in METAPOST and C, and the output is rendered in PostScript. The preliminary results show that the approach is very promising. Some of the characters generated by the CCSS are shown in Figure 7.

Currently, we are in the process of fine-tuning the composition parameters. We hope the system is able to produce visually pleasing characters. There are many factors that may affect the quality of the output, for example, the thickness of the strokes, the allocation of the space occupied by each component, and so on. Therefore, a considerable amount of experimentation is required to determine a set of parameters for composing characters.

There are many applications of such a system. The most important ones are in exchanging Chinese textual information in an open, heterogenous environment, and in Chinese font generation.

References

- [1] 蘇培成 (SU Pei Cheng). 二十世紀的現代漢字研究 (*The 20th century research on modern Chinese characters*). 書海出版社 (Su Hai Press), 2001.
- [2] 劉連元 (LIU Lian Yuan). 漢字拓扑結構分析 (Analysis of the topological structure of Chinese characters). In 漢字. 上海教育出版社 (Shanghai Education Press), 1993.
- [3] John Hobby and Gu Guoan. A Chinese metafont? *TUGboat*, 5(2):119–136, 1984.
- [4] Don Hosek. Design of Oriental characters with METAFONT. *TUGboat*, 10(4):499–501, 1989.
- [5] Fan Jiangping. Towards intelligent Chinese character design. In *Raster Imaging and Digital Typography II (RIDT91)*, pages 166–176. Cambridge University Press, 1992.
- [6] Donald E. Knuth. *The METAFONTbook*. Addison-Wesley, 1986.
- [7] Soon-Bum Lim and Myung-Soo Kim. Oriental character font design by a structured composition of stroke elements. *Computer-aided design*, 27(3):193–207, 1995.
- [8] 傅永和 (FU Yong He). 漢字結構和構造成分的基楚研究 (*Basic research on the structure of Chinese characters and their constituents*). 上海教育出版社 (Shanghai Education Press), 1993.
- [9] 傅永和 (FU Yong He). 中文信息處理 (*Chinese information processing*). 廣東教育出版社 (Guangdong Education Press), 1999.
- [10] Sakai Toshiyuki, Nagao Makoto, and Terai Hidekazu. A description of Chinese characters using subpatterns. *Information Processing Society of Japan Magazine*, 10:10–14, 1970.
- [11] Dong YunMei and Li Kaide. A parametric graphics approach to Chinese font design. In *Raster Imaging and Digital Typography II (RIDT91)*, pages 156–165. Cambridge University Press, 1992.

- $$\begin{aligned} \langle \text{hanglyph} \rangle &::= \langle \text{expr} \rangle + & (1) \\ \langle \text{expr} \rangle &::= \langle \text{glyph_expr} \rangle \mid \langle \text{macro} \rangle \mid \langle \text{char} \rangle & (2) \\ \langle \text{glyph_expr} \rangle &::= \langle \text{glyph} \rangle ; & (3) \\ \langle \text{macro} \rangle &::= \text{let}(\langle \text{id} \rangle)\{\langle \text{glyph} \rangle\} & (4) \\ \langle \text{char} \rangle &::= \text{char}(\langle \text{code} \rangle)\{\langle \text{glyph} \rangle\} & (5) \\ \langle \text{glyph} \rangle &::= \langle \text{glyph} \rangle \langle \text{glyph} \rangle \langle \text{opn} \rangle & (6) \\ &\mid \langle \text{stroke} \rangle \\ &\mid \langle \text{id} \rangle \\ \langle \text{opn} \rangle &::= \langle \text{parallel_operator} \rangle \langle \text{parallel_rels} \rangle & (7) \\ &\mid \textcircled{\langle \text{full_enc_rels} \rangle} \\ &\mid \sim \langle \text{dir_all} \rangle \langle \text{half_enc_rels} \rangle \\ &\mid + \langle \text{cross_rels} \rangle \\ \langle \text{parallel_operator} \rangle &::= = \mid | & (8) \\ \langle \text{dir} \rangle &::= .(\text{E} \mid \text{S} \mid \text{W} \mid \text{N} \mid \text{e} \mid \text{s} \mid \text{w} \mid \text{n}) & (9) \\ \langle \text{dir_all} \rangle &::= \langle \text{dir} \rangle \mid .(\text{NE} \mid \text{SE} \mid \text{NW} \mid \text{SW} \mid \text{ne} \mid \text{se} \mid \text{nw} \mid \text{sw}) & (10) \\ \langle \text{parallel_rels} \rangle &::= \langle \text{dimens} \rangle ? \langle \text{aligns} \rangle ? \langle \text{touch} \rangle ? \langle \text{scale} \rangle ? & (11) \\ \langle \text{full_enc_rels} \rangle &::= \langle \text{dimens} \rangle ? \langle \text{touch} \rangle ? \langle \text{scale} \rangle ? & (12) \\ \langle \text{half_enc_rels} \rangle &::= \langle \text{dimens} \rangle ? \langle \text{aligns} \rangle ? \langle \text{touch} \rangle ? \langle \text{scale} \rangle ? & (13) \\ \langle \text{cross_rels} \rangle &::= \langle \text{dimens} \rangle ? \langle \text{align} \rangle ? & (14) \\ &\quad (\langle \text{align} \rangle \mid \langle \text{intercept} \rangle) ? \langle \text{scale} \rangle ? \\ \langle \text{intercept} \rangle &::= * \langle \text{dir} \rangle (\langle + \text{int} \rangle (\langle \text{real} \rangle ? \langle \text{int} \rangle ?)) ? & (15) \\ \langle \text{dimens} \rangle &::= \langle \text{comp} \rangle (\langle \text{comp} \rangle \mid \langle \text{num} \rangle) ? & (16) \\ &\mid \langle \text{num} \rangle \langle \text{comp} \rangle ? \\ &\mid \langle \text{num} \rangle , \langle \text{num} \rangle \\ \langle \text{comp} \rangle &::= < \mid > \mid ! < \mid ! > \mid - & (17) \\ \langle \text{aligns} \rangle &::= \langle \text{align} \rangle \langle \text{align} \rangle ? & (18) \\ \langle \text{align} \rangle &::= ‘ \mid _ \mid [\mid] \mid \# & (19) \\ \langle \text{touch} \rangle &::= \sim \langle \text{dir_spec} \rangle * & (20) \\ &\mid ! \sim (\langle \text{dir_spec} \rangle \langle \text{num} \rangle ?) * \\ \langle \text{dir_spec} \rangle &::= (. \langle \text{dir} \rangle) + & (21) \\ \langle \text{scale} \rangle &::= / \langle \text{num} \rangle ? & (22) \\ \langle \text{num} \rangle &::= \langle \text{int} \rangle \mid \langle \text{real} \rangle & (23) \end{aligned}$$

Figure 6: The syntax of *HanGlyph* descriptions.



Figure 7: Some characters generated by CCSS.

Abstracts — Fonts

T_EX at the End: Omega and Zapfino

William Adams

ATLIS Graphics

`willadams@aol.com`

The future of type is OpenType (Adobe and Microsoft's successor to Apple's "Royal" font technology which was licensed to Microsoft as TrueType), Unicode, and other extensions of TrueType and the Type 1 font format such as ATSUI (Apple Typographic System for Unicode Information). While T_EX has been extended to support other recent formats and standards such as PDF, support for the new font formats has been limited at best, and in its most promising incarnation (QuickDraw/GX and T_EX/GX) has withered on the vine, as it were.

Fortunately, for Unicode in T_EX we have Omega, which, coupled with the other strengths of T_EX, can be sufficient to take advantage of new technologies without explicit support with the proper (or improper) techniques.

This paper will be an explanation and exploration of this, looking at a specific font and format (the `.dfont` ATSUI-enabled version of Zapfino), arguably very nearly a worst-case scenario, and how it can be disassembled into individual glyphs in EPS format and seamlessly stitched back together as an Omega Virtual Font with a matching Omega Translation Process to automatically insert ligatures and swash and variant forms using ASCII markup in an otherwise ordinary `.tex` source file which can then be used in a pre-press ready workflow.

(We expect to publish the full paper in the next regular issue of *TUGboat*. *Ed.*)

ε - Ω : A step towards the future with a look at the past

“A che serve vivere, se non c'è il coraggio di lottare?” (Giuseppe Fava)
“What purpose is living, if you don't have the courage to fight?”

Giuseppe Bilotta

Dipartimento di Matematica e Informatica

Università di Catania

viale A. Doria, 6

95125 Catania

Italy

`gip.bilotta@iol.it`

Abstract

In recent times, a topic of increasing relevance in discussions on the future of \TeX has been the number of different extensions to Knuth's original work, and the possibility of bringing them all together in a single program. In particular, on the one hand we have the features introduced in $\varepsilon\text{\TeX}$ which are almost essential to developers of modern formats (Con \TeX t, \LaTeX 3); on the other hand, the advanced typesetting features present in Ω are of vital importance, especially for \TeX users using non-Latin scripts.

This talk presents $\varepsilon\text{\Omega}$, a project whose aim is to provide a stable, fast variant of Ω supporting the $\varepsilon\text{\TeX}$ extensions. We will present the short history of the project (focusing in particular on the reasons behind some debatable choices), its current status and ideas for the project's future.

Goals and history of the project

In much the same way as $\varepsilon\text{\TeX}$ is (was) intended to fill the gap between \TeX 3 and $\mathcal{N}\mathcal{T}\mathcal{S}$, the goal of $\varepsilon\text{\Omega}$ is to fill the gap between the current Ω release(s) and the future ones that promise to have “every feature everyone wanted”. $\varepsilon\text{\Omega}$ thus intends to provide functional programs and tools that satisfy more modest requirements.

The need for a separate branch of Ω arose because the Ω development was not being responsive to important requests; $\varepsilon\text{\TeX}$ extensions were not provided despite long-time requests, and long-standing bugs and deficiencies were not being addressed; as a result, neither of the two available versions of Ω (1.15 and 1.23) was fully feasible for production use:

- 1.15 because of a major bug affecting day-to-day usage; this bug prevents Ω 1.15 from tripping correctly;
- 1.23 trips successfully, but is sadly too slow and bloated (both memory-wise and output-wise)¹ to be usable for heavy jobs;

¹ This depends on the introduction of a very important and useful node (*info_node*), but the advancements provided by it cannot be effectively turned off when not needed.

- Finally, both versions have buggy supporting utilities (the ones that deal with font-metric creations, Ω CPs and Ω TPs, etc.).

Therefore, the goal of the $\varepsilon\text{\Omega}$ project is to provide a program that:

- has $\varepsilon\text{\TeX}$ extensions;
- is stable enough (trips correctly);
- is fast enough;
- produces non-bloated DVI code;
- has solid supporting utilities.

My job has been first to choose which Ω to use as a base, and then to try to merge the $\varepsilon\text{\TeX}$ changefile. Luckily, I discovered that the differences between the two series had very little influence on the resulting changefiles, which meant I could focus on working on only one of them and still easily adapt the final outcome to the other version. Speed and leanness being two of the main considerations, I chose to concentrate on the 1.15 series.

Of course, support for $\varepsilon\text{\TeX}$ multidirectional typesetting was dropped. A more important and arguable change was the removal of the SGML/XML code from Ω : first, since part of it conflicted with some $\varepsilon\text{\TeX}$ code (the introduction of the `\middle`

primitive) and second, since (in the team’s opinion) the XML feature had a lower priority than merging ε - \TeX . This, making a timely release dictated the (temporary) removal of the Ω SGML/XML code from ε - Ω .

A first release of ε - Ω was thus officially made on December 21, 2002 (and yes, I must confess that the choice of the day was also dictated by aesthetic reasons . . .). This release still had the “tripping bug” that affected the 1.15 series of Ω , but did provide ε - \TeX enhancements for those who needed them (which for now means essentially Con \TeX t users).

Current status

In late April, having a little free time on my hands, I decided to give ε - Ω a second shot, trying to look for the code that caused the “tripping bug” that affected the Ω 1.15 series (and its variants): the bug, which revealed itself with a crash when running the `trip` test, affected production use of Ω in many contexts where over-/underfull boxes appeared, as well as causing the disappearance of ligatures, and other problems.

Since the bug did not affect any other version/variant of \TeX in my possession (including the 1.23 series of Ω), spotting the problem was rather easy by three-way diff’ing \TeX , Ω 1.15 and Ω 1.23. Once the culprit code was found, the solution was trivial. As a result, the latest official ε - Ω fully addresses three of the five target points, and is a good step forward towards a fourth one (stability).

There still are some known issues. (In particular, ε - Ω does still not pass `trip`: if `mem_bot` is set to 1, glue/skip assignments fail.) At this point, however, none of the known bugs in Ω itself prevent use of the program.

Future developments

The focus for the next release is to get closer to the final goals, by fixing the remaining bugs in the chief executable. This includes finding and fixing any Ω / ε - \TeX incompatibilities.

The following step will be working on the complementary utilities, making them functional again. This will then complete the five main goals of the project. Once these are accomplished, and if deemed necessary by the Ω status at the time of completion, a forward-port to the latest Ω branch will be attempted.

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 $\mathcal{N}\mathcal{T}\mathcal{S}$ team, for giving us ε - \TeX ,
- Idris Samawi Hamid, Alan Hoenig and Hans Hagen for pushing me into attempting the merge and supporting me through the whole long process,
- Christian Schenk and Fabrice Popineau for their constant feedback, help and support.

Editorial Note

The name of the project formerly known as ε - Ω has since been changed to “Aleph” (\aleph).

Makor: Typesetting Hebrew with Omega

Alan Hoenig

City University of New York
ahoenig@suffolk.lib.ny.us

Abstract

It's relatively easy to typeset a language using a keyboard customized for that language. A more interesting problem arises when trying to set a 'foreign' language (say, Russian, Hebrew, or Arabic) using a native keyboard (American, for example). This leads to the problem of *transliteration*: how to represent some language, call it *A*, using the conventions of a different language *B*?

This paper concerns the author's attempted solution to one such problem: to create quality Hebrew typesetting using the conventions of an English language keyboard. Apart from the different alphabet, which invokes a different set of sounds than does its English counterpart, Hebrew can involve as many as two distinct sets of diacritics, uses special glyph forms (sometimes) at word endings, and is, of course, typeset from right to left. The solution involves using the Omega extension of T_EX.

Makor, the name for this Hebrew typesetting system, consists of a user manual, fonts from seven distinct font families, and a special set of macros and conventions. Many examples of its use will be shown. All this software is publicly and freely available.

1 Introduction

We English speakers and readers are lucky — T_EX or its equivalent would have been quite different, and arguably more difficult to create, had Don Knuth needed to typeset different scripts with different conventions. I never fully realized this until I turned my attention to typesetting Hebrew. Today, we all realize how robust T_EX is, and how it can be coerced into doing stuff totally undreamt of by its author, but certain foreign languages break T_EX's back.

Hebrew, in fact, can not be handled by the original T_EX. Just in case you've never seen Hebrew, here's what we expect — at a bare minimum — from a Hebrew typesetting system; see figure 1 for pure Hebrew and figure 2 for mixed Hebrew-English text. (*Makor* produced these samples, and indeed all examples of Hebrew in this article.)

Discussing ways in which T_EX would fail will also deepen the a reader's understanding of Hebrew (or at least how to typeset it!).

2 Why T_EX can typeset Hebrew

Here are some things which are *not* a problem. First off, fonts are not a problem. It's easy enough to define a Hebrew font for use within a T_EX document.

As is well known, Hebrew is an RTL (right-to-left) language, whereas T_EX is an LTR (left-to-right)

typesetter. This is not really a problem at all. Early on, T_EX was extended to handle RTL. These early versions, T_EX--X_ET and so on, early provided this capability. In recent years, this RTL-capability has been subsumed in the various extended T_EX's that have appeared. Mixed Hebrew/English text should look something like figure 2.

Hebrew and Yiddish, like other Semitic languages, demand that certain letterforms be used only in word-initial and word-final positions. (Actually, only Yiddish has word-initial glyphs.) We'd like to design an input convention so that the typesetting engine makes the decisions as to which letterform is appropriate depending on context. This, too, T_EX can handle, by means of virtual fonts.

3 Why T_EX can't typeset Hebrew

So what *can't* T_EX do? One of the many fascinating things about Hebrew (and Arabic too) is that texts normally contain only the consonants of the words. Vowels are viewed as adjuncts, and are indicated solely by means of diacritical marks rather than by full-fledged letters. The trouble is, unlike English diacrits, each Hebrew letter has its own axis around which we need to (horizontally) center the vowel mark. So, for example, figure 3 shows two letters with the same vowel mark. You don't need to be a *bona fide* Hebrew reader to see that vowels

וַיִּמַּת שֵׁם מֹשֶׁה עֶבֶד־יְהוָה בְּאֶרֶץ מִדְבָּר עַל־פִּי יְהוָה: וַיִּקְבֹּר אֹתוֹ בְּגִי בְּאֶרֶץ מִדְבָּר מִזֶּמֶן בְּיַם סוּף וְלֹא־יָדַע אִישׁ אֶת־קְבֻרָתוֹ עַד הַיּוֹם הַזֶּה: וּמֹשֶׁה בֶן־מֵאָה וְעֶשְׂרִים שָׁנָה בָּמָתוֹ לֹא־כָהָתָה עֵינָיו וְלֹא־נָס לָחָה: וַיִּבְכּוּ בְּנֵי יִשְׂרָאֵל אֶת־מֹשֶׁה בְּעַרְבַת מִדְבָּר שְׁלֹשִׁים יוֹם וַיִּתְּמוּ יָמָיו בְּכִי אֲבָל מֹשֶׁה: וַיְהִי־שָׁעָה בֶן־נֹנֶן מְלֵא רוּחַ חֲכָמָה כִּי־סָמַךְ מֹשֶׁה אֶת־יָדָיו עָלָיו וַיִּשְׁמְעוּ אֵלָיו בְּנֵי־יִשְׂרָאֵל וַיַּעֲשׂוּ כְּאֲשֶׁר צִוָּה יְהוָה אֶת־מֹשֶׁה: וְלֹא־קָם נְבִיא עוֹד בְּיִשְׂרָאֵל כְּמֹשֶׁה אֲשֶׁר יָדָעוּ יְהוָה יְהוָה פְּנִימָה לְכָל־הָאָדָם וְהַמּוֹפְתִים אֲשֶׁר שָׁלַח יְהוָה לַעֲשׂוֹת בְּאֶרֶץ מִצְרָיִם לְפָרְעֹה וּלְכָל־עַבְדָּיו וּלְכָל־אֶרֶץ־מִצְרָיִם: וְלֹא־יָדָע הַחֲזֹקָה וְלֹא־יָדָע הַמּוֹרָא הַגָּדוֹל אֲשֶׁר עָשָׂה מֹשֶׁה לְעֵינֵי כָל־יִשְׂרָאֵל:

Figure 1: Hebrew with vocal diacritics.

Rabbinic Hebrew (RH) does not differ greatly from Biblical Hebrew (BH) in its inflection of the noun, although the neutralization of final *mem* and *nun* means that the masculine plural is often, as in Aramaic, מִן. Apart from the more frequent use of the archaic feminine suffix ת- as in כַּהֲנָתַי ‘priest’s wife’ and אִלְמָתַי ‘dumb woman’, RH also employs the suffixes ית- and ות- for example אַרְמִית ‘Aramaic’ and עֲבָדוּת ‘servitude’. RH developed distinctive feminine plural suffixes in -וֹתַי (Babylonian) or -וֹתַי (Palestinian), for example מְרַחֲצוֹת/מְרַחֲצוֹתַי ‘bath-houses’ and מְלָכוֹתַי, as in מְלָכוֹתַי ‘kingdoms’ for BH מְלָכוֹת, for nouns ending in ית- in the singular. Masculine plural forms sometimes differ from those that would be expected, or are normally found, in BH, for example, נִזְקִין from נִזֶּק ‘damage’, שְׁוֹרִים from שׁוֹר ‘ox’, שוּקִים from שׁוּק ‘market’, צָדָרִים from צָד ‘side’, חֲצָאִין from חֲצַי ‘half’, and שְׁלוֹחִין from שְׁלִיחַ ‘envoy’. The same is true of feminine nouns, for example אוֹתַי from אוֹת ‘letter (of alphabet)’, בְּרִיתוֹת from בְּרִית ‘covenant (without plural in BH)’, and אִמָּהוֹת from אִם ‘mother’.

Some masculine nouns take the feminine plural suffix ית-, for example, חַן from חָן ‘favour’, כְּלָלוֹת from כָּלֵל ‘rule’, תִּינוּקוֹת from תִּינוּק ‘baby’, חֵילוֹת from חֵיל ‘army’, עִירוֹת from עִיר ‘city’, and מִימוֹת from מַיִם ‘water’. Similarly, there are some feminine nouns which take the masculine plural suffix ים—יוֹנִים from יוֹנָה ‘dove’, נְמָלִים from נְמָלָה ‘ant’, and בִּיצִים from בִּיצָה ‘egg’, for example. Occasionally, both types of plural are evidenced, as with יָמִים/יָמוֹת from יוֹם ‘day’ or שָׁנִים/שָׁנוֹת from שָׁנָה ‘year’, with each form having a slightly different shade of meaning and the ‘feminine’ variant only used with suffixes. In RH we sometimes find plurals of nouns only attested in the singular in BH, for example אֲבָרִים from אֲבָר ‘limb’, דְּשָׁאִין from דְּשָׁא ‘grass’, and תְּמִידִים from תְּמִיד ‘daily sacrifice’. Likewise, there are singular forms of nouns only attested in the plural in BH, for example אֲלִמוֹג from אֲלִמוֹג ‘coral-wood’, בִּיצָה from בִּיצָה ‘egg’, and בְּצָל from בְּצָל ‘onion’. The dual is used more than in BH, with existing forms retained and new ones created, for example מְסַפְרִים ‘scissors’ and בְּנֵתִים ‘meanwhile’. (1993: A. S’aenz-Badillos, *A History of the Hebrew Language*, Cambridge University Press, pp. 188-89.)

Figure 2: Mixed Hebrew/English text.

are positioned in very different places. Actually, the situation is even worse than that, for each letter contains *two* such axes, one to be used for vowel marks appearing below the letter, and another for those above the letter. In theory, the typesetting engine

has to be able to keep track of axis placements for each individual letter. (In practice, though, many letters share the same axis placement.) As far as I can see, there is no really robust way to encode this axis information within a Hebrew font.

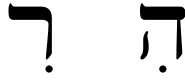


Figure 3: Same vowel, different axes.

Speaking of vowel marks, there is a second set of diacritics we should be concerned with. To be sure, they are only necessary in Biblical texts, but \TeX typesetters tend to be neurotically completist about things like these, and so for the sake of completeness, we'd like our system to contain this capability. (This alternative set of diacritics provides information on how to chant the words in the sacred texts.) \TeX is hard enough pressed to typeset normal vowels without worrying about this second set. See figure 4 for Hebrew containing both sets of diacrits. Even if you don't know how to read Hebrew, a quick comparison with figure 1 shows which accents belong to the second set.

Another interesting aspect of Hebrew typography is that of alternative conventions. One or two letters, such as the *lamed* (with an 'l' sound), might appear in two distinct forms. Also, there are alternative choices for diacritic placement in certain instances. In addition, the presence or absence of the vocal diacrits and the cantorial diacrits themselves count as alternatives. Since, in the compulsive manner common to a certain class of \TeX users, we would like to enable an aspiring author to use any selection of these alternatives with any other, it's not clear how a \TeX solution for this could arise. Different fonts? We'd need $2^4 = 16$ for each base font. Macros? That would involve too much author markup. Active characters? Way too dangerous. In my view, no good pure \TeX solution exists.

The hitherto unspoken assumption on my part up to now is that we're typing at an American keyboard. As a result, we'll need a really swell input scheme to lessen the possibility of making typing errors. \TeX certainly makes it possible to get, say, the Hebrew equivalent of 'l' by typing `l`. One problem involves letters with sounds that don't occur in English. To be sure, virtual font virtuosity allows us to type `ch` to get the Hebrew letter corresponding to a throat-clearing guttural, which is what 'ch' corresponds to (in German, at any rate). However there are additional keyboard entry issues that would require stretching virtual font definitions to the limit, so much so as to put them out of the reach of essentially any \TeX user. (The *Makor* manual describes these keyboard entry conventions.)

Another problem with proper typesetting Hebrew (and Arabic too, for that matter) has to do

with numbers. Oddly enough, numbers appear in standard LTR order in a Hebrew document. Suppose `h_1` and `h_2` represent strings of number-free input which typeset the proper Hebrew text h_1 and h_2 . Suppose `\[` and `\]` are the markup switches that enter and exit Hebrew typesetting modes. Then, we expect to be able to enter

```
\[h_1 12345 h_2\]
```

in order to typeset the fragment

$$h_212345h_1.$$

\TeX , though, will typeset $h_254321h_1$. You might think we could get the proper text if we exit and enter Hebrew mode before and after typesetting the number. But think about it — if you do typeset

```
\[h_1\] 12345 \[h_2\]
```

what you get is the opposite-of-correct $h_112345h_2$. Of course, you could design, using recursion, a (hypothetical) `\HebrewNumber` macro to do the job, but somehow you should expect to be able to key in numeric data in an input file without requiring special markup.

For these reasons, I have deemed it unrealistic to aspire to perfect Hebrew typesetting using \TeX or $\varepsilon\text{-}\TeX$.

4 Omega

Omega is a superset of \TeX originally created (and still being developed) by Yannis Haralambous and John Plaiçe. It was developed to handle typesetting idiosyncrasies in all the world's languages. I cannot testify about its success in other languages, but it does a splendid job with Hebrew.

Although Omega contains \TeX (and therefore all of \TeX 's capabilities) at its heart, Omega differs from \TeX in several well-defined ways. I should mention that as a matter of course, it includes RTL typesetting.

More significantly, its registers have been extended to 32-bits, so, for example, it can handle large Unicode fonts. Next, it includes the capability of analyzing patterns in the input and modifying the input stream, before sending this stream to Omega for typesetting. For example, an Omega text filter could check input for the string `ffi` and replace it by the appropriate ligature. (This is a silly example, because \TeX 's ligature mechanism already does that quite nicely.) However, the modifications to the input stream could also be inclusion of a macro call, so for the first time in the history of \TeX , it's now possible to modify the course of typesetting by means of macros which the typesetting engine itself inserts for you within the text of your input.

LIBER ESTHER

3 א אחר ׀ הדברים האלה גדל המלך אחשוורוש את המן בן המרתא האנני ונשאהו נישם את כסאו מעל כל־השרים אשר אתו: ב וכל־עבדי המלך אשר־בשער המלך כרעים ומשתחוים להמן כי־כן צוה־לו המלך ומרדכי לא יכרע ולא ישתחוה: ג ויאמרו עבדי המלך אשר־בשער המלך למרדכי מדוע אתה עובר את מצות המלך: ד ויחי [באמרם] כאמרם אליו יום ויום ולא שמע אליהם ויגידו להמן לראות תעמדו דברי מרדכי כי־הגיד להם אשר־הוא יהודי: ה וירא המן כי־אין מרדכי כרע ומשתחוה לו וימלא המן חמה: ו ויבז בעיניו לשלח יד במרדכי לבדו כי־הגידו לו את־עם מרדכי ויבקש המן להשמיד את־כל־היהודים אשר בכל־מלכות אחשוורוש עם מרדכי: ז בחדש הראשון הוא־חדש ניסן בשנת שתים עשרה למלך אחשוורוש הפיל פור הוא הגדל לפני המן מיום ׀ ליום ומחדש לחדש שנים־עשר הוא־חדש אדר: ח ויאמר המן למלך אחשוורוש ישנו עם־אחר מפגז ומפגד בין העמים בכל מדינות מלכותך ודתייהם שנות מכל־עם ואת־דתי המלך אינם עשים ולמלך אין־שזה להניחם: ט אם־על־המלך טוב יכתב לאבדם ועשרת אלפים כפר־כסף אשכול על־ידי עשי המלאכה להביא אל־נגני המלך: י ויסר המלך את־מבעתו מעל ידו ויתנה להמן בן־המרתא האנני צרר היהודים: יא ויאמר המלך להמן הנסף נתון לך והעם לעשות בו כטוב בעיניך: יב ויקראו ספרי המלך בחדש הראשון בשלושה עשר יום בו ויכתב ככל־אשר־צוה המן אל אחשדרפני המלך ואל־הפחות אשר ׀ על־מדינה ומדינה ואל־שרי עם ועם מדינה ומדינה ככתבה ועם ועם כל־שנו בשם המלך אחשוורוש נכתב ונחתם בטבעת המלך: יג ונשלוח ספרים ביד הרצים אל־כל־מדינות המלך להשמיד להרג ולאבד את־כל־היהודים מנער ועד־זקן טף ונשים ביום אחד בשלושה עשר לחדש שנים־עשר הוא־חדש אדר ושללם לבז: יד פתשגן הכתב להנתן דת בכל־מדינה ומדינה גלוי לכל־העמים להיות עתדים ליום הנה: טו הרצים יצאו דחופים בדבר המלך והדת נתנה בשששן הבירה והמלך והמן ישבו לשחות והעיר שושן נבוכה:

Figure 4: Typesetting with two sets of diacritic marks; cf. figure 1.

In my opinion, Omega represents a truly significant extension to TeX. But I don't really want to stand before you as Omega booster. Yannis and John are forceful and articulate advocates of their own work, and I encourage the interested author to explore the large base of Omega literature and to join the Omega list.

5 Makor

Makor is my name for the system I created for typesetting Hebrew.

sh[^]aulom, 'Olaum!
 שְׁלוֹם, עוֹלָם!
Hello, world!

Figure 5: Makor input and output.

I'm not going to talk about specific methods for using *Makor*, nor about any of the underlying tricks I used in the *Makor* macros, thereby doing my part to uphold a longstanding tradition at these meetings of banning audience-unfriendly discussion. This package comes with a user manual, `mkr2man.pdf`, and I invite interested and masochistic authors to dip into

the macro file `makor2.tex` and the Omega `.otp` files that are part of the package. The package also includes `refcard.tex`, which is a reference card for all *Makor* conventions; figure 6 displays part of that for anyone who's interested.

If you need to typeset Hebrew and quality of output and ease of input is your concern, here's why you should use *Makor*:

- It's *easy* to enter consonants *and* vowels into a document. For example, the Hebrew equivalent of 'Hello, world!' might be casually transliterated as `shalom`, `'olam!`; see figure 5 for the *Makor* equivalent input.
- *Makor* automatically decides if a final form for a letter is necessary. If you know Hebrew, you'll appreciate that you get these forms automatically in figure 5.
- *Makor* takes care to position the vowel symbols properly with respect to different letters, as we've discussed.
- It's easy to finagle these and other aspects—altering placement of a vowel, forcing or suppressing the final form of a letter, and so on.
- *Makor* adopts the view that the diacrits we've mentioned are part of the logical structure of the document. It's a good idea to include them

Consonants	Star Convention	'Doubles'	Basic Vowels	More Vowels
ב v	א '*	א ' ^ '	א ' ae	י y:a
ג g	ב v*	ב vv	ב vei	כ kh:e
ד d	ג g*	ג gg	ג g'	ל l:+
ה h	ד d*	ד dd	ד da	מ m
ו w	ה h*	ה hh	ה he	נ n "
ז z	ו w*	ו ww	ו wi	ס .s i
ח ch	ז z*	ז zz	ז zu	צ ' a
ט .t	ח ch*	ח chch	ח cho	פ f e
י y	ט .t*	ט .t.t	ט .tau	ק ts +
כ kh	י y*	י yy	ט .t+	ק q '
ל l	כ kh*	כ khkh	ו oo	
מ m	ל l*	ל ll	ו w*	
נ n	מ m*	מ mm	ו 0	
ס .s	נ n*	נ nn		
ע '	ס .s*	ס .s.s		
פ f	ע ' *	ע ''		
צ ts	פ f*	פ ff		
ק q	צ ts*	צ tsts		
ר r	ק q*	ק qq		
ש sh	ר r*	ר rr		
ש sh^	ש sh*	ש shsh		
ש ^s, ^sh	ש sh^*	ש shsh^		
ת s	ש ^s*	ש ^shsh		
ת th	ת th*	ת thth		
ת th	ת s*	ת ss		

Conventions	Special Dagesh	Defective Cholam
char^ force final letter	ב b	בא bo'
char_ suppress final	כ k	באם bo\/'aum
^^ forces א	פ p	
_^ forces ,	ת t	

Cantorial Accents	Miscellanea	Adjusting Vowel Placement
ה h[etnahta]	- -	_ .4_rau moves the au vowel 40% of the width of the glyph starting at the left and moving to the right.
ה h[segol]	- =	_3.5pt,-3.5pt_rau displaces the au vowel 3.5 pt to the left of where it should be and 3.3 pt up from where it should be.
ה h[shalshélet]	--- ---	^ .9^r [shalshélet] puts the upper cantorial accent 90% of the way to the right of the left edge.
ה h[zaqefqatan]	" "	^3pt,1.5pt^r [shalshélet] displaces the upper accent by 3 pt to the left and 1.5 pt up.
ה h[zaqefgadol]	' '	_1pt,1pt;3pt,3pt_zau [etnahta] to displace the pair of vowels and the increase the offset between them.
ה h[tipeha]	, ,	
ה h[revia]	' ts'	
ה h[zarqa]	ת123א '123th	
ה h[pashta]	? !?, ?!	
ה h[yetiv]	□ **	
ה h[tevir]		
ה h[geresh]		
ה h[gereshmuqdam]		
ה h[gershayim]		
ה h[qarneypara]		

Commands
\[start Heb. typesetting
\] finish Hebrew
\/ pattern breaker
\noindent suppresses indentation
\HINDENT enforces a Hebrew indentation
\HPAR terminates a Hebrew paragraph
\CENTERLASTLINE centers last line of paragraph
\< unskip command
\> synonym for unskip
\hfontdef{HH}{ezra2}{9pt} defines a Hebrew font for use
\hfont{HH} uses the named font
\PrintChar{242} identifies a character by slot, and prints it
\MakorEnvironment{MKR} selects typesetting environment

End of Word
ך kh
ך k
ם m
ם mm, m*
ן n
ן nn, n*
ף f
ף p
ץ ts
ץ tsts, ts*
'ץ ts'

Figure 6: Part of the Makor reference card.

אבְּנֵי־הַיְהוּדִים כִּי לָמַם נִן סַעֲפָר זִץ קַרְשָׁא	ezra2	Ezra Medium (at 12 pt)
אבְּנֵי־הַיְהוּדִים כִּי לָמַם נִן סַעֲפָר זִץ קַרְשָׁת	ezrai2	Ezra Italic (Oblique; 12 pt)
אבְּנֵי־הַיְהוּדִים כִּי לָמַם נִן סַעֲפָר זִץ קַרְשָׁא	ezrag12	Ezra Grey1 (12 pt)
אבְּנֵי־הַיְהוּדִים כִּי לָמַם נִן סַעֲפָר זִץ קַרְשָׁא	ezrag22	Ezra Grey2 (12 pt)
אבְּנֵי־הַיְהוּדִים כִּי לָמַם נִן סַעֲפָר זִץ קַרְשָׁא	ezrag32	Ezra Grey3 (12 pt)
אבְּנֵי־הַיְהוּדִים כִּי לָמַם נִן סַעֲפָר זִץ קַרְשָׁא	ezrag42	Ezra Grey4 (12 pt)
אבְּנֵי־הַיְהוּדִים כִּי לָמַם נִן סַעֲפָר זִץ קַרְשָׁא	ezrag52	Ezra Grey5 (12 pt)
אבְּנֵי־הַיְהוּדִים כִּי לָמַם נִן סַעֲפָר זִץ קַרְשָׁא	ezrag62	Ezra Grey6 (12 pt)
אבְּנֵי־הַיְהוּדִים כִּי לָמַם נִן סַעֲפָר זִץ קַרְשָׁא	ezrag72	Ezra Grey7 (12 pt)
אבְּנֵי־הַיְהוּדִים כִּי לָמַם נִן סַעֲפָר זִץ קַרְשָׁא	ezrag82	Ezra Grey8 (12 pt)
אבְּנֵי־הַיְהוּדִים כִּי לָמַם נִן סַעֲפָר זִץ קַרְשָׁא	ezrag92	Ezra Grey9 (12 pt)
אבְּנֵי־הַיְהוּדִים כִּי לָמַם נִן סַעֲפָר זִץ קַרְשָׁת	osh2	OmegaSerifHebrew (12 pt)
אבְּנֵי־הַיְהוּדִים כִּי לָמַם נִן סַעֲפָר זִץ קַרְשָׁת	oshi2	OmegaSerifHebrew Italic (12 pt)
אבְּנֵי־הַיְהוּדִים כִּי לָמַם נִן סַעֲפָר זִץ קַרְשָׁת	hasida2	Hasida Monowidth (12 pt)
אבְּנֵי־הַיְהוּדִים כִּי לָמַם נִן סַעֲפָר זִץ קַרְשָׁת	hasidai2	Hasida Monowidth Italic (12 pt)
אבְּנֵי־הַיְהוּדִים כִּי לָמַם נִן סַעֲפָר זִץ קַרְשָׁת	frankruehl2	Frank Ruehl (12 pt)
אבְּנֵי־הַיְהוּדִים כִּי לָמַם נִן סַעֲפָר זִץ קַרְשָׁת	frankruehli2	Frank Ruehl Italic (12 pt)
אבְּנֵי־הַיְהוּדִים כִּי לָמַם נִן סַעֲפָר זִץ קַרְשָׁת	frankrueh1b2	Frank Ruehl Bold (12 pt)
אבְּנֵי־הַיְהוּדִים כִּי לָמַם נִן סַעֲפָר זִץ קַרְשָׁת	frankrueh1bi2	Frank Ruehl Bold Italic (12 pt)
אבְּנֵי־הַיְהוּדִים כִּי לָמַם נִן סַעֲפָר זִץ קַרְשָׁת	nachlight2	Nachlieli Light (12 pt)
אבְּנֵי־הַיְהוּדִים כִּי לָמַם נִן סַעֲפָר זִץ קַרְשָׁת	nachlighti2	Nachlieli Light Italic (12 pt)
אבְּנֵי־הַיְהוּדִים כִּי לָמַם נִן סַעֲפָר זִץ קַרְשָׁת	nachbold2	Nachlieli Bold (12 pt)
אבְּנֵי־הַיְהוּדִים כִּי לָמַם נִן סַעֲפָר זִץ קַרְשָׁת	nachboldi2	Nachlieli Bold Italic
אבְּנֵי־הַיְהוּדִים כִּי לָמַם נִן סַעֲפָר זִץ קַרְשָׁת	rashi2	Rashi Script (at 9 pt)
אבְּנֵי־הַיְהוּדִים כִּי לָמַם נִן סַעֲפָר זִץ קַרְשָׁת	rashii2	Rashi Italic (9 pt)
אבְּנֵי־הַיְהוּדִים כִּי לָמַם נִן סַעֲפָר זִץ קַרְשָׁת	rashib2	Rashi Bold (9 pt)
אבְּנֵי־הַיְהוּדִים כִּי לָמַם נִן סַעֲפָר זִץ קַרְשָׁת	rashibi2	Rashi Bold Italic (9 pt)
XW9 9P270T4YLY Z0E3Y <14K	oldheb	Old Hebrew (at 14 pt)

Figure 7: Specimens of all Makor fonts.

in the input, even if you don't want them in the output, because it makes it easier to proofread the source document. Consequently, *Makor* has a software switch for including or suppressing these vowel markers.

- The *Makor* package comes with over twenty different fonts, as you see in figure 7.
- Authors can enter cantorial diacritics (trope) into the text, as we've discussed. See figure 4.
- *Makor* also supports Yiddish with a separate input convention and special Yiddish characters.

Makor is also Ladino-ready, but as I've been unable to find a reliable explanation of Ladino typographic conventions, I have not (yet) implemented a Ladino input scheme. (Ladino bears a similar relationship to Hebrew and Spanish as does Yiddish to Hebrew and German. There are, in addition, other dialects that use the Hebrew alphabet, and *Makor* could support these conventions as well.)

- Alternative typographic conventions are supported, as we've already discussed.

ארבעה אבות מייקין אבות קרי להיך דכמינן בקרא צהדיא. ונגמרא מפרש הי זיהו מולדות: השור והבור כו' כסדר שפן כחוצין צפרשה סדרן צמשה. צפרשה ראשונה נאמרה בשור שניה בור: מצנעה מפשה צנמי: הצערי כו' מלא אש: לא הרי השור כהרי המצנעה כלומר אי כתב רחמנא לא נפק מצנעה מיניה ואמטו להכי אצטרכו למיכתב ולהכי נקט צרישא כהרי המצנעה ולא נקט להו כסדר לא הרי השור כהרי הבור משום דמו לא הוי מצי למימתי לא זה חיה שיש צנה רוח חיים ועוד טעמא אחרתא דהא רבואא אשמעי שאע"ג שיש לשניהם רוח חיים לא נפיק חד מהצניה וצנמי מפרש מאי לא הרי דקאמר: כהרי האש שאין צו רוח חיים ואי לא תצניה רחמנא הוא חמינע ליפטר: לע זה חיה כו' אלו שלשמן דרכן ליך ולהויק: האש השוא כו' מפרש צנמי לאחויי מאי: צמיטב הערין מעדמ נכסיו יצבה דמי הוקו אס רוצה לפדוע לא קרקע: גמ' טעמא צשווג: סקילה -

ארבעה אבות מייקין, השור והבור והמבצה וההבצור. לא הרי השור כהרי המבצה, ולא הרי המבצה כהרי השור, ולא זה וזה שיש בהן רוח חיים, כהרי האש שאין בו רוח חיים, ולא זה וזה שדרבן לילך ולהזיק. הצד השונה שבהן שדרבן להזיק ושמידתן עליך, וכשהזיק חב המזיק לשלם תשלומיו נזק במיטב הארץ: גמ' מדקתני אבות מכלל דאבא תולדות.

ארבעה

השור והבור - פירוש צנוטערס כסדר שנכתבו צפרשה סדרן צמשה ואף על גב דלמ"ד מן שור לרגלו לא הוי כסדר הפרשה דרגל נפק לן מושלח את צעירה דכמיני צמר בזה מ"מ שם שור כמיני קודם צפרשה דהיינו נגיבה דקבן ולמ"ד מצנעה זה אש שאע"ג דלצמר הצערה כמיני צפרשת אמור מנה בהמה אשלמנה דהיינו אש דאזיק שור לא מס לטותו כסדר הפרשה לפי שרמון כל כך ושנאו כסדר לא הרי דיפא ש מצנעה קודם להצנע:

לא הרי השור כהרי המצנעה - פירוש אין קולמו של שור כקולמו של מצנעה דמפרש לקמן צנמי למ"ד מנא שור לקרנו ומצנעה לשיני משום דשור כוונתו להזיק ומצנעה אין כוונתו להזיק ולפיך אי כתב רחמנא שור לא חמי מצנעה מינה שהוא קל מינה ואין פירושו כשאר מקומות שצמלמוד לא ראי זה דהתם פירושו אין חומרא של סה כחומרא של זה ולכן אין ההומרות גורמות זה הדין אלא הצד השונה שבהן גורם הדין ושינה כאן החלמוד פירושו מנשאר מקומות משום דאזכיר החומר תחילה בלא זה וזה שיש צנה רוח חיים:

ולא זה חיה צנמי רוח חיים כהרי האש - גבי שור ומצנעה לא הולך לפרש החומרא כי הכא משום דמד לקל למצנע חומר באחד מה שאין תצירו זה דלא מני הך לא הרי האש כרי השור ומצנעה דקתני לעיל לא הרי המצנעה כרי השור משום שלע היה יכול למצנע חומרא מה שאין צמיטה דאי משום דכח אחר מעורב צו ואין הולך לדעתו כמו שצרו אין זה חומרא מלא שכיב לה גבי חומר בהם מצנעה ואלא דלמנין לקמן (ק"ג: ד"ו): גבי אבנו וסמיני מאי שגא אש דקח אחר מעורב צו ואין הולך לדעתו ה"ק מאי שגא אש שאע"ג שכה אחר מעורב צו ראי להחמיצ צו משום שהוע מונך ושמינרו כו' וצפיא גבי צור הוה מצי למימר לא הרי הבור שחמילת עשייתו לנזק

אכל וזמן ועת לבלתיחפץ תחת השמים: עת ללדת ועת למות עת לשעת ועת לעקור נטוע: עת להרוג ועת לרפוא עת לפרוץ ועת לבנות: עת לבכות ועת לשחוק עת ספור ועת רקוד: עת להשליך אבנים ועת לבקש אבנים עת לחבוק ועת להפזר עת לאבד עת לשמור ועת לשנא עת מלחמה ועת עמל: ראיתי אתהענין אתההכל עשה יפה בעתו לאימצא האדם ועד"סוף: ידעתי כי אין בתינו: ונזם כל האדם שיאכל ושתה וראה טוב ככל עמלו יהיה לעולם עליו אין להותיף וממנו אין לגרע והאלקים כפר הוה והאלקים יבקש אתהנדרף: ועוד ראיתי תחת אמרתי אני בלבי אתהצדיק ואת הרשע ישפט האלקים כיעת לבלתיחפץ ועל כליהמעשה שם: אמרתי אני בלבי עלידברת בני האדם לבדם האלקים ולראות שהם יבנהמה המה להם: כי מקרה בני האדם ויקרה בהמה ויקרה אחר להם כמות זה בן מות זה ורעה אחר לכל ומותר האדם מןהבהמה אין כי הכל הכל: הכל הולך אלמקום אחד הכל יהיה מןהענין והכל שבו אלמעשה: מי יודע רוח בני האדם העלה היא למעלה ורוח בהמה הירדת היא למטה לארץ: וראיתי כי אין טוב מאשר ישמח האדם במעשיו כיהוה חלקו כי מי יביאנו לראות במה שיהיה אתרו:

Figure 8: Traditional Hebrew typography from the Talmud.

מקור

אכל וזמן ועת לבלתיחפץ תחת השמים: עת ללדת ועת למות עת לשעת ועת לעקור נטוע: עת להרוג ועת לרפוא עת לפרוץ ועת לבנות: עת לבכות ועת לשחוק עת ספור ועת רקוד: עת להשליך אבנים ועת לבקש אבנים עת לחבוק ועת להפזר עת לאבד עת לשמור ועת לשנא עת מלחמה ועת עמל: ראיתי אתהענין אתההכל עשה יפה בעתו לאימצא האדם ועד"סוף: ידעתי כי אין בתינו: ונזם כל האדם שיאכל ושתה וראה טוב ככל עמלו יהיה לעולם עליו אין להותיף וממנו אין לגרע והאלקים כפר הוה והאלקים יבקש אתהנדרף: ועוד ראיתי תחת אמרתי אני בלבי אתהצדיק ואת הרשע ישפט האלקים כיעת לבלתיחפץ ועל כליהמעשה שם: אמרתי אני בלבי עלידברת בני האדם לבדם האלקים ולראות שהם יבנהמה המה להם: כי מקרה בני האדם ויקרה בהמה ויקרה אחר להם כמות זה בן מות זה ורעה אחר לכל ומותר האדם מןהבהמה אין כי הכל הכל: הכל הולך אלמקום אחד הכל יהיה מןהענין והכל שבו אלמעשה: מי יודע רוח בני האדם העלה היא למעלה ורוח בהמה הירדת היא למטה לארץ: וראיתי כי אין טוב מאשר ישמח האדם במעשיו כיהוה חלקו כי מי יביאנו לראות במה שיהיה אתרו:

Figure 9: Another example of complex Hebrew typography.

- Fonts include oddball special characters so that it's possible to typeset the Hebrew Bible with *Makor*.
- Numbers are entered normally; *Makor* takes care to typeset them properly within the Hebrew (that is, the numbers are LTR even though the surrounding text is RTL).
- *Makor* understands the conventions of Arab \TeX , so you can process Arab \TeX Hebrew documents in *Makor*. This is actually a consequence of Omega's filtering mechanism. It's just a question of prepending to the filter a sub-filter that translates Arab \TeX 's input into *Makor* input.
- *Makor* understands the conventions of BHS, so you can process Biblia Hebraica Stuttgartensia in *Makor*. This downloadable ASCII file contains the full text of the Hebrew Bible, vowels, vocal diacrits, and special symbols included, but using a vastly different input convention from that of *Makor*.
- One of *Makor*'s fonts allows scholars to typeset Old Hebrew (see the last line of figure 7).
- Authors and scholars can typeset using the archaic Palestinian or Babylonian vowel systems. These were systems of vocal diacritics that died out of use about one thousand years ago or so.
- When you revise the document, either by changing your text or altering layout parameters (say, a column width), these changes automatically propagate into your text.
- Because \TeX is Omega's underlying typesetting engine, layouts of arbitrary complexity are possible. Figures 8 and 9 show some of the complex Hebrew typography that, over the centuries, has become traditional.
- All of the versatility that's part of \TeX and of Omega is always available to the author using this system.
- ... And much, much more!

6 Getting Makor

The latest version of *Makor* is always available from the CTAN archives. You should find it at

`tex-archive/language/hebrew/makor`

but this directory may be a bit fouled up. To reliably locate the Makor software, simply visit your local CTAN site, and search for the file `mkr2man.pdf`. Then, pick up all the files in that directory and all sub-directories.

Please feel free to contact me with any questions or comments about *Makor*. You can reach me via email at `ahoenig@suffolk.lib.ny.us`

A multidimensional approach to typesetting

John Plaice, Paul Swoboda

School of Computer Science and Engineering
The University of New South Wales
UNSW Sydney NSW 2052, Australia
plaice@cse.unsw.edu.au, pswoboda@cse.unsw.edu.au
<http://www.cse.unsw.edu.au/~plaice>

Yannis Haralambous

Département Informatique
École Nationale Supérieure des Télécommunications de Bretagne
BP832, F-29285 Brest Cédex, France
Yannis.Haralambous@enst-bretagne.fr
<http://omega.enstb.org/yannis>

Chris Rowley

Faculty of Mathematics and Computing
The Open University, UK
Milton Keynes MK7 6AA, United Kingdom
C.A.Rowley@open.ac.uk

Abstract

We propose to create a new model for multilingual computerized typesetting, in which each of language, script, font and character is treated as a multidimensional entity, and all combine to form a multidimensional context. Typesetting is undertaken in a typographical space, and becomes a multiple-stage process of preparing the input stream for typesetting, segmenting the stream into clusters or words, typesetting these clusters, and then recombining them.

Each of the stages, including their respective algorithms, is dependent on the multidimensional context. This approach will support quality typesetting for a number of modern and ancient scripts. The paper and talk will show how these are to be implemented in Ω .

Introduction

We propose to create a radically new and practical model for character-level typesetting of all the world's languages and scripts, old and new. This goal is currently unattainable by any existing system, because of the underlying assumption that entities such as script, language, font, character and glyph are discrete, eternal and unchanging, as is supposed, for example, in the standards for Unicode [20], XML [21] and XSL [22].

The key innovations in this proposal are (a) the assumption that these entities, their relationships and the processes (programs) applied to them are all arbitrarily parametrizable by a tree-structured context, and (b) the explicit manipulation of the complex and dynamic relationships between a (logical)

character stream input and its visual representation on a particular medium as positioned glyphs.

These innovations lead directly to the concept of a *typographical space* that constrains the variance in the context and effectively embodies a certain set of processes and customs — as once might have been practiced in a typesetting workshop — while still allowing parametrization by the context.

Quality multilingual typesetting, as opposed to quality typesetting of unilingual documents for a number of different languages, requires the juxtaposition of separate typographical spaces. The separate spaces encourage the development of specialized algorithms to properly support widely different languages, scripts, and output substrates. The fact that the same tree-structured context permeates all of these different spaces ensures that key parameters

can be shared across—or at least have correspondences in—different spaces, thereby ensuring consistency from one typographical space to another.

This approach greatly simplifies a number of tricky problems when one refers to language and script. Consider, for example, the English language: it is a multidimensional complex, varying through time (Old, Middle and Modern English), space (national and regional Englishes), and culture (science, arts, business, diplomacy, etc.). Understanding this variance is important: just as a simple example, US English and UK English have different spellings and hyphenations, and require different rules.

Scripts and their use have evolved similarly. In the past century, German and Turkish have both adopted the Latin script (from the Gothic and Arabic scripts, respectively). Chinese is printed, depending on the country, using ‘traditional’ or ‘simplified’ characters; transliteration for Chinese into the Latin script uses either the Wade-Giles or the *pinyin* method. This evolution and diversity means that documents encoded using one script should, possibly with the aid of linguistic tools, be printable using other scripts. For example, Ω , using a German morphological analyzer, has been used to automatically print historical German texts in the Gothic script [4].

The relationship between character and glyph has also evolved, in inconsistent ways. A *character* is a unit of information exchange, while a *glyph* is a unit of visual information. If we consider, for example, the glyph *æ*, used in *mediæval*, it is considered to be a ligature—a variant glyph—in English, while in Danish it is considered to be a character in its own right. In fact, one of the authors (Haralambous) [3] has shown that glyphs and characters are not absolutes, but, rather, are fluid concepts dependent on the context.

These relationships become more complex when we are faced with paleo-scripts from the mediæval and ancient worlds. For example, there are something like 200 recognized Indic scripts, all derived from the Brahmi script. They all have similar—but clearly not identical—structure and there are situations in which it is natural to consider them as separate scripts while in other situations it is easier to consider them as variants of a single script.

We propose to use a tree-structured context to describe, to the desired level of precision, the entities that are being manipulated. This context will be used to describe (a) how exactly to interpret the input; (b) the exact format of the output; and (c) the required processing. The latter should define how many passes over the input are required,

what linguistic, layout or other plug-in tools should be used, along with the parametrization for each of these tools. An example context would be:

```
<character: <Unicode +
                encoding: <UTF8>> +
input: <XML + DTD: <TEI>> +
language: <English +
                spelling: <Australian> +
                script: <Latin>> +
output: <PDF +
                viewer: <AcrobatReader +
                version <5.0 +
                OS: <MacOSX>>>>>>
```

where *input* and *output* are called *dimensions*, and *language:script* a *compound dimension*. The context will be inferred from environment variables, system locale, user profiles, command-line arguments, menu selections, and document markup.

This approach was first outlined in a position paper written by three of the authors [14], but at the time we had not understood the importance of the *typographical space*. It is the typographical space that allows us to fix exactly the meanings of character, glyph, language, script and font. In so doing, we facilitate the construction of modular and flexible typesetters that allow automatic linguistic tools to add arbitrary markup to a text before it is printed, much as a traditional typographer might have used dictionaries and grammar books before pouring lead.

To transform the above basic ideas into real, functional software usable for typesetting real, multilingual documents is not a trivial task. In this paper, we outline the steps that have led to the current ideas, and elaborate on problems still to be resolved.

We begin with a quick summary of the \TeX character-level typesetter. Then we explain how the introduction of Ω TPs and Ω TP-lists in Ω provides a sophisticated means for adapting \TeX 's typesetter to multilingual typesetting. What Ω offers to the specialist user is great flexibility in manipulating the many different parameters needed for high-quality typesetting of different scripts.

However, this programming flexibility, with its large numbers of parameters, greatly complicates the user interface. The answer lies in being able to explicitly manipulate an *active run-time context* that permeates the entire typesetting process. We describe below how versioned macros, Ω TPs, and Ω TP-lists have been added to Ω to offer a high-level interface that a non-specialist user can manipulate with success.

Once such an active context is added to the interface, it becomes natural to incorporate the context into the entire process, and to completely redesign \TeX 's character-level typesetter. We examine below the initial proposal for such a typesetter, using the typographical space. We conclude by proposing a number of natural typographical spaces, along with their relevant parameters.

Computer typesetting, \TeX and Ω

For this paper, we define computer typesetting to be “*The production of printed matter by computer, ultimately to be viewed on some output medium*”. The origins of computer typesetting go back to the 1950s, but it was not until 1982, with \TeX [7], that it became possible to use computer software for high-quality typesetting of English and mathematics, as in *The Art of Computer Programming* [6].

At the character level, \TeX can work in *text-mode* or in *math-mode*. In *text-mode*, *characters* in the input file are transformed almost directly into *glyphs* (‘pictures’ of characters) in the current font, and these glyphs are positioned side-by-side ‘on the *baseline*’. A font-specific finite-state automaton can be used to change the glyphs used (by using *ligatures*) and their horizontal placement (by *kerning*). The ‘words’ thus typeset are then separated by a font-specific amount of stretchable inter-word space (*glue*) to form the stream of typeset glyphs that is passed to \TeX 's paragrapher. In *math-mode*, \TeX uses a hand-crafted algorithm to lay out glyphs in 1.5 dimensions (this notation comes from frieze patterns).

The resulting stream of typeset glyphs is fed to \TeX 's *paragraphing algorithm* [8], which breaks the typeset stream for a paragraph at optimal — according to some acceptability criterion — places to produce lines of text placed in horizontal boxes. A much simpler algorithm is used for cutting pages from a continuous galley of such boxes. All computations in \TeX are based on the width, height and depth of boxes, and these are derived ultimately from the same metrics for glyphs in the fonts.

The Ω system [11], developed by Plaice and Haralambous, is a series of extensions to the \TeX system that facilitate multilingual typesetting. In Ω , the input character stream is processed by a series of filters, ultimately generating another character stream. Once all of the filters are applied, the resulting stream is passed to the \TeX text-mode typesetter. We have written filters for character set conversion, transliteration, morphological analysis, spell-checking, contextual analysis, and 1.5-dimensional layout. The Ω system has been used to typeset al-

phabetic scripts from Europe and the Caucasus, curvilinear scripts from the Middle East, South Asia and South-East Asia, and East-Asian ideograms.

With the Ω TP mechanism, one can call many different filters for many different tasks. It often happens that some of these filters are only to be used in a selective manner, which very quickly creates a combinatorial explosion of new Ω TP-lists, hardly a favorable situation. This is resolved by introducing the run-time context of intensional programming, explained in the following sections.

Intensional programming

Intensional programming [15] is a form of computing that supposes that there is a multidimensional context, and that all programs are capable of adapting themselves to this context. The context is pervasive, and can simultaneously affect the behavior of a program at the lowest, highest and middle layers.

When an intensional program is running, there is a *current context*. This context is initialized upon launching the program from the values of environment variables, from explicit parameters, and possibly from active context servers. The current context can be modified during execution, either explicitly through the program's actions, or implicitly, through changes at an active context server.

A context is a specific point in a multidimensional space, i.e., given a dimension, the context will return a value for that dimension. The simplest contexts are dictionaries (lists of attribute-value pairs). A natural generalization is what will be used in this paper: the values themselves can be contexts, resulting in a tree-structured context. The set of contexts is furnished with a partial order \sqsubseteq called a *refinement relation*.

During execution, the current context can be queried, dimension by dimension, and the program can adapt its behavior accordingly. In addition, if the programming language supports it, then contextual conditional expressions and blocks can be defined, in which the *most relevant* case, with respect to the current context and according to the partial order, is chosen among the different possibilities.

In addition, any entity can be defined in multiple *versions*, (context, object) pairs. Whenever an identifier designating an entity appears in an expression or a statement, then the most relevant version of that entity, with respect to the current context, is chosen. This is called the *variant substructure principle*. The general approach is called *intensional versioning* [17].

The ISE programming language [16, 19] was the first language combining both intensional program-

ming and versioning. It is based on the procedural scripting language Perl, and it has greatly facilitated the creation of multidimensional Web pages. Similar experimental work has been undertaken, under the supervision of the author Plaice, with C, C++, Java, and Eiffel. And, when combined with a context server (see Swoboda's PhD thesis [18]), it becomes possible for several documents or programs to be immersed in the same context.

Structuring the context

We use the same notation to designate contexts and versions of entities. This section has three subsections. First, we define *contexts* and the refinement relation. Then, we define *version domains*, which hold versioned entities. Finally, we define *context operators*, which are used to change from context to context. In the following section, we will show how all of these are to be used.

Contexts and refinement Let $\{(\mathbb{S}_i, \sqsubseteq_i)\}_i$ be a collection of disjoint sets of ground values, each with its own partial order. Let $\mathbb{S} = \cup_i \mathbb{S}_i$. Then the set of contexts \mathbb{C} ($\ni C$) over \mathbb{S} is given by the following syntax:

$$C ::= \perp \mid A \mid \Omega \mid \langle B; L \rangle \quad (1)$$

$$B ::= \epsilon \mid \alpha \mid \omega \mid v \quad (2)$$

$$L ::= \emptyset \mid d:C + L \quad (3)$$

where $d, v \in \mathbb{S}$.

There are three special contexts:

- \perp is the *empty context* (also called *vanilla*);
- A is the *minimally defined context*, only just more defined than the empty one;
- Ω is the *maximally defined context*, more defined than all other contexts.

The normal case is that there is a *base value* B , along with a *context list* (L for short), which is a set of *dimension-context* pairs. We write δL for the set of dimensions of L .

A sequence of dimensions is called a *compound dimension*. It can be used as a path into a context. Formally:

$$D = \cdot \mid d:D \quad (4)$$

If C is a context, $C(D)$ is the subtree of C whose root is reached by following the path D from the root of C :

$$C(\cdot) = C \quad (5)$$

$$\langle B; d:C' + L \rangle (d:D) = C'(D) \quad (6)$$

As with contexts, there are three special base values:

- ϵ is the *empty base value*;

- α is the *minimally defined base value*, just more defined than the empty base value;
- ω is the *maximally defined base value*, more defined than all others.

The normal case is that a base value is simply a scalar.

To the set \mathbb{C} , we add an *equivalence* relation \equiv , and a *refinement* relation \sqsubseteq . We begin with the equivalence relation:

$$\perp \equiv \langle \epsilon; \emptyset \rangle \quad (7)$$

$$A \equiv \langle \alpha; \emptyset \rangle \quad (8)$$

$$\Omega \equiv \left\langle \omega; \sum_{d \in \mathbb{S}} d:\Omega \right\rangle \quad (9)$$

$$\frac{L_0 \equiv_L L_1}{\langle B; L_0 \rangle \sqsubseteq \langle B; L_1 \rangle} \quad (10)$$

Thus, \perp and A are notational conveniences, while Ω cannot be reduced. The normal case supposes an equivalence relation \equiv_L over context lists:

$$\emptyset \equiv_L d:\perp \quad (11)$$

$$d:\langle B; L + L' \rangle \equiv_L d:(\langle B; L \rangle + \langle B; L' \rangle) \quad (12)$$

$$L \equiv_L \emptyset + L \quad (13)$$

$$L \equiv_L L + L \quad (14)$$

$$L + L' \equiv_L L' + L \quad (15)$$

$$L + (L' + L'') \equiv_L (L + L') + L'' \quad (16)$$

The $+$ operator is idempotent, commutative, and associative. Now we can define the partial order over entire contexts:

$$\perp \sqsubseteq C \quad (17)$$

$$C \sqsubseteq \Omega \quad (18)$$

$$\frac{C \neq \perp}{A \sqsubseteq C} \quad (19)$$

$$\frac{C_0 \equiv C_1}{C_0 \sqsubseteq C_1} \quad (20)$$

$$\frac{B_0 \sqsubseteq_B B_1 \quad L_0 \sqsubseteq_L L_1}{\langle B_0; L_0 \rangle \sqsubseteq \langle B_1; L_1 \rangle} \quad (21)$$

which supposes a partial order \sqsubseteq_B over base values:

$$\epsilon \sqsubseteq_B B \quad (22)$$

$$B \sqsubseteq_B B \quad (23)$$

$$B \sqsubseteq_B \omega \quad (24)$$

$$\frac{B \neq \epsilon}{\alpha \sqsubseteq_B B} \quad (25)$$

$$\frac{v_0, v_1 \in \mathbb{S}_i \quad v_0 \sqsubseteq_i v_1}{v_0 \sqsubseteq_B v_1} \quad (26)$$

The last rule states that if v_0 and v_1 belong to the same set \mathbb{S}_i and are comparable according to the

partial order \sqsubseteq_i , then that order is subsumed for refinement purposes.

The partial order over contexts also supposes a partial order \sqsubseteq_L over context lists:

$$\emptyset \sqsubseteq_L L \quad (27)$$

$$\frac{L_0 \equiv_L L_1}{L_0 \sqsubseteq_L L_1} \quad (28)$$

$$\frac{C_0 \sqsubseteq C_1}{d:C_0 \sqsubseteq_L d:C_1} \quad (29)$$

$$\frac{L_0 \sqsubseteq_L L_1 \quad L'_0 \sqsubseteq_L L'_1}{L_0 + L'_0 \sqsubseteq_L L_1 + L'_1} \quad (30)$$

Rule 30 ensures that the $+$ operator defines the least upper bound of two context lists.

Context and version domains When doing intensional programming, we work with *sets* of contexts, called *context domains*, written \mathcal{C} . There is one operation on a context domain, namely the *best-fit*. Given a context domain \mathcal{C} of existing contexts and a requested context C_{req} , the best-fit context is defined by:

$$\text{best}(\mathcal{C}, C_{\text{req}}) = \max\{C \in \mathcal{C} \mid C \sqsubseteq C_{\text{req}}\} \quad (31)$$

If the maximum does not exist, there is no best-fit context.

Typically, we will be versioning *something*, an object of some type. This is done using *versions*, simply (C, object) pairs. *Version domains* \mathcal{V} then become functions mapping contexts to objects. The *best-fit object* in a version domain is given by:

$$\text{best}_O(\mathcal{V}, C_{\text{req}}) = \mathcal{V}(\text{best}(\text{dom } \mathcal{V}, C_{\text{req}})) \quad (32)$$

Context operators Context operators allow one to selectively *modify* contexts. Their syntax is similar to that of contexts.

$$C_{\text{op}} ::= C \mid [P_{\text{op}}; B_{\text{op}}; L_{\text{op}}] \quad (33)$$

$$P_{\text{op}} ::= -- \mid E \quad (34)$$

$$B_{\text{op}} ::= - \mid \epsilon \mid B \quad (35)$$

$$L_{\text{op}} ::= \emptyset_{L_{\text{op}}} \mid d:C_{\text{op}} + L_{\text{op}} \quad (36)$$

A context operator is applied to a context to transform it into another context. (It can also be used to transform a context operator into another; see below.) The $-$ operator removes the current base value, while the $--$ operator in P_{op} is used to clear all dimensions not explicitly listed at that level.

Now we give the semantics for C C_{op} , the application of context operator C_{op} to context C :

$$C_0 C_1 = C_1 \quad (37)$$

$$\Omega C_{\text{op}} = \text{error} \quad (38)$$

$$\langle B; L \rangle [--; B_{\text{op}}; L_{\text{op}}] = \quad (39)$$

$$\langle B; L \setminus (\delta L - \delta L_{\text{op}}) \rangle [E; B_{\text{op}}; L_{\text{op}}]$$

$$\langle B; L \rangle [E; B_{\text{op}}; L_{\text{op}}] = \quad (40)$$

$$\langle (B B_{\text{op}}); (L L_{\text{op}}) \rangle$$

The general case consists of replacing the base value and replacing the context list. First, the base value:

$$B - = \epsilon \quad (41)$$

$$B \epsilon = B \quad (42)$$

$$B_0 B_1 = B_1 \quad (43)$$

Now, the context list:

$$L \emptyset_{L_{\text{op}}} = L \quad (44)$$

$$(d:C + L) (d:C_{\text{op}} + L_{\text{op}}) = \quad (45)$$

$$d:(C C_{\text{op}}) + (L L_{\text{op}})$$

$$L (d:C_{\text{op}} + L_{\text{op}}) = \quad (46)$$

$$d:(\perp C_{\text{op}}) + (L L_{\text{op}}), \quad d \notin \delta L$$

Context operators can also be applied to context operators. There are two cases:

$$[P_{\text{op}}; B_{\text{op}0}; L_{\text{op}0}] [E; B_{\text{op}1}; L_{\text{op}1}] = \quad (47)$$

$$[P_{\text{op}}; (B_{\text{op}0} B_{\text{op}1}); (L_{\text{op}0} L_{\text{op}1})]$$

$$[P_{\text{op}}; B_{\text{op}0}; L_{\text{op}0}] [--; B_{\text{op}1}; L_{\text{op}1}] = \quad (48)$$

$$[--; (B_{\text{op}0} B_{\text{op}1}); ((L_{\text{op}0} \setminus (\delta L_{\text{op}0} - \delta L_{\text{op}1})) L_{\text{op}1})]$$

Now that we have given the formal syntax and semantics of contexts, version domains, and context operations, we can move on to typesetting.

The running context in Ω

As is usual, the abstract syntax is simpler than the concrete syntax, which offers richer possibilities to facilitate input. Here is the concrete syntax for contexts in Ω :

C	::=	$\langle \rangle$	Empty context
		$\sim \sim$	Minimum context
		$\sim \sim$	Maximum context
		$\langle \text{val} \rangle$	Base value
		$\langle L \rangle$	Subversions
		$\langle \text{val} + L \rangle$	Base & subversions
val	::=	\sim	Minimum value
		\sim	Maximum value
		string	Normal value
L	::=	$\text{dim}:C [+ \text{dim}:C]^*$	
dim	::=	string	

Here is the concrete syntax for context operations:

C_{op}	::=	C	Replace the context
		$[\]$	No change
		$[val_{\text{op}}]$	Change base
		$[L_{\text{op}}]$	Change subversions
		$[val_{\text{op}}+L_{\text{op}}]$	Change base & subs
val_{op}	::=	$-$	Clear base
		val	New value
		$--$	Clear subversions
		$val+--$	New base, clear subs
		$---$	Clear base & subs
L_{op}	::=	$dim:C_{\text{op}} [+ dim:C_{\text{op}}]^*$	

In Ω , the current context is given by:

```
\contextshow{}
```

If D is a compound dimension, then the subversion at dimension D is given by:

```
\contextshow{D}
```

while the base value at dimension D is given by:

```
\contextbase{D}
```

This context is initialized at the beginning of an Ω run with the values of environment variables and command-line parameters. Once it is set, it can be changed as follows:

```
\contextset{Cop}
```

Adapting to the context

During execution, there are three mechanisms for Ω to modify its behavior with respect to the current context: (1) *versioned execution flow*, (2) *versioned macros*, and (3) *versioned Ω TPs*.

Execution flow The new `\contextchoice` primitive is used to change the execution flow:

```
\contextchoice{\{Cop1\}=>\{exp1\},
               \dots
               \{Copn\}=>\{expn\}
               }
```

Depending on the current context C , one of the expressions exp_i will be selected and expanded. The one chosen will correspond to the *best-fit* context among $\{C, C_{\text{op1}}, \dots, C_{\text{opn}}\}$ (see the discussion above of Context and Version Domains).

Macros The Ω macro expansion process has been extended so that any control sequence can have multiple, *simultaneous* versions, at the same scoping level. Whenever `\controlsequence` is expanded, the *most relevant*, i.e. the *best-fit*, definition, with respect to the current context, is expanded.

A version of a control sequence is defined as follows:

```
\vdef{Cop}\controlsequence args{definition}
```

If the current context is C , then this definition defines the C C_{op} version of `\controlsequence`. The scoping of definitions is the same as for \TeX .

This approach is upwardly compatible with the \TeX macro expansion process. The standard \TeX definition:

```
\def\controlsequence args{definition}
```

is simply equivalent to

```
\vdef{<>}\controlsequence args{definition}
```

i.e., it defines the empty version of a control sequence.

As stated above, during expansion the best-fit definition, with respect to the current context, of `\controlsequence` will be expanded whenever it is encountered. It is also possible to expand a particular version of a control sequence, by using:

```
\vexp{Cop}\controlsequence
```

Ω TPs and Ω TP-lists Beyond the ability to manipulate larger data structures than does \TeX , Ω allows the user to apply a series of filters to the input, each reading from standard input and writing to standard output. Each of the filters is called an Ω TP (Ω Translation Process), and a series of filters is called an Ω TP-list.

There are two kinds of Ω TP: internal and external. Internal Ω TPs are finite state machines written in an Ω -specific language, and they are compiled before being interpreted by the Ω engine. External Ω TPs are stand-alone programs, reading from standard input and writing to standard output, like Unix filters.

Internal and external Ω TPs handle context differently. For external Ω TPs, the context information can be passed on through an additional parameter to the system call invoking the external Ω TP:

```
program -context=context
```

Internal Ω TPs have been modified so that every instruction can be preceded by a context tag. Using the simplest syntax, this becomes:

```
<<context>> pattern => expression
```

When an internal Ω TP is being interpreted, an instruction is only examined if its context tag (defaulting to the empty context) is less than the current running context.

When Ω TPs and Ω TP-lists are being declared in Ω , the `\contextchoice` operator can be used to build versioned Ω TP-lists. With versioned Ω TP-lists, it becomes possible to define a single Ω TP-list with n Ω TPs, and each of the n Ω TPs can be activated with a separate parameter.

The versioned interface finally provides a user-level means for manipulating the large sets of parameters that must be handled when doing complex multilingual typesetting. When a transliterator is needed, the appropriate parameter is set. When a more complex layout mechanism is chosen, then another parameter is set. When spell-checking is desired, then another parameter is set. And so on. And the macros and Ω TP-lists adapt accordingly.

Because of the flexibility of the new interface, it is simpler to suppose that Ω always has an active Ω TP-list, and that it changes its behavior as the text changes its parameters. According to this vision, then, multilingual typesetting simply means changing parameters as needed.

The versioned approach also resolves an issue that has been vexing the authors ever since the Ω and \LaTeX projects have been trying to design a high-level interface for Ω usable by \LaTeX . The problem is that a language is not a monolithic, isolated, eternal and unchanging entity. Versioning of the macros and Ω TPs allows one to deal with the variance in language and script, as well as encouraging the sharing of resources across multiple languages.

Context-dependent typesetting

The existing Ω framework is very powerful, in the sense that the Ω TPs can make the \TeX character-level typesetter stand on its head to produce amazing results, without the end-user having to know what is going on. However, it is hardly a natural process to take a character-level typesetter designed for English with its isolated glyphs and occasional ligatures and then to use it to undertake complex Arabic typesetting with its numerous ligatures and floating diacritics.

Far more appropriate is to break up the typesetting process into separate modules, and to parameterize each of these with the current context.

In the most general sense, a typesetter is a program that transforms a stream of characters into a stream of positioned glyphs. We can separate out three themes:

- *Atomic typesetting* is the transformation of a (small) fully marked-up stream of characters into a stream of positioned glyphs. An atomic typesetter might be used directly by an application that prints one or two words at different points on a computer screen, e.g. by mapping software to print out a city or river name, or by a more complex continuous typesetter.
- *Continuous typesetting* is the transformation of a (larger) stream of characters into a stream of positioned glyphs that can be segmented at dif-

ferent points to produce several lines (or other structures) of typeset text.

- *Preparing the input* is the process of applying several programs to a stream of characters to add additional markup so that the typesetter can fully do its work.

A continuous typesetter would typically use one or more atomic typesetters, and might also require input to be prepared.

Below, we give a simple model of a continuous typesetter. It is split into four separate *phases*: *preparation*, *segmentation*, *micro-typesetting* and *recombination*. Each of these phases is dependent on the context, and we write the process, using C++ syntax, as:

```
stream<Glyph>
typeset(stream<Char> input,
        Context context) {
    stream<Char> prepared =
        input.apply(otp_list.best(context));
    stream<Cluster> segmented =
        segmenter.best(context)(prepared);
    stream<TypesetCluster> typeset =
        clusterset.best(context)(segmented);
    stream<Glyph> recombined =
        recombine.best(context)(typeset);
    return recombined;
}
```

where *function.best(context)* means that the most relevant version of *function*, with respect to *context*, is selected. We examine each of the phases in detail.

Preparation

```
stream<Char> prepared =
    input.apply(otp_list.best(context));
```

The preparation phase in this new approach is similar to the current situation in the Ω system. At all times, there is an active Ω TP-list. This list consists of individual Ω TP's, each of which is a filter reading from standard input to standard output. What is new is that the whole process becomes context-dependent. First, the most relevant Ω TP-list, with respect to the context and using the refinement relation over contexts, is the one that is active. Second, once chosen, it can test the current context and adapt its behavior, by selectively turning on or off, or even replacing, individual Ω TP's.

The preparation phase works entirely on *characters*, i.e. at the *information exchange* level, but it allows additional typographic information to be added to the character stream, so that the following phases can use the extra information to produce better typography.

Segmentation

```
stream<Cluster> segmented =
  segmenter.best(context)(prepared);
```

The segmentation phase splits the stream of characters into clusters of characters; typically, segmentation is used for word detection. In English, word detection is a trivial problem, and segmentation just means recognizing ‘white space’ such as the blank character, Unicode U+020. By contrast, in Thai, where there is normally no word-delimiter in the character stream (blanks are traditionally used only as sentence-delimiters), it is impossible to do any form of automatic processing unless a sophisticated morphological analyzer is being used to calculate word and syllable boundaries. In many Germanic and Slavic languages, it is also necessary to find the division of compound words into their building blocks. These processes are closely related to finding word-division points, so this should be incorporated into this part of the process (a very different approach to that of \TeX). The choice of segmenter is thus clearly seen to be context-dependent.

Cluster typesetting

```
stream<TypesetCluster> typeset =
  cluster.set.best(context)(segmented);
```

During the typesetting phase, a *cluster engine* processes a character cluster, taking into account the current context including language and font information, and produces the typeset output—a sequence of positioned glyphs. In many cases, such as when hyphenation or some other form of cluster-breaking is allowed, there are multiple possible typeset results, and all of these possibilities must be output. When dealing with complex scripts or fonts allowing great versatility (as with Adobe Type 3 fonts), many different cluster engines are needed: these are selected and their behaviour is fine-tuned according to the context.

Recombination

```
stream<Glyph> recombined =
  recombine.best(context)(typeset);
```

The final phase, before calling a higher-level formatting process such as a paragrapher, is the recombination phase. Here, the typeset clusters are placed next to each other. For simple text, such as the English in this proposal, this simply means placing a fixed stretchable space between typeset words. In situations such as Thai and some styles of Arabic typesetting, kerning would take place between words. Once again, the recombiner’s behavior is context-dependent.

Typographical spaces

Given the sophistication of the multiple-phase process, and that the choice of segmenter, cluster engine and recombiner are all context-dependent, and that the actions of each of these, once they are chosen, also depends on the context, this new model of typesetting engine is potentially *much* more powerful than anything previously proposed or implemented. However, there remains a key problem in the type of the function:

```
stream<Glyph>
typeset(stream<Char> input,
        Context context);
```

In this type declaration, the types `Glyph` and `Char` appear to be normal datatypes, i.e., fixed, unchanging sets, which is not at all consistent with our view that character and glyph should be perceived as multidimensional entities.

Really, the sets for character and glyph should be context-dependent. However, if these basic types were to continually change, then it would be very difficult to write any of the algorithms, because one could never be sure of the ultimate particles, the atoms, with which one was working.

To resolve this problem, we introduce the *typographical space*. This space is designed to constrain the variance in the context. Within a specific typographical space, the types for character and glyph remain fixed. Hence the above type becomes something like:

```
stream< Glyph<TS> >
typeset(stream< Char<TS> > input,
        Context context);
```

In a typographical space, certain parameters are kept fixed, or at least their values are kept within a certain range. Other parameters may vary at will, and their values may be manipulated as appropriate by the algorithms within that space.

Suppose there was a typographical space for Greek typesetting, including modern and ancient Greek, literary Greek and colloquial Greek, as well as other languages that have been typeset using the Greek alphabet. Then the character datatype would most likely correspond to a subset of Unicode, augmented by additional characters that were not included in the standard. The glyph datatype would consist of many glyphs, and could contain a number of precomposed multi-accented glyphs or a smaller set of isolated glyphs, including accents, that are to be placed at appropriate places.

The typographical space is a necessary solution to the problem raised by the existence of multi-script

character sets such as Unicode. It is simply infeasible to write a single typesetter that will do quality typesetting of Egyptian hieroglyphics, Japanese kanji with furigana, Persian in Nastaliq style, and German using Fraktur fonts.

By creating separate typographical spaces for these different kinds of situation, we can allow specialists to build typesetters for the scripts and languages that they know best. What is still needed for quality multilingual typesetting is to define some basic parameters, or dimensions, that apply across different typographical spaces, so that it becomes possible to move smoothly from one typographical space to another.

Example spaces

We intend to test and validate the model described above by creating typographical spaces for at least the following scripts:

- *Latin, Greek, Cyrillic, IPA*: left-to-right, discrete glyphs, numerous diacritics, stacked vertically, above or below the base letters, liberal use of hyphenation;
- *Hebrew*: right-to-left, discrete glyphs, optional use of diacritics (vowels and breathing marks), which are stacked horizontally below the base letter;
- *Arabic*: right-to-left, contiguous glyphs, contextually shaped, many ligatures, optional use of diacritics (vowels and breathing marks), placed in 1.5-dimensions, above and below;
- *Indic scripts*: left-to-right, 1.5-dimensional layout of clusters, numerous ligatures, applied selectively according to linguistic and stylistic criteria;
- *Chinese, Japanese*: vertical or left-to-right, often on fixed grid, with annotations to the right or above the main sequence of text, automatic word recognition — in Chinese and Japanese, “words” use one or more characters, but these are not visually apparent — needed for any form of analysis;
- *Egyptian hieroglyphics*: mixed left-to-right and right-to-left, 1.5-dimensional layout.

Once these basic spaces are validated, then further experiments, viewing language as a multidimensional entity, can be undertaken. Already with Ω , we have typeset Spanish with both the Hebrew and Latin scripts; Berber with the Tifnagh, Arabic and Latin scripts; Arabic with Arabic, Hebrew, Syriac, Latin and even *Arabized Latin* (Latin script with a few additional glyphs reminiscent of the Arabic script). The Arabic script can be rendered in

Naskh or Nastaliq or many other styles. Japanese can be typeset with or without *furigana*, little annotations above the *kanji* (the Chinese characters) to facilitate pronunciation. Some of the corresponding typographical spaces will be quite interesting.

The objective is to incorporate solutions to all such problems, currently solved in an *ad hoc* manner, into our framework; each time, the key is to correctly summarize the typographical space. With this key, then the choice of segmenters, clusters engines and recombiners to build, and of how they are built, is clarified; nevertheless, these algorithms may remain complex, because of the inherent complexity of the problems they are solving.

Conclusions

When we have fully developed this model, we will be able to produce, with relative ease, high-quality documents in many different languages and scripts.

Furthermore, this new approach of using contexts can be used to improve not just micro- but also macro-typesetting. Rowley, as one of the leaders of the L^AT_EX3 Project, has worked with closely related ideas in the context of Mittelbach’s *templates* for higher-level formatting processes [2]. Here the particular instance of a template object that is used to format a document element will depend on a context that is derived from both the logical position of that element in the structured document and from the formatting of the physically surrounding objects in the formatted document. Collaboration between the current authors and other members of the L^AT_EX3 team will lead to many new interfaces that give access to the new functionality.

Other examples of the importance of such a structured context in document processing can be found in work by Rowley with Frank Mittelbach [10].

Another example of dependence on this visual context occurs in the use of Adobe Type 3 fonts, which are designed so that glyphs can be generated differently upon each rendering (see [1] for a discussion of a number of effects). On another level, the OpenType standard for font resources [12] allows for many different kinds of parameters beyond the basic three of width, height, and depth, such as multiple baselines, and a much richer notion of ligature. Our new engine for micro-typography will provide new capabilities, adaptable to new kinds of parameters, and increased control. Thus we shall be able to provide a simple high-level interface that takes advantage of new developments in font technologies.

Finally, this proposed model should be understood as the preparation for a much more ambitious project, that will deal not just with low-level type-

setting but also with general problems of document structuring and layout for demanding typographic designs in a highly automated environment. Detailed discussion along these lines has already been initiated between the Ω and L^AT_EX3 projects, which look forward to these wider horizons.

References

- [1] Jacques André. *Création de fontes en typographie numérique*. Documents d'habilitation, IRISA+IFSIC, Rennes, 1993.
- [2] David Carlisle, Frank Mittelbach and Chris Rowley. New interfaces for L^AT_EX class design, 1999. <http://www.latex-project.org/papers/tug99.pdf>
- [3] Yannis Haralambous. Unicode et typographie : un amour impossible. *Document numérique* 6(3-4):105-137, 2002.
- [4] Yannis Haralambous and John Plaice. Traitement automatique des langues et composition sous Omega. *Cahiers GUTenberg* 39-40:1-28, 2001.
- [5] P. Karow. *hz-Programm, Mikrotypographie für den anspruchsvollen Satz*. Gutenberg-Jahrbuch, Mainz, 1993.
- [6] D. E. Knuth. *The Art of Computer Programming*. 3 vol., third ed., Addison-Wesley, 1997.
- [7] D. E. Knuth. *Computers and Typesetting*. 5 vol., Addison-Wesley, 1986.
- [8] D. E. Knuth and M. F. Plass. Breaking paragraphs into lines. *Software — Practice and Experience* 11(11):1119-1184, 1981.
- [9] Frank Mittelbach and Chris Rowley, 1996. Application-independent representation of text for document processing. <http://www.latex-project.org/papers/unicode5.pdf>
- [10] Frank Mittelbach and Chris Rowley. Language information in structured documents, 1997. <http://www.latex-project.org/papers/language-tug97-paper-revised.pdf>
- [11] Omega Typesetting and Document Processing System. <http://omega.cse.unsw.edu.au>
- [12] OpenType. <http://www.opentype.org>
- [13] John Plaice and Yannis Haralambous. Generating multiple outputs from Omega. EuroT_EX 2003 proceedings, *TUGboat*, 2003. To appear.
- [14] John Plaice, Yannis Haralambous and Chris Rowley. An extensible approach to high-quality multilingual typesetting. In *RIDE-MLIM 2003*, IEEE Computer Society Press, 2003.
- [15] John Plaice and Joey Paquet. Introduction to intensional programming. In *Intensional Programming I*, World-Scientific, Singapore, 1996.
- [16] John Plaice, Paul Swoboda and Ammar Alamar. Building intensional communities using shared contexts. In *Distributed Communities on the Web*, LNCS 1830:55-64, Springer-Verlag, 2000.
- [17] John Plaice and William W. Wadge. A new approach to version control. *IEEE-TSE* 19(3):268-276, 1993.
- [18] Paul Swoboda. *A Formalization and Implementation of Distributed Intensional Programming*. PhD Thesis, The University of New South Wales, Sydney, Australia, 2003.
- [19] Paul Swoboda. *Practical Languages for Intensional Programming*. MSc Thesis, University of Victoria, Canada, 1999.
- [20] Unicode Home Page. <http://www.unicode.org>
- [21] Extensible Markup Language (XML). <http://www.w3c.org/XML>
- [22] The Extensible Stylesheet Language (XSL). <http://www.w3c.org/Style/XSL>

dvipdfmx, an eXtension of dvipdfm

Jin-Hwan Cho

Korea Institute for Advanced Study

chofchof@ktug.or.kr

In this presentation I would like to introduce a DVI to PDF translator, *dvipdfmx* (formerly *dvipdfm-cjk*), which is an extension of *dvipdfm* developed by Mark A. Wicks.

One might ask why we consider a DVI to PDF translator at this time, since we already have the powerful \TeX software *pdf \TeX* , which generates PDF results directly from \TeX sources without using the DVI format. It is true for people using languages which make use of the Latin alphabet (or other 8-bit character set) that *pdf \TeX* is usually sufficient.

However, the situation is quite different for those who use Northeast Asian languages (Chinese, Japanese and Korean; simply CJK) or Unicode using 16-bit characters. The current version of *pdf \TeX* has no ability to handle 16-bit characters. Even though a PDF viewer shows 16-bit characters in a PDF file generated by *pdf \TeX* , the codes are not 16-bit but 8-bit. Thus, extracting and searching those 16-bit characters is impossible. Furthermore, it is hard to generate a PDF file with *pdf \TeX* having bookmarks or text annotations with 16-bit characters.

That is the main reason why I am introducing *dvipdfmx*. The DVI driver software, *dvipdfmx*, handles 16-bit character using CID-keyed font technology which is already included in the PDF specification. Therefore, *dvipdfmx* works well with almost all \TeX variants including ASCII *p \TeX* , the most popular \TeX software in Japan, and Omega. In particular, it is interesting to see a PDF example containing 16-bit characters from dozens of different languages, which are extractable and searchable as a matter of course.

Recently there was revolutionary progress in developing *dvipdfmx*, namely when *dvipdfmx* began to support *Con \TeX t*. Much of *dvipdfmx* was rewritten at this point. At present, *dvipdfmx* handles many *Con \TeX t* documents containing complex MetaPost figures (color shading too) and interactive forms (JavaScript too). I would like to show those fantastic examples in the presentation.

There are also many features in *dvipdfmx* not mentioned above, PDF encryption for example. More information on *dvipdfmx* can be found at the project home page, <http://project.ktug.or.kr/dvipdfmx>. The *dvipdfmx* project is a combination of the *dvipdfm-jpn* project by Shunsaku Hirata and its modified version, *dvipdfm-kor*, by Jin-Hwan Cho.

Literate programming meets UML

Dr. Alun Moon

School of Informatics

University of Northumbria

Newcastle upon Tyne, UK

`a.lun.moon@unn.ac.uk`

Abstract

This work is an ongoing small project to apply the benefits of *literate programming* to UML. Literate programming is a powerful tool in that it places the emphasis on the documentation of the algorithm, and allows the code to be developed in a logical order. UML is a useful graphical notation to describe features of a software system. However, it lacks the ability to document the code and algorithm in detail. This gap can be filled by literate programming. Elements of UML can usefully enhance the documentation part of a web, with “a picture worth a thousand words”. Finally the process of *tangling* a web into a program is applied to the UML to create a final diagram from fragments throughout the web. The diagrams are ‘enhanced’ by having \TeX available to typeset the text.

1 Introduction

Literate programming is a powerful tool in that it places the emphasis on the documentation of the algorithm, and allows the code to be developed in a logical order. UML (Uniform Modeling Language) is a useful graphical notation to describe features of a software system. However, it lacks the ability to document the code and algorithm in detail. This gap can be filled by *literate programming*. Elements of UML can usefully enhance the documentation part of a web, with “a picture worth a thousand words”.

METAPOST has been used to develop the graphical part of the system; macros for \TeX are included in the web document. METAFONT has all the geometrical tools to allow a diagram to be built up, and its equation solving mechanism allows the elements to be defined in relation to each other. METAPOST also has facilities for typesetting text, making it the suitable tool to use.

1.1 No existing packages

Existing packages on CTAN such as PSTricks have many of the layout tools and arrow decoration needed for UML. This project is in part a learning exercise in writing METAPOST and \TeX macro packages. The \TeX components are written for plain \TeX , as this is what CWEAVE produces.

2 Conventions

These tools were developed with Java in mind as the language. Java and UML feature heavily in the

teaching within the School at Northumbria University. Some form of literate programming may be introduced to the undergraduates, if only just the concept of writing documentation, to help emphasise design in software engineering.

Although Java allows multiple classes in a source file, for the purposes of this tool only one is allowed. Each web file generates one Java file, which compiles to one class. Multiple classes may be possible later. This keeps the management of the diagram elements simple.

3 Design of the macros

The initial set of macros have a slightly *object oriented* feel about them. Class names are used as suffix parameters making a readable file. As the diagrams become more complex, additional data structures are used to ease processing by METAPOST. The \TeX macros write material to a `.uml` file which is post-processed to create METAPOST input files, much as an index is processed with `makeindex`.

3.1 Tangled or Weaved?

Are UML diagrams tangled or weaved? The answer is a bit of both. They are weaved as they form part of the documentation, and include \TeX material. They are tangled as the material is defined in the order of the web file, but has to be rearranged into a program or hierarchical order.

```

\def\private{${-}$} \def\public{${+}$}
\def\package{${\phantom{+}}$}
\def\protected{${\sim}$}

\def\classformatproperties#1{%
  \vbox{\halign{##\hfil\cr #1 }}}

% List macros after Knuth in
% The TeXbook, page 378
\def\leftlist#1{%
  \def\##1{\relax##1\cr}%
  \vbox{\halign{##\hfil\cr#1}}}

\def\classformatlist#1{\leftlist#1}

```

Figure 1: T_EX macros for class diagram contents.

4 Class diagrams

The T_EX and METAPOST macros are shown in figures 1 and 2.

The METAPOST class is built up as a picture. The `class` macro takes three arguments: pictures for the title, attributes and operations of the class. These are given as `btex...etex` formatted pictures. Once all the attributes and operations are known, the class has a fixed size. The code declares three points as suffixes to the class name. The pair `reg` is a registration point, used to position the class when finally drawing it. The two pairs `top` and `bot` are points to connect inheritance arrows to. The picture variable `pic` holds the picture of the formatted class for drawing. The points for the inheritance arrows are a fixed distance from the left edge of the class only because I prefer to align the edges of the boxes.

The T_EX macros are used to format the contents of a class. There is a set of symbols for the access qualifiers, to allow for easy alignment. The attributes and operations can be formatted using the `\classformatproperties` macro, where the elements are separated by `\cr` tokens. The `\classformatlist` macro formats a list of elements, with the list in the form suggested by Knuth in *The T_EXbook* (Knuth, 2000, p. 378).

4.1 Alignment

The attributes and operations are aligned in a `\vbox` using `\halign`. One of the macros above must be used. The T_EX macros writing the `.uml` file write out fragments of METAPOST. If the `\halign` macro was used then the `#` symbol in the template is expanded by `\write` to `##`.

```

vardef class@#(expr title)(expr attributes)
  (expr operations) :=
  save x,y;
  scantokens("pair " & str @# & " top");
  scantokens("pair " & str @# & " bot");
  scantokens("pair " & str @# & " reg");
  scantokens("picture " & str @# & " pic");
  @#pic := nullpicture;
  @#reg + right scaled 1cm = @#top;
  @#top-z0 = @#bot-z6;
  pen ln; ln = pensquare scaled 1pt;
  z0 = origin;
  x1-x0 = x3-x2 = x5-x4 = x7-x6
    = max(width title, width attributes,
          width operations, 2cm) + 1pc;
  x0 = x2 = x4 = x6;
  y0-y1 = y2-y3 = y4-y5 = y6-y7 = 0;
  y0-y2 = 1.5pc + height title;
  y2-y4 = 1pc + height attributes;
  y4-y6 = 1pc + height operations;

  addto @#pic doublepath z0--z1--z7--z6--cycle
    withpen ln;
  addto @#pic doublepath z2--z3 withpen ln;
  addto @#pic doublepath z4--z5 withpen ln;
  addto @#pic also title shifted (z2+(.5pc,.75pc));
  addto @#pic also attributes shifted
    (z4+(.5pc,.5pc)-llcorner attributes);
  addto @#pic also operations shifted
    (z6+(.5pc,.5pc)-llcorner operations);
enddef;

```

Figure 2: METAPOST code for a class.

4.2 Example

An example class diagram is shown in figure 3, and the code that generated it in figure 4.

5 Sequence diagrams

The sequence diagram has been developed in a simple human-friendly form, and a complex machine form. The simple form allows simple sequence diagrams to be drawn. There is a limitation: only one method per class can be drawn.

Unlike class diagrams where classes can be laid out on a grid, elements of sequence diagrams affect not only the position but also the size of other elements. For this reason the points that form an element must be declared before it can take part in the diagram. Sequence diagrams have three main sections in the code: declaration, creation and drawing.

6 Modifying CWEB

The original plan was to modify CWEB to work with Java and UML. This has not been pursued as the author has learned much more about CWEB. The modifications if any are likely to be minor, and there may be a better route using T_EX macros or other tools, for instance:

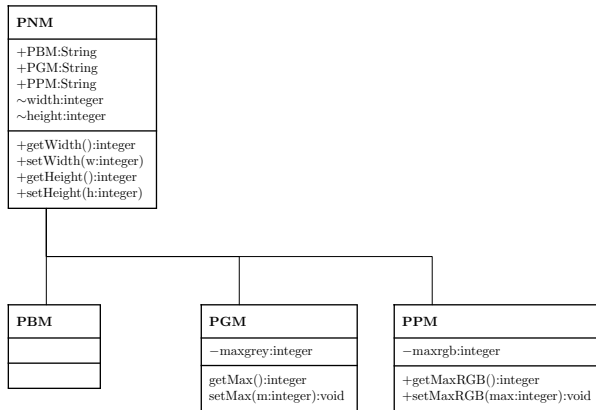


Figure 3: Sample class diagram.

- CWEB produces C++, which is close enough to Java. A web file using the @s mechanism to modify the syntax to Java is given in appendix A.
- UML creation can be done largely through \TeX macros via an intermediate .uml file, just as indexes are produced to be read as a set of macros, after sorting and cross-referencing.
- By choosing good macro names and calling conventions, a language such as Perl can be very useful, especially if helpful data is put into comments in the web source and intermediate files.
- A simple sed script (`sed -e 's/^#\//\//'`) converts the # line pragmas into line comments. (Can anyone come up with a version of javac that can make use of the # line pragmas?)

7 Web UML meta-tools

The web meta-tools for UML are currently in a primitive state. Most of the effort is currently on getting a good set of \TeX macros. The METAPOST data structures are undergoing a major revision which fundamentally changes the internals of the tools. Two tools are needed to do the tangling:

- class builder — to collect attribute and operation lines and write the \TeX /METAPOST class macro.
- sequencer — to arrange the sequences, write all the sections, declarations, creation, and drawing.

8 Data structures and macros

The data structures and macro calling conventions are undergoing a major revision. The macros presented here work well, but have a limiting simplicity, especially the sequence diagram, which has the following limitations:

```

beginfig(0)
class.pnm(btex \bf PNM etex)
(btex \classformatlist{
  \{\public PBM:String\}
  \{\public PGM:String\}
  \{\public PPM:String\}
  \{\protected width:integer\}
  \{\protected height:integer\}} etex);
(btex \classformatlist{
  \{\public getWidth():integer\}
  \{\public setWidth(w:integer)\}
  \{\public getHeight():integer\}
  \{\public setHeight(h:integer)\}} etex);

class.pbm(btex \bf PBM etex)(btex ~ etex)
(btex ~ etex);

class.pgm(btex \bf PGM etex) (btex \classformatlist{
  \{\private maxgrey:integer\}} etex)
(btex \classformatlist{
  \{\getMax():integer\}
  \{\setMax(m:integer):void\}} etex);

class.ppm(btex \bf PPM etex) (btex \classformatlist{
  \{\private maxrgb:integer\}} etex)
(btex \classformatlist{
  \{\public getMaxRGB():integer\}
  \{\public setMaxRGB(max:integer):void\}} etex);

pnm.reg = origin;
pnm.bot - pbm.top = (0,1in);
ppm.reg - pgm.reg = pgm.reg - pbm.reg = (2in,0);

forsuffixes $=pnm,pbm,pgm,ppm: drawclass$ ; endfor;

draw pbm.top connect pnm.bot ;
draw pgm.top connect pnm.bot;
draw ppm.top connect pnm.bot;
endfig;
  
```

Figure 4: METAPOST code for a class diagram.

- only one call per sequence element can be made;
- each sequence element can be called by only one other.

This is due to the use of suffix names for the elements.

8.1 Revised structure

In the revised structure a sequence block would be referred to as, for instance, l_2s_3 , meaning the third sequence block down in the second swim-lane. This makes for nearly unreadable METAPOST code for a complex diagram, but does allow complex diagrams to be built by the meta-tools. Losing the name to refer to an element allows no restrictions on the number of calls to an operation.

```

vardef sequ@#(text call_list) =
  @#.n = .5[@#.nw,@#.ne];
  @#.s = .5[@#.sw,@#.se];
  @#.ne - @#.nw = @#.se - @#.sw = @#.ce - @#.cw
  = @#.re - @#.rw = (seq_width,0);
  @#.nw - @#.cw = @#.rw - @#.sw = @#.ne - @#.ce
  = @#.re - @#.se = (0,seq_width);
  @#.nw - @#.sw = (0,whatever);
  if (length(str call_list) >0):
    @#.ce + (seq_space,0) = call_list.nw;
    @#.re + (seq_space,0) = call_list.sw;
  else:
    @#.ce = @#.re;
  fi;
enddef;

```

Figure 5: Sequence diagram element.

```

declaresequence.main; declaresequence.bezier;
declaresequence.bernstein; declaresequence.binomial;
declaresequence.fact;

sequ.main(bezier); sequ.bezier(bernstein);
sequ.bernstein(binomial); sequ.binomial(fact);
sequ.fact();

main.nw = origin;

beginfig(0)
  pickup pensquare scaled 1pt;
  drawsequence.main;
  drawsequence.bezier; drawsequence.bernstein;
  drawsequence.binomial; drawsequence.fact;
  drawarrow main.ce--bezier.nw;
  drawarrow bezier.ce--bernstein.nw;
  drawarrow bernstein.ce--binomial.nw;
  drawarrow binomial.ce--fact.nw;
endfig;

```

Figure 6: Sequence diagram usage.

9 Teaching

CWEB is being introduced to colleagues in the school and suggested for use on a Masters in embedded systems. There are issues in relation to UML as ANSI C or MISRA C are the preferred choices of language. Is there a neat way of generating header files without too much repetition in the WEB source?

Literate programming has also been suggested as a way to help undergraduate students think about the design (engineering) of program code, by concentrating on the documentation rather than the coding.

References

Knuth, Donald. *The T_EXbook*. Addison-Wesley, 2000.

A Java web file

```
% NULL->null
```

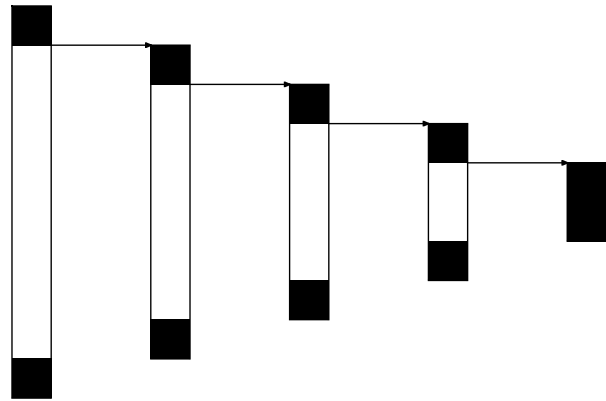


Figure 7: Sequence diagram.

```

@s null NULL

% Java keywords *not* in CWEB
@s abstract int      @s interface int
@s boolean int      @s native int
@s byte int         @s package int
@s extends int      @s strictfp int
@s final int        @s super int
@s finally if       @s synchronized int
@s implements int   @s throws int
@s import include   @s transient int
@s instanceof sizeof

% CWEB keywords *not* in Java
@s and variable      @s namespace variable
@s and_eq variable  @s not variable
@s asm variable      @s not_eq variable
@s auto variable     @s offsetof variable
@s bitand variable  @s operator variable
@s bitor variable    @s or variable
@s bool variable     @s or_eq variable
@s clock_t variable @s pragma variable
@s compl variable   @s ptrdiff_t variable
@s const_cast variable @s register variable
@s define variable   @s reinterpret_cast variable
@s defined variable  @s sig_atomic_t variable
@s delete variable   @s signed variable
@s div_t variable    @s size_t variable
@s dynamic_cast variable @s sizeof variable
@s elif variable     @s static_cast variable
@s endif variable    @s struct variable
@s enum variable     @s template variable
@s error variable    @s time_t variable
@s explicit variable @s typedef variable
@s export variable   @s typeid variable
@s extern variable   @s typename variable
@s FILE variable     @s undef variable
@s fpos_t variable   @s union variable
@s friend variable   @s unsigned variable
@s ifdef variable    @s using variable
@s ifndef variable   @s va_dcl variable
@s include variable  @s va_list variable
@s inline variable   @s virtual variable
@s jmp_buf variable  @s wchar_t variable
@s ldiv_t variable   @s xor variable
@s line variable     @s xor_eq variable
@s mutable variable

```

Math in ConTeXt: Bridging the gap with (AMS-)L^AT_EX

Giuseppe Bilotta

Dipartimento di Matematica e Informatica

Università di Catania

viale A. Doria, 6

95125 Catania

Italy

`gip.bilotta@iol.it`

Abstract

The core of ConTeXt development is focused on the textual aspects of typography in T_EX: many features are provided that easily compete with and surpass those of other high-level T_EX formats, like L^AT_EX. Progress has been lagging, though, in the field of mathematical typesetting. This talk presents a work-in-progress whose final purpose is to provide the features of the most common and powerful mathematical packages in L^AT_EX (AMS and Nath) in the form of ConTeXt modules, possibly with the addition of new features in the spirit of interactivity and graphical richness which is typical of ConTeXt's textual features.

Introduction

One of the strongest points of ConTeXt is its extensive capability to deal with text documents, which make it the most appropriate format for nontechnical writings. At the same time, this is also one of its weakest points: the development of text-based features has been done at the expense of mathematical capabilities, which have therefore made ConTeXt less appealing in technical/scientific environments, where L^AT_EX is still the preferred format. The core math capabilities in ConTeXt have in fact been for a long time barely superior to those of plain T_EX, making math cumbersome to type (at least for those coming from a L^AT_EX background).

At one time, a ConTeXt module (`m-math`), developed by Taco Hoekwater, brought to ConTeXt most of L^AT_EX's environments, macros, and mathematical font capabilities, with additional features from even more packages (notably `breqn`). However, the module made extensive changes to some core macros, especially font-related ones. Thus, when ConTeXt underwent a thorough redesign (with a completely new font loading/selection mechanism, based on typescripts) the `m-math` module was effectively broken.

It is the writer's intention to bring powerful and easy math back to ConTeXt. In particular, we aim for:

1. providing the functionality of (AMS-)L^AT_EX and Nath, with as much ease if not necessarily the same syntax;

2. providing as much command compatibility as possible, so that compatible commands/environments are achieved with the same or very similar commands;
3. (low priority) adding new features without breaking anything, in the spirit of graphics and interactivity peculiar to ConTeXt;
4. (low priority) ensuring that documents look the same (or as similar as possible) to those obtained with (AMS-)L^AT_EX when no extra ConTeXt features are used (to customize section headings, itemizations, numbering, etc.).

Project status

The major overhaul that broke the `m-math` package also provided the basis for potentially augmenting ConTeXt's math capability. Much work has been done to offer in ConTeXt a symbol set as extensive as that of L^AT_EX, and basic math environments have been provided. A new math module (`m-newmat`) has been developed, as a placeholder to add new features as the need arises.

Starting from this base, I'm developing two new packages: `t-amsl`, focused on AMS-L^AT_EX compatibility, and `t-nath`, to bring the ease of use and power of Nath (a package developed by Michal Marvan, presented at the EuroT_EX 2001 conference, implementing NATURAL maTH notation) to ConTeXt.

Nath is almost fully implemented. This has been obtained by using the same source as the L^AT_EX

package itself, with due adaptation. Some interesting side-effects of the adaptation of Nath to ConTeXt were the discovery of a couple of bugs, and some format-independent work on robustness, both macro-wise (allowing for example extensible arrows in the sub- or superscript part of another extensible arrow) and engine-wise (making Nath aware of ε -T_EX and therefore allowing it to typeset more complex formulas, a job which requires a notably large number of registers).

Work to date on the AMS macros is much less extensive: it currently implements some basic environments (equation alignment and gathering) and some classic macros like `\eqref`. It also provides some non-AMS but important L^AT_EX math environments like `array`; some of these may be moved to the `m-newmat` module in the future, to leave AMS-specific macros only in the `t-ams1` module.

Finally, the module restores (in math mode) L^AT_EX-style behavior of a few font-selection commands; selection mechanisms for such fonts were already present in ConTeXt, but they were text-centric (quod erat demonstrandum) and therefore cumbersome to use in math mode; `t-ams1` makes them again available in math mode as well, with the familiar `\mathcal`, `\mathfrak`, etc. command interface (as well as `\cal`, `\frak`, etc.).

Project future

On the one hand, one might think that there is still much work to do, (re)implementing all the various L^AT_EX and AMS-L^AT_EX environments; on the other hand we should consider the level of compatibility we actually want between the packages.

As a first step, it is important to provide the same typesetting power, as easily as or more easily than in L^AT_EX. For example, the advanced math typesetting features of Nath make many of the AMS environments unnecessary. We therefore prefer to concentrate initially on completing the port of Nath.

After this has been provided, and for the remaining needs which are not dealt with by Nath, we will move to improving command compatibility with the AMS-L^AT_EX environments, so as to let the transition from one typesetting environment to the other be as smooth and painless as possible. If possibly, aesthetical compatibility will be preserved (or created as necessary), to allow ConTeXt-typeset documents to be usable for standard journal submissions.

When choosing whether to be compatible with one system or the other, in some cases the L^AT_EX way of doing things will be abandoned in favour of the ConTeXt one, when the latter makes more sense or is easier to manipulate, from the user's perspective and in the author's opinion. For example, for theorem creation and management the rigid positional configuration options of AMS-L^AT_EX will be abandoned in favour of the dynamic key/value configuration capabilities which are standard in ConTeXt; the L^AT_EX form might still be provided for compatibility, but this will have a lower priority.

Finally, new features might be included to allow typesetting of more "ConTeXtual" math formulas: features like interactive formulas (click to cycle through passages), or "hidden" explanatory passages that display in pop-up windows will be implemented (some of these are already in the works).

POV-ray: A 3D graphics tool for T_EX

Kaveh Bazargan, CV Radhakrishnan,
CV Rajagopal
Focal Image (India) Private Ltd
kaveh@focalimage.com, cvr@focalimage.com,
cvr3@focalimage.com

POV-ray (<http://www.povray.org>) is a 3D image renderer. T_EX is a text typesetting engine. The two programs have much in common: Freely available; multi-platform; unrivalled output quality; text-based input; ability to read and write files; fully programmable. We will show how these two programs can work together to make beautiful documents. In particular, we will concentrate on how T_EX can use POV-ray to add photorealistic embellishments to textual documents.

◇

The teT_EX distribution

Thomas Esser
dvg Hannover, Germany
te@tug.org

This talk is about teT_EX. I will explain what teT_EX is, and my role in the development of teT_EX and T_EX Live. Some of my own contributions (e.g. configuration tools: texconfig, updmap, fntutil) are explained in more detail.

◇

T_EXPower: Dynamic presentations with L^AT_EX

Stephan Lehmké
QuinScape GmbH
Dortmund, Germany
Stephan.Lehmke@QuinScape.de
<http://www.QuinScape.de>

In the talk, a bundle of L^AT_EX packages and classes is presented which provides an environment for designing *dynamic pdf presentations*, mainly for the purpose of displaying with a video beamer.

The heart of the bundle is the texpower package, providing:

1. commands for *incremental display* of page contents;
2. commands for designing page backgrounds and ‘panels’;
3. commands for *navigation helpers*.

As the effects provided by texpower are implemented entirely based on the L^AT_EX kernel, without resorting to special effects like PostScript, T_EXPower is independent of the method of *pdf generation* and does not rely on external postprocessors or such. It is also completely independent of the *document class* used, though seminar-based classes harmonising well with the texpower package are part of the bundle.

Because of the unique way incremental display is implemented, it is sometimes harder in T_EXPower than in other similar packages to keep ‘static’ parts of the page from “moving around” during display. In fact, almost all pitfalls can be avoided by adhering to a number of simple design rules, which will be pointed out in the talk.

The upside of “doing it all in T_EX” is the unique flexibility and customizeability of

- the *order* in which things are displayed and
- the *way* in which hidden/appearing things are displayed.

Concerning the order of display, the full range of L^AT_EX’s abilities for constructing case distinctions can be employed.

Concerning the way of hiding and displaying things, the possibilities range from things just appearing out of blank space via objects being replaced by other objects to hidden text being displayed with dimmed colors and undimming incrementally. Special effects such as objects flying around or growing into place are also possible, limited only by the algorithmic capabilities of T_EX (and the performance of the computer running Acrobat Reader).

T_EXPower is currently in a pre-alpha state and will probably stay that way for some time, but the development release is quite stable and usable. The project web site <http://texpower.sourceforge.net> gives easy access to updates and communication with developers.

(We expect to publish the full paper in the next regular issue of *TUGboat*. *Ed.*)

XemTeX: An integrated platform for high quality scientific typesetting

Fabrice Popineau

Fabrice.Popineau@supelec.fr

Marie-Louise Chaix

Marie-Louise.CHAIX@edpsciences.org

This talk will describe a project funded by the French Ministry for Education. This project aims at building a tightly integrated TeX+XEmacs distribution which will be distributed to French high schools. There is a growing demand by mathematics and physics teachers for a TeX-based solution. The first target platform will be Windows, Linux being second. In order to draw as many people as possible to TeX, even novices, they must be provided with a package up to the standards of most word processors: the users will be exposed to only one application and not to dozens of binaries. Another point that keeps many people from using TeX is not so much the (L^A)TeX language itself, but that maintaining a TeX distribution is difficult and can be time consuming.

The X_eTeX project has been submitted and accepted for funding to build a free platform that should be much easier to use than the current ones, based on the XEmacs editor and a subset of the current TeX Live distribution. The project will address several problems, including creating an enhanced XEmacs mode for typesetting TeX documents, tightly integrating the viewer into XEmacs and documenting the product. These points will be addressed in this talk, as well as the current status of the project, and possibly how to get funding for such projects.

TeX on Mac OS X using teTeX and TeX Live

Gerben Wierda

Sherlock@rna.nl

Mac OS X is the successor to Mac OS (a.k.a. Mac OS Classic). Mac OS X is based on a modern open source Unix foundation (though most Mac OS users will not be aware of this) and as such is currently the most widely used desktop Unix. Since it is indeed a Unix, the famous TeX distribution by Thomas Esser (teTeX) may be used as a TeX engine.

Mac OS X differs with other Unixes in several ways. Technically, there are differences most noticeably at the file system level, the text format level and the graphical display level. With respect to the latter, Mac OS X display technology is entirely based on PDF, and as such it is a system where pdfTeX is really “at home”.

But at least as important, there is a difference between Mac OS X *users* and users of other Unix desktops, in that they are generally far less ‘computer literate’ at the technical level. Anything presented to typical Mac OS users should follow the motto “it just works” (and without any use of Unix-level technologies like shells). This includes handling complexities like updating TeX on a regular basis without having detailed knowledge of the technicalities involved.

Bringing TeX to Mac OS X has therefore been a complex project with hurdles and pitfalls on many levels. The talk will present some of these hurdles and the solutions inspired by them, some of which are solutions reached at in collaboration with others or more often entirely created by others.

Experiences and lessons learned teaching L^AT_EX to university students

Gary L. Gray

Associate Professor
Department of Engineering Science and Mechanics
The Pennsylvania State University
212 Earth & Engineering Sciences Building
University Park, PA 16802
USA
gray@engr.psu.edu
<http://www.esm.psu.edu/faculty/gray/>

Francesco Costanzo

Associate Professor
Department of Engineering Science and Mechanics
The Pennsylvania State University
212 Earth & Engineering Sciences Building
University Park, PA 16802
USA
costanzo@engr.psu.edu
<http://www.esm.psu.edu/faculty/costanzo/>

Abstract

This paper will describe our experiences and lessons learned while teaching L^AT_EX to a class of students (undergraduate and graduate students) during the fall 2001 semester at The Pennsylvania State University. This was a one-credit course taken by 9 undergraduate students (all were juniors or seniors) and 15 graduate students. We will discuss what material was covered in class, what resources were used in preparing the material, and what assignments were given to the students. In addition, we will discuss those materials and assignments that proved to be useful and those that were not so useful. We will discuss the lessons (both pedagogical and L^AT_EX-related) learned by us. In addition, based on our experiences and feedback provided by the students, we will present those lessons learned by the students and their recommendations for improving the class in the future. Finally, we will give our wisdom and recommendations to those instructors who might wish to teach a similar class at their institution.

Introduction

We supervise the work of both graduate and undergraduate students in our group at Penn State University (PSU) and are, therefore, frequently reading, correcting, and sharing technical documents* with these students. In addition, the work on which we collaborate with our students is generally turned into one or more journal publications. Hence, more often than we would like, we have to face the fact that almost all students are “brought up” on Micro-

soft WordTM and so their first instinct when joining our group is to use Word for all of their writing.[†]

While we recognize that reasonably nice output can be obtained with Microsoft Word with the proper use of its styles, equation editor, text boxes, and the like, we have found that not only do students not know how to use these tools, but they do not even know that they exist. We could, of course, either teach them to use these tools or require that they learn them on their own, but we are rather enthusiastic users and evangelists of L^AT_EX and are not interested in translating everything they do in

*When we use the term *technical document*, we mean a document with many equations and figures.

[†]Amazingly, they even use Word to draw figures, but that is a topic for a different paper.

Word into L^AT_EX. In addition, our (rather extensive) experience with Word has been that it does not handle long documents (such as dissertations) well, does not handle floats well, is prone to file corruption, and frequently exhibits behavior that can only be explained by postulating the existence of supernatural forces. Therefore, we have a substantial incentive to teach our students how to use L^AT_EX in their work.

In addition, we are frequently asked by students who are taking our classes and who see the rather nice-looking course handouts that we generate using L^AT_EX, what we used to create the handouts. When they hear about L^AT_EX and the facility with which it handles technical documents, many are intrigued to learn more. These students have asked us many times if we would teach a course on how to use L^AT_EX.

With all of these motivating factors in mind, the stars finally aligned in the fall 2001 semester and we had the opportunity to teach a course about L^AT_EX entitled *Technical Documents with L^AT_EX* to a group of graduate and upper level undergraduate students. In what follows, we will attempt to convey not only what we taught in the course, but how we made the decisions to do what we did with the hope that this may be useful to anyone wanting to teach a similar course elsewhere.

Class Structure and Organization

The course we taught in the fall 2001 semester was a one-credit course that met once per week for 75 minutes.* We met in a classroom in which each student had a laptop computer with (L^A)T_EX and with access to the web. The required text for the course was the 3rd edition of Kopka and Daly's excellent book on L^AT_EX (Kopka and Daly, 1999), though we also suggested that each student obtain a copy of Grätzer's book that nicely covers $\mathcal{A}\mathcal{M}\mathcal{S}$ -L^AT_EX (Grätzer, 2000). The course met 7 weeks of our full 15 week semester.

T_EX Resources at Penn State At the time the course was offered, we had recently switched to Mac OS X as our primary operating system, but our university computing labs had not yet done so and were still running Mac OS 9. We used, and very much liked, the combination of T_EXShop (Koch, 2003) as a front-end and Wierda's T_EX distribution (Wierda, 2003), so we had to make the decision on what implementation of T_EX to have our Center for Academic Computing (CAC) install in our computing labs. One of the authors had a little experience with

both CMacT_EX (Kiffe, 2002) and OzT_EX (Trevorrow, 2002), and since, at the time, CMacT_EX was available for both Mac OS 9 and Mac OS X, we decided to go with CMacT_EX.

Penn State has an extensive system of public computing labs and we would estimate that approximately 20% of the computers are Mac OS-based and the remainder are Windows-based machines. Since neither author had extensive experience with the installation or use of T_EX under Windows, we chose not to ask CAC to install T_EX under Windows since neither of us was likely to be able to answer any questions that might come up. We ended up demonstrating CMacT_EX on the first day of class and told the students that they are, of course, free to install L^AT_EX on their home or lab computer. We told them that installations are available for virtually every operating system, though we only had experience with T_EX on Macs, so if they needed help with another OS, they would have to see our web site for a list of resources.

Administrative Details In the advertisement for the class and on the first day of class, we told students that we would:

- Give them an introduction to the typesetting language L^AT_EX through the use of tutorials, example documents, and homework assignments.
- Show them how to easily write a professional-looking dissertation, conference paper, and/or journal paper. We emphasized the word “looking” because the content is up to them.
- Show them how to create a professional-looking presentation (such as this) with L^AT_EX.

When we taught the class, we had a combined 19 years of experience with L^AT_EX so it was clear that we could not teach the students everything we knew about it. In addition, our experience was almost entirely as L^AT_EX users and not as L^AT_EX programmers (though this course turned out to be a good excuse to learn a little about programming in L^AT_EX — more on that later), so the knowledge we would convey to the students was going to be of a very practical nature. Our goal for the course was to get the students started and to point them to the numerous other resources that are available for help with and information about L^AT_EX.

There were weekly homework assignments and all homework was to be handed in electronically. In addition, the students certainly needed to be comfortable with a computer.[†] Therefore, we told the students that they needed to be able to:

[†] Contrary to what many of us “old timers” think, many of today's undergraduates only know how to surf the web, send

*The vast majority of courses at PSU are 3-credit courses that meet 2–3 times per week for a total of 150 minutes/week.

- move files between computers (i.e., either by file sharing, ftp, email, or the web);
- download, install, and launch applications (for those students wanting to install and use \TeX on their personal or lab machines);
- use a text editor.

Grades are a necessary evil in every course, and this course was no different. The grade for each student was entirely based on their homework, which was 75% of their grade, and their class attendance, which was 25% of their grade. No exams were given. We chose to make attendance a significant portion of their grade because we knew that a lot of the learning would be done in the classroom and we didn't want students to miss out on that. There was weekly homework and the students always found the next assignment at the end of the current week's lecture (more on the lecture format later). We tried to create homework assignments such that each one would not take more than 2–3 hours to complete. We told the students that if an assignment was taking them more time than that, then they were probably heading in the wrong direction and that they should see us. Despite this, when turning in an assignment, we would have some students tell us that they had spent 9 hours on the assignment and they still had not finished. We can't emphasize enough that this behavior seems to be rather common and is, most certainly, counterproductive. Therefore, it is important to stress to the students that they should not "beat their head against the wall" trying to get these things done—they should seek assistance.

We created a rather simple web site for the course where students could:

- find course announcements;
- download the course information as either a PDF file or the `.tex` course file;
- download the lectures as either a PDF file or the `.tex` course file;
- download the `.tex` source of a number of sample documents with some some reasonably complex formatting (e.g., the ad for the course, the course information, an equation sheet for a sophomore-level course, etc.); and
- find links to \TeX -related resources on the web.

The web site can be found at:

`<http://www.esm.psu.edu/courses/latex-course/>`.

Taking a cue from an old Chicago voting motto, we told students to "visit it early and visit it often".

and receive email, and send and receive instant messages. Even seemingly mundane things like files attached to email messages will perplex some students.

Finally, we wanted students to take the course seriously and didn't want students looking for an easy one credit. We told the students that they had to want to be there to learn \LaTeX and if they were looking for an easy one credit, then they might like to find another course.

Class Content

In creating the course material to be presented, we spent some time looking around on the web to see if anyone had created a similar course. While there were several courses that had been created, they were either in a language other than English or did not cover as much material as we hoped to do. In addition, it is generally the case that it is hard to take someone else's course notes and use them as your own. So, we decided to create the course from scratch, using experience, Kopka and Daly (1999), and Grätzer (2000) as our guides.

Largely following the order of presentation in Kopka and Daly, the seven lectures we created were entitled:

1. Introduction & Basic \LaTeX
2. Displayed Text
3. Typing Mathematics in \LaTeX
4. Multiline Equations in $\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\text{\LaTeX}$
5. Graphics & Floats
6. User Customization & Bibliographies
7. The PSU Thesis Package

Introduction & Basic \LaTeX In the first lecture, we outlined the course objectives, discussed what \LaTeX is and why it is useful for students to know it, and told the students our expectations of them. We then covered a section entitled *Getting Started with \LaTeX* , in which we discussed:

- the overall structure of a \LaTeX document;
- the general structure of \LaTeX commands;
- environments and declarations;
- characters, words, sentences, and paragraphs;
- quotes, hyphens, and dashes;
- classes, class options, and packages;
- page layout (e.g., headers, footers, margins).

The last part of the first lecture was entitled *Putting it All Together*, in which we demonstrated how to put all the elements discussed previously together in order to create a typesettable document. Finally, we demonstrated how to typeset a document and view the resulting output using CMacTeX . The first homework assignment consisted of exercises out of Chapters 2 and 3 of Kopka and Daly.

Displayed Text In the second lecture we talked about:

- understanding and changing font characteristics (i.e., `\emph`, font sizing commands, families, shapes, and series);
- centering and quoting text;
- lists;
- typewriter-like tabs, the `tabbing` environment;
- boxes (i.e., how T_EX defines boxes, `\parbox`, `\rule`);
- tables (we did not cover the `booktabs` package (Fear, 2000) as it has been a recent discovery, but we will do so in the future).

The second homework assignment consisted of exercises out of Chapter 4 of Kopka and Daly.

Typing Mathematics in L^AT_EX The first two lectures were largely based on material from Kopka and Daly (1999), but Lectures 3 and 4 were largely based on Grätzer (2000). We use $\mathcal{A}\mathcal{M}\mathcal{S}$ -L^AT_EX exclusively, so we began by telling the students that everything we would be covering would assume that they had loaded the following $\mathcal{A}\mathcal{M}\mathcal{S}$ -L^AT_EX packages and options (American Mathematical Society, 2000).

```
\usepackage{amsmath}
\usepackage{amssymb}
\usepackage{exscale}
\usepackage[mathscr]{eucal}
```

We then discussed features of L^AT_EX and $\mathcal{A}\mathcal{M}\mathcal{S}$ -L^AT_EX that are relevant for inline mathematics and single-line displayed mathematics. We emphasized to the students that the mathematics in a document is part of the narrative and should be punctuated as such. In addition, we discussed:

- equation numbering;
- arithmetic operations;
- superscripts and subscripts;
- resources for typesetting mathematics (Swanson, 1999; Higham, 1998);
- ellipses, integrals, roots;
- text within mathematics;
- delimiters;
- operators;
- math accents;
- spacing with mathematics;
- math alphabets and symbols (e.g., bold mathematics); and
- generalized fractions.

The third homework assignment consisted of exercises out of Chapter 5 of Kopka and Daly, but with the requirement that $\mathcal{A}\mathcal{M}\mathcal{S}$ -L^AT_EX structures be used when available.

Multiline Equations in $\mathcal{A}\mathcal{M}\mathcal{S}$ -L^AT_EX Again relying on Grätzer (2000) for source material, we then presented an entire lecture on displayed multiline equations using $\mathcal{A}\mathcal{M}\mathcal{S}$ -L^AT_EX. We covered the philosophy behind $\mathcal{A}\mathcal{M}\mathcal{S}$ -L^AT_EX's multiline equation structures and then went on to cover each new environment introduced by $\mathcal{A}\mathcal{M}\mathcal{S}$ -L^AT_EX. We covered:

- grouping formulas and `gather`;
- splitting long formulas and `multline`;
- breaking and aligning formulas;
- numbering of formulas, equation tags, and the `subequations` environment;
- organization of equations into multiple columns via the `align` environment, the `flalign` environment, and the `alignat` environment;
- subsidiary math environments, that is, `split`, `aligned`, `alignedat`, and `gathered`;
- adjusted, multi-column math environments, for example, `matrix`, `cases`, and `pmatrix`

The fourth homework assignment consisted of two handouts: the first was two pages from a 1963 paper from a Russian mathematical journal (Melnikov, 1963) and the second was Section 8.5.1 from a book on numerical linear algebra (Golub and Van Loan, 1989). We asked the students to typeset the pages we had given them. In the case of the paper from the mathematical journal, we wanted students to *improve* the typesetting of the text and equations and in the case of the pages from the book on numerical linear algebra, we wanted the students to simply replicate the layout.

Graphics & Floats The fifth lecture covered the inclusion of graphics in L^AT_EX via its float mechanism. We covered the:

- `graphicx` package (Carlisle and Rahtz, 1999) with its `includegraphics` command and options such as:
 - `scale`
 - `width, height, keepaspectratio`
 - `angle`
 - `bb`
- `lscape` package (Carlisle, 2000);
- importing of graphics and troubleshooting;
- `color` package (Carlisle, 1999), including the `monochrome`, `dvipsnames`, and `usenames` options, and setting the color of a page and text;
- float environments: `figure` and `table`.

The fifth homework assignment asked the students to create a one-page flyer conveying any messages or advertising anything they liked. We told

them that they had learned a fair bit about L^AT_EX, so they should try and make it interesting and creative. We also told them that the flyer had to include some mathematics. In addition to this, the flyers were to include:

- the use of a background color for the page;
- the tasteful and artistic use of a number of colors for the text;
- the use of *at least* three different JPEG images (`.jpg`), at least one of which must be scaled and one of which must be rotated.

User Customization & Bibliographies Material for this lecture came out of various sections of Kopka and Daly (1999); in particular, we asked the students to read Chapter 7, Sections 4.3.6, 8.3.3, and Appendix B. In regard to customizing L^AT_EX, we discussed:

- counters: how to set and reset them, and how to define new counters;
- how L^AT_EX uses lengths and how to: set a length using either `\settowidth` or `\setlength`, define a new length using `\newlength`, and add to a length using `\addtolength`;
- the creation of user-defined commands, both with and without arguments, as well as the redefinition of commands;
- the use of the `\input` command to read in “boilerplate”;
- the scope of commands and environments defined in the preamble versus the scope of those defined within environments.

With regard to bibliographies, we began by discussing the basic and simple environment for generating a bibliography via the `thebibliography` environment. In addition, we talked about how one can change the title of the bibliography using either `\refname` or `\bibname`, depending on the class used. We also discussed the limitations and disadvantages of using the `thebibliography` environment without the aid of BIB_TE_X. We emphasized that BIB_TE_X provides a way to use a database of references (via a `.bib` file), along with a bibliography style definition (found in `.bst` files), to automatically generate bibliographies. This is useful for the following reasons:

- one can maintain any number of reference databases and BIB_TE_X will only use those references it needs; this is especially nice when one uses many of the same references in several different documents;
- one can use the same databases of references and the chosen `.bst` file will format them automatically.

We also briefly discussed the `natbib` package (Daly, 2000) for author-year citations and the use of End-NoteTM (ISI ResearchSoft, 2002) with BIB_TE_X.

The sixth homework assignment consisted of exercises out of Chapters 4 and 7 of Kopka and Daly.

The PSU Thesis Package As preparation for this seventh and final lecture, the authors chose to undertake their first major L^AT_EX customization/programming project by creating a document class conforming to the *Thesis Guide: Requirements and Guidelines for the Preparation of Masters and Doctoral Theses* (The Pennsylvania State University, 2002), which is published by Penn State’s Graduate School. Among many other things, this guide specifies the detailed technical requirements that each thesis or dissertation must satisfy. These requirements include, among other things, the specification of: line spacing, font size, front matter, chapter formatting, margins, page number location, etc. All of these requirements can be rather overwhelming for students who, while trying to implement them, are also desperately trying to actually *write* their thesis. In addition, as we are all aware, L^AT_EX can do a wonderful job of removing the tedium of assembling a Title Page, Table of Contents, List of Figures, List of Tables, Signature Page, and all the other little things that must go into a thesis. The class we created, `psuthesis.cls`, is heavily documented, should be relatively easy for individuals at other institutions to modify, and can be found on one of the pages at the course web site (Gray and Costanzo, 2002).

Our lecture gave an overview of the thesis class and an example thesis template illustrating the use of the thesis class. There was no homework associated with this lecture.

Lecture Format and Creation

We estimate that each 60–75 minute lecture took us anywhere from 4–6 hours to prepare. The first two lectures were prepared as slides, presented as PDF files, using Foil_TE_X (Hafner, 1998). We both found this to be awkward since we had to worry so much about the amount that went on each slide. It was also more difficult to show the “natural” behavior of L^AT_EX since, by their very nature, slides or foils are heavily modified to use larger fonts, landscape orientation, etc. Finally, the slide format impeded us from adding little tidbits and changing the L^AT_EX source during a lecture since, often times, the addition or deletion of one or two words would completely alter the formatting of a slide. Therefore, we decided to use the standard `article` class and create lectures that simply looked like a standard

L^AT_EX article. Before each lecture (sometimes only a few minutes before, but always before) we would post the `.tex` source for the lecture as well as the corresponding `.pdf` typeset output on the class web site. After each lecture, we would re-post the lecture (source and output), thus incorporating any changes to the document that took place during the lecture. Each lecture was presented by connecting a laptop to an LCD projector and using T_EXShop to show the source and preview.

We tried our best not to take the entire class time to talk about L^AT_EX since 75 minutes is a long time to try and focus on someone standing at a computer and showing L^AT_EX source and output. On the other hand, as most instructors will testify, it is nearly impossible not to have one's lecture expand to fill the available time. Therefore, with only two or three exceptions, our lectures took the entire 75 minute time period. When we teach this class in the future, we will unquestionably leave more time to work with the students in the classroom.

Finally, our experience in other classes in which we used this same computing environment, that is, an environment in which each student is at a laptop that is connected to the Internet, has been that it was terribly tempting for students to surf the web, check their email, and/or instant message with their friends rather than listen to what we had to say. Therefore, we made it clear from the first day of class that students were welcome to do all of these things at the computers, they simply were not allowed to do it during our class. We found that the students generally respected our request.

Student Response to the Class

At the end of each semester, it is the policy of our department to have students anonymously evaluate the course they are about to complete. In addition to generic questions about the instructor and the course requiring numerical evaluations, three additional questions are asked for which the students give written answers. These questions are:

- Q1. What did you like best about this course?
- Q2. What did you like least about this course?
- Q3. What suggestions do you have for improving this course?

Question 1 Ignoring responses such as “L^AT_EX was explained well by the instructors”, which, while nice, doesn't really tell us much, student response to the first question emphasized:

- the utility of the web site;
- the teaching of L^AT_EX through examples;

- that the course assumed no prior knowledge of L^AT_EX;
- that L^AT_EX provided an alternative to Microsoft Word; and
- that they were happy to be learning a skill that would be useful in their careers.

Question 2 To the second question, we found the following themes among the student responses:

- there was no instruction on how to use particular software packages for L^AT_EX;
- there was too much work for a 1-credit course;
- some of the homework took them much too long to do;
- there were numerous problems with CAC computer labs that prevented them from doing their homework; and
- the course felt rushed.

Question 3 To the third question, we *overwhelmingly* heard that:

- we should spread the course out for at least 12 weeks, if not the entire semester;
- the homework should be graded more leniently;
- the class should meet more often so that each lecture is shorter;
- the homework should be shorter or spread out more; and
- we should provide help to people with Windows machines.

So, what conclusions can we draw from these comments that would allow us to improve the course the next time we teach it? Well, the responses to the first question tell us that we should: continue to provide resources and information via the class web site, continue to teach using a myriad of examples, and teach the course at a very introductory level. The responses to the second and third questions indicate that:

- too much was taught in too little time and the course should be spread out over a larger part of the semester;
- classes should be shorter and meet more often (this would also allow us to make each homework assignment shorter); and
- we should provide additional support for those students who are having trouble getting T_EX to work on either their own computer or a public computer.

On the other hand, it has been our experience, that in all courses that involve computers and programming, students almost *invariably* find them to be

more time-consuming than they would like. Thus, some of the feedback was not unexpected. We do agree, however, that the course could and should be spread out throughout more of the semester. In addition, we will more strongly encourage students to seek out our assistance rather than beating their heads against the wall.

The one problem the students were having that we will have the most difficulty reconciling is the issue of getting \TeX running on either public computers or personal machines. We hope that we can fix the issue of running \TeX on public machines by using \TeX Shop along with Wierda's $\text{te}\text{\TeX}$ - \TeX Live distribution. As for getting students up and running on their own machines, we will most likely proceed as we did the first time we taught the course, that is, we will tell the students that we will only support the public machines and if they want help on their personal computers, they will have to seek out help on the web.

Thoughts for the Future

Based on student feedback, the utility of (\LaTeX) for academic work, particularly at the graduate level, and our experience teaching this course, we are left with the question: Is there a place for a course like this in the university curriculum? There is no question that the majority of students in the course, despite thinking it was too much work, expressed a great deal of enthusiasm for it. In addition, we have found via a recent follow-up survey of students who took our class that more than half the students in the course continue to use what they learned about \LaTeX on a regular basis. We did find, however, some reservations about offering this course on the part of our department. There were people who felt that this was not the kind of course that should be offered at a university for credit. They felt that it should be offered as an "extra-curricular" activity. Given the importance of publishing in academia, at least for graduate students, and the fact that all graduate students (and many undergraduate students) need to write a substantial thesis or dissertation during their tenure as students, we feel that a practically important course such as this one can be an important part of the curriculum.

We welcome the thoughts and experiences of other instructors in academia on these issues.

Acknowledgements

We would like to thank CAC for providing the licenses for $\text{C}\text{\TeX}$. In addition, we would like to thank all of those students who have provided valuable feedback on our Penn State thesis class.

References

- American Mathematical Society. "The $\mathcal{A}\mathcal{M}\mathcal{S}$ - \LaTeX packages". Available from CTAN, `macros/latex/required/amslatex/`, 2000.
- Carlisle, David. "The color package". Available from CTAN, `macros/latex/required/graphics/`, 1999.
- Carlisle, David. "The `lscap` package". Available from CTAN, `macros/latex/required/graphics/`, 2000.
- Carlisle, David and S. Rahtz. "The `graphicx` package". Available from CTAN, `macros/latex/required/graphics/`, 1999.
- Daly, Patrick W. "The `natbib` package". Available from CTAN, `macros/latex/contrib/natbib/`, 2000.
- Fear, Simon. "The `booktabs` package". Available from CTAN, `macros/latex/contrib/booktabs/`, 2000.
- Golub, Gene H. and C. F. Van Loan. *Matrix computations*. Johns Hopkins Series in the Mathematical Sciences; 3. Johns Hopkins University Press, Baltimore, Md., 2nd edition, 1989.
- Grätzer, George. *Math into \LaTeX* . Birkhäuser, Boston, 3rd edition, 2000.
- Gray, Gary L. and F. Costanzo. "Penn State Thesis Class". Available from <http://www.esm.psu.edu/courses/latex-course/lectures.html>, 2002.
- Hafner, James. "The `FoilTeX` package". Available from CTAN, `macros/latex/contrib/foiltex/`, 1998.
- Higham, Nicholas J. *Handbook of Writing for the Mathematical Sciences*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2nd edition, 1998.
- ISI ResearchSoft. "EndNote". Available from <http://www.endnote.com/>, 2002.
- Kiffe, Thomas R. "`CMacTeX`". Available from <http://www.kiffe.com/cmactex.html>, 2002.
- Koch, Richard. "`TeXShop`". Available from <http://darkwing.uoregon.edu/~koch/texshop/texshop.html>, 2003.
- Kopka, Helmut and P. W. Daly. *A Guide to \LaTeX : Document Preparation for Beginners and Advanced Users*. Addison-Wesley, Harlow, England, 3rd edition, 1999.
- Melnikov, V. K. "On the Stability of the Center for Time-Periodic Perturbations". *Transactions of the Moscow Mathematical Society* **12**, 1–56, 1963.

Swanson, Ellen. *Mathematics into Type*. American Mathematical Society, Providence, RI, updated edition, 1999. Updated by Arlene Ann O'Sean and Antoinette Tingley Schleyer.

The Pennsylvania State University, Graduate School. "Thesis Guide: Requirements and Guidelines for the Preparation of Masters and Doctoral Theses". Available from <http://www.gradsch.psu.edu/enroll/thesisguide.html>, 2002.

Trevorrow, Andrew. "OzT_EX". Available from <http://www.trevorrow.com/oztex/>, 2002.

Wierda, Gerben. "t_eT_EX-T_EXLive Distribution". Available from <http://www.rna.nl/ii.html>, 2003.

The (L)TeX project: A case study of open source software

Alexandre Gaudeul

University of Toulouse

France

`alexandre.gaudeul@univ-tlse1.fr`

Abstract

The TeX typesetting software was developed by Donald E. Knuth in the late 1970s. It was released with an open source license and has become a reference in scientific publishing. TeX is now used to typeset and publish much of the world's scientific literature in physics and mathematics.

This case study serves as a critical examination of the stylized facts uncovered in previous studies of other open source software projects, such as GNU/Linux, an operating system, and Apache, a web server. It is sponsored by CNRS, a French research agency, and is supported by the University of Toulouse in France and the School of Information Management and Systems in Berkeley.

The comparison centers on the historical development of the project, the organization, both formal and informal, that supports it, the motivations of the developers, and the various dynamics that are at work and influence the project.

The case study explores the economic impact of the TeX software which is sold through TeX-based commercial applications and used in the typesetting industry and various institutions. It is an exploration of how the open source nature of the program made a difference relative to what would have happened had it been commercial software.

1 Motivation

I have been working for one year now on a case study of TeX as open source software. Since TeX branched out into many different projects, this case study is in fact a sum of case studies about those different projects, and a reflection on the dynamics of the whole project. This whole project will be called the TeX project or simply 'TeX'. My aim is to provide some elements to improve the way in which open source software projects ('OSSPs') are managed, and also help policy makers gain a better understanding of the open source ('OS') phenomenon.¹ This case study serves as a critical examination of the stylized facts uncovered in previous studies of other open source software projects. Some better known and studied OSSPs are GNU/Linux, Perl and Apache (an operating system, a programming language and a web server, respectively). The TeX

This research paper includes open-ended questions and projects for future research. This is very much a work in progress, and no statements here are definitive. I am very interested in feedback from all participants in the TeX community, and you are invited to point out my errors, false opinions and omissions.

¹ For simplicity, the difference between free and open source software will not be dealt with here, and the term 'open' will be used.

project differs from those projects: While TeX did fulfill unmet software needs and was general-purpose software, its users' community was not necessarily technically sophisticated, and the software was not part of a computing infrastructure. It was indeed quite specialized (font design, typesetting) and what is more, had to face intense competition on all sides, from word processing software to industrial publishing software.

There are few case studies that deal with one open source software project and try to look at its functioning in economic terms. In the last few years, open source software economics has been the subject of much empirical and theoretical research. That research relied on an examination of the most well-known and successful OS projects, or on the study of limited aspects of open source software, based on some partial statistical measures like the number of contributors, lines of codes, bugs or release dates.

This case study tries to go beyond these well-trodden areas by studying a less well-known software project, which differs in many ways from those that have already been studied; it also aims at having a global vision of its history and functioning so as to generate new measures of the economic impact of open source. The conclusions from this study chal-

lenge the consensus built from previous case studies on open source software ('OSS') development. This case study goes deeper into the complexity of the internal working of the various T_EX projects, and eliminates the 'survivor' bias present in previous case studies by going into the T_EX project problems encountered along the way as much as the successes.

This case study is sponsored by GREMAQ, a CNRS research group in mathematical economics at the University of Toulouse in France, and IDEI, a research institute in industrial economics. A widely attended conference on the economics of the software and Internet industries is held in Toulouse every year, and open source software is one important research area for those two laboratories. This case study also benefited from the support of the School of Information Management Systems in Berkeley. I have worked with Jacques Crémer and Jean Tirole in France and Hal Varian in the USA, and I thank them for their advice and suggestions. I also thank the many T_EX developers, maintainers and associations members who answered my questions with unflappable kindness.

In the first part of this paper, the theoretical background to this case study is presented; in the second part, the choice of T_EX as a case study subject is motivated; and the third and main part presents some preliminary findings.

2 Research background

There are three main themes in the existing body of economic literature on open source software. Economists first tried to explain how people could collaborate freely and for free and produce in that way valuable information goods. Some principles were then expressed for the regulation of such economic activity, and finally, tools were devised to evaluate the welfare impact of OS production.

How do open source software projects work, and why do they work so? The literature on this topic builds upon the theory of incentives: the way somebody is motivated determines what he will do. Bessen (2002) defined the different categories of participants in an OSSP and their motivations. Core developers are those whose work determines the pace of the overall development, as other developers' work depends on what they do. Satellite developers are those who build upon the work of core developers to add features that are geared to special interests. Other developers make that work available to the general public by building interfaces to the program, maintaining distributions, or reporting problems with the software. There is generally an organization that coordinates the work of every developer

and defines some goals for the project. That organization usually builds around an individual, usually the initiator of the project but, with time, coordination and development tasks are shared.

The existence of OSSPs can be explained with simple economics — OS software is cheaper than proprietary ones, developers want to work on it to develop their reputation and then trade on it in the job market or to develop an expertise in software they use professionally. It can also be explained with other reasons that Richard Stallman of the Free Software Foundation was instrumental in promoting — free expression of creativity, sense of belonging in a community, ideological motivations, wanting to reciprocate the gifts in software codes made by others, etc. From a technological point of view, the birth of OSSPs may have been inevitable: as people learned how to program and could customize software to their own needs, they developed a common body of work and shared it like general knowledge.

The second theme in economic research on OSS deals with the economic principles that must inform their regulation and legal environment. It uses the theory of organization and public economics to determine how OSSP should be regulated to produce maximum welfare. There is for example an important debate over what license terms are best in what setting. License terms balance the need for control over the development of the software versus the possibility of change under the influence of others, and balance private incentives versus group incentives: Proprietary license terms give individual developers more control over their work, while GPL-type licenses reduce individual economic incentives — the economic surplus generated by software cannot be appropriated — but may generate higher overall welfare. BSD-type license terms stand in between. The legal environment also influences the level of innovativeness in software design — people may not want to contribute their best ideas to OSSPs — but a wider pool of developers who are not concerned about the acceptability of their ideas to the wider users' community may end up generating more original ideas. License terms also influence how the welfare will be distributed, as they may favor developers vs. end-users. Finally, proprietary software favors efficient coordination in a closed environment at the expense of keeping development secret to most people.

The third theme in OSS economics is the interaction between not-for-profit and commercial software. Industrial economics and game theory explain how both types of development methods compete and complete, and how commercial firms use

OSS and draw on the OS developers' community. The various strategies for making money on OSS are studied — selling CDs, manuals, developing proprietary software based on OS and using it for professional purposes or selling it to the public, selling advice to OSS users, etc. The efficiency with which both types of software are developed are compared, as well as the end-product's quality and how they compete on the software market. Because we already have tools to evaluate the welfare effects of proprietary software, the comparison between OSS and proprietary ones gives some leads for the appraisal of their welfare effects.

3 A reference point: GNU/Linux

Before presenting of some preliminary findings, the main differences between T_EX and Linux are outlined; this gives a reference point to people who learned about open source from the example of Linux. I also motivate the choice of T_EX as the subject of this case study. The T_EX project differs in many important ways from the Linux project. They were not developed in the same period, T_EX has a much longer history, and they are distributed under different license terms. The T_EX project's size, evaluated by the number of people who develop T_EX and L^AT_EX, is considerably smaller than the size of GNU/Linux project; as a matter of fact, GNU/Linux distributions generally include T_EX and L^AT_EX. Finally, the goals of the projects were different.

Donald E. Knuth developed the T_EX system in the late 70s, before the Internet came to be the tool it is today for organizing open source communities. The community surrounding the software went through many changes over many years, accompanying the evolution in the standards used for publishing and in the way software developer communities work. Linux, on the other hand, was started in the 90s, relied on the existing open source community, and used tools already developed for the GNU project and others. The T_EX project provides a long-term view of the history of an open source software project. Its relatively self-contained developer community went through several stages in its development: this study may thus help in predicting the future of other more recent open source software projects.

T_EX is a medium size software project; it is not an operating system like Linux, but is still a complete typesetting system with many interdependencies. T_EX provides a sufficient level of complexity to be the subject of a self-contained case study, but small enough to be studied as a whole. The project can be understood without relying on catch phrases

and slogans, unlike many studies of Linux.

The communities that grew up around the two systems were different. T_EX was developed by academics as part of their research programs, publishers who used it for typesetting books and journals, and developers who provided commercial versions of the software. Development of T_EX was pragmatic, funded by government research programs and universities, by its release under proprietary license terms, or from the revenues of selling CDs and manuals. Linux on the other hand drew a community that was motivated by more abstract, ideological goals — building an alternative to Microsoft — or by the programming challenge — getting to work on an operating system. Of course, the contrast should not be pushed too far; independent, 'amateur' developers who were not motivated by profit also contributed to the development of T_EX.

The license under which T_EX is distributed is essentially a BSD type license, while Linux was released under the GPL. Their license terms made a difference in the way both software developed; BSD licensed software must compete with proprietary systems based on the same source code. Because of that higher level of competitive pressure — and maybe for other reasons too — BSD projects are usually more disciplined than GPL ones; all OS development efforts bear onto the same, coherent distribution. This guarantees in principle that no development effort is wasted and that the OS software doesn't split into many incompatible projects.

The L^AT_EX Project Public License thus promoted the creation of a single common T_EX distribution; all changes to it must be distributed with the original distribution. The T_EX system was thus very stable, but it was difficult for newcomers to integrate and influence the team that decided what that distribution was going to consist of. There were times when many competing versions of the same package existed until one became dominant and a part of the standard distribution. Therefore, no one person asserted him/herself as a leader for the T_EX project; its development was the product of the competition between packages, and each package in T_EX remained under the control of one person or of a stable and limited set of developers.

Modules in Linux drew a more diverse set of contributions and there was thus the need for a leader who would coordinate and integrate contributions. D.E. Knuth implemented changes in T_EX's core (`tex.web` and the kernel) after consultation with other developers but essentially alone, as he took sole responsibility for the T_EX core. Linus Torvalds, on the other hand, had to integrate changes in

the code that were proposed by others, because anyone could take the kernel and make his own changes in it. This is how D.E. Knuth's authority was built into the system while Linus Torvalds had to assert his authority based on his charisma as a leader.

TeX was user-oriented from the beginning; it was meant to provide an interface between authors and publishers. People without any programming background were to be able to learn how to use it. This is in contrast with Linux or Apache, which were meant for people with a programming background. This difference allows one to test whether OSS can be popular beyond the programming community. While Linux versions were released very frequently, the users' orientation of TeX led its developer to release new versions of their packages only after consultations with the user base, and only after having made sure they would maintain compatibility with older versions of the software and that they did not contain serious bugs. The development of Linux was made in the open while TeX packages were mostly created in small developers' groups and released only after full completion. In both cases, though, the interface between developers and users was taken care of by the people who managed the distributions of the software—those who organize and classify others' independent work, make their code work together, and choose which packages to include in a standard installation of the software.

4 Some preliminary findings

This part shows you have to be very careful when writing the initial code of a software project, as it will influence all future development. Any choice at this stage should be carefully evaluated using the lessons from the past.

This part is organized into three main sections. The first deals with the output from the TeX OSSP—the software code. Its initial quality influenced its later development. The software's quality is evaluated by comparing it to equivalent proprietary software. The second section examines the software development process and its dynamics, and will focus on its leadership: OSSPs need independent minded leaders who begin by implementing their ideas and only then share the result with others. The third section is a study of the framework in which the development of the software took place—it is concerned with the governance and institutional design of OSSPs. TeX provides a rare example of an OSSP where users organized to influence the development of the software.

4.1 TeX code: Characteristics and quality

4.1.1 Importance of the initial code

There is a conflict between the perfection of the coding of the initial software and the ease with which it can be changed afterwards. Knuth wanted to produce compact software that would run fast and be devoid of any bugs. He thought a stable system was preferable to an evolving one. This was justified for TeX, as it was to become a system used by non-specialists. The OS development model—'release early and often'—would have led to much confusion in the user community, and to compatibility problems for those using different versions of TeX.

Knuth's code was originally organized in modules but, as it got optimized, the code became very tightly integrated. Each part became dependent on each other and the whole began to look monolithic. The language that was chosen at the beginning (Pascal) soon went out of fashion, and the software's restrictive license terms made it difficult to change, as changes couldn't gain official status.

On the other hand, while the software remained monolithic, TeX82 was a complete reworking of TeX78 that made many settings parametric instead of automatic, making powerful macros from TeX's primitives possible. This satisfied TeX developers for a long while, during which the core code remained firmly under Knuth's control.

This is why it is only quite late in TeX's development that the core's limitations became apparent, and it became necessary to make it easier to change, for example by organizing its modules into libraries. TeX's license terms are such that the name "TeX" is reserved, so that Knuth was able to freeze TeX's core. This would not have been a problem—developers always could take the TeX program and rename it—but since Knuth did not designate a successor, there was no focal point on which developers could synchronize. Developers were not able to change the core, or more to the point, couldn't initiate a group dynamic to adopt the changes they made. This would have required a long-term commitment, perfect knowledge of the program and close coordination since any change by one would affect all the others. It soon became clear it was not possible to lead such a project with people linked only through electronic means; the core of TeX had to be reworked by a devoted team so as to make it modular.

This task was taken up by the NTS team, but it took too long to deliver a finished product. When NTS was finally delivered, it was not used except for experimental purposes. This shows the importance

of getting things right in the first place; by the time the program had been rewritten, most \TeX users and developers had preferred to base their future use of \TeX on other, less ambitious, alternatives, such as pdf \TeX .

In summary, independent development of the program was delayed because \TeX evolved into a monolithic program that was intended to become a standard in publishing and was developed in a closed academic setting. While Knuth's objectives were realized, subsequent development was difficult, in a new setting where \TeX users and developers had to coordinate through electronic means and the OSS developer community was established around concepts and tools different from those of the first \TeX implementors. The \TeX program had to be translated into the now-ubiquitous C programming language, and when the rewriting of the core into a modular structure proved impossible, the efforts had to be directed towards helping \TeX users manage the \TeX legacy by making it compatible with the new typesetting standards.

On the other hand, it is not clear that \TeX could have been developed from the beginning in an OS fashion. While the core did not get changed in an OS way, the program did attract a lot of independent development, notably on \LaTeX . That \TeX 's core was not developed strictly according to OS paradigms could be evidence that OSS development methods are only appropriate when a base product has already been completed but are difficult to put into practice for the base product. It is also possible this was due to the nonexistence of an organized OSS developer community at the time.

4.1.2 Impact of \TeX on innovation and welfare, and evaluations of quality

In this section, the quality of the software from various points of views — users, developers, and computer science researcher — is compared with competing proprietary software.

There was no software even remotely up to the standards of \TeX when it was developed.² The general program used at that time for typesetting was called ROFF, a text formatting language/interpreter associated with Unix, and for a long time there was some competition between the partisans of ROFF and \TeX . The main competing software for the casual user is now Microsoft Word. Even though Word is WYSIWYG while \TeX is not, and the audience is therefore very different, the two compete because

² With the exception of a couple of very expensive, proprietary systems, e.g., Penta. *Ed.*

\TeX saw itself as a potential standard for document exchange. The main competing software for typesetting of complex mathematical documents in the publishing industry is 3B2. Adobe Framemaker and QuarkXPress are also popular alternatives.

A frequently asked question is whether OSS replaces proprietary software and whether it undermines innovation by imitating proprietary companies. In the case of \TeX , it is quite clear which way the inspiration went. Some aspects of \TeX were imitated, for example the equation editor in MS's Word and \TeX 's hyphenation and justification algorithm in Adobe's InDesign. Other commercial software eased the use of \TeX by adding a graphical user interface and porting it to other platforms — this is the case of Personal \TeX 's PCT \TeX , the first IBM PC-based \TeX system, or of MacKichan Software's Scientific WorkPlace which integrates \TeX and Maple. It is \TeX which inspired commercial development much more than the reverse.

It is also not clear whether commercial and OS products complement or substitute for one another. There are examples of dual use, some typesetting firms using \TeX internally and delivering the finished product with 3B2. There are also examples of users and firms switching back and forth between OS and proprietary software. \TeX Live, for example, did take some business from commercial implementations, especially since it is easier to maintain using Linux based network management software. The competition is very rarely frontal, and few \TeX projects see themselves as ideologically opposed to commercial software. \TeX did take the place of other commercial software though, but while it replaced obsolete proprietary typesetting software at the AMS, it also inspired other proprietary software (principal concepts, line breaking algorithm, syntax) and it paved the way for getting typesetting software in the hand of the users instead of that of the typesetter. It initiated a new workflow in publishing. Additionally, some of the first people to use \TeX did not see commercial software as an alternative and \TeX was a way for them to obtain functionality not present in (affordable) commercial software.

Finally, the development of \TeX was encouraged by potentially competing commercial software. Hàn Thế Thành received a scholarship from Adobe to develop the pdf \TeX program; this was in the interest of Adobe as it wanted to gain more general acceptance for its software and was also a way to encourage exchanges with the OS community. The competition between OS and proprietary software is based on subtle mechanisms that are deserving of further study.

While a comparison of the welfare generated by T_EX with that which would have been generated by a proprietary program may appear to be a futile academic exercise, T_EX was developed as an alternative to commercial software that was used by the AMS in its publishing section, and the AMS did ponder what was the best option: wait for commercial software to be released that would fit their need, or give the impulse to a new, open source, software. The comparison between T_EX and a hypothetical equivalent commercial software can be made in terms of innovation, responsiveness to users' needs, pace of development, capacity to integrate into existing systems and the efficiency with which the software is developed.

Proprietary software is sometimes out of touch with users, as developers are not users. But in OS, developers are sophisticated users, which means the software may not be at the reach of the average user. As far as OSS is a tool for average users whose needs are not fulfilled by proprietary firms, then its development may be as misdirected as that of closed-source software, although in other ways. However, the development of T_EX and L^AT_EX was made after consultation with professionals from the publishing industry and meetings with the AMS — the first sponsor and user of T_EX. In the summer of '79, Barbara Beeton of the AMS and Michael Spivak — both of whom went on to important positions in the TUG organization and others — spent time in Stanford developing T_EX macros to test T_EX capabilities for such AMS requirements as generating indexes. Their work led to a series of suggestions for improvements, and to the AMS giving its backing to the project.

The L^AT_EX3 project members also consulted with the AMS and various T_EX user groups, publishers such as Addison-Wesley and Elsevier, and got support from companies, some that sold T_EX-based software — Blue Sky Research, TCI Research, PCT_EX — but also Digital Equipment Corporation, Electronic Data Systems, etc. David Rhead gathered the wishes of users from email discussion on the L^AT_EX discussion list. Those wishes were mainly about the page layout specifications and the user-interface design, things that are of primary concern to users and not so much to developers. This close collaboration with professionals in the typesetting and publishing industry, which can be illustrated in many other examples, goes against the view that OSS that is too geared towards specialist use will not be successful (Schmidt and Porter [2001]).

It is often said that the pace of improvements is quicker in OSSPs. Improvements in proprietary soft-

ware are not released frequently, since there is a cost to doing so, and their owners want the improvement to be valuable enough for existing users to buy it. But with OSS, it is difficult to coordinate the user community on the most recent improvement; this is a problem as the software is used for collaborative work, and people want stability. In the case of T_EX, the solution was to design standards for the classification of packages and requiring new packages to be distributed with older, approved ones so as to guarantee the availability of a complete working set of packages to users.

Standards are difficult for proprietary software firms to adhere to because they want to protect their user base — prevent it from switching — and also because the source is closed, so that it is difficult to create applications linked to it. However, software firms that propose development platforms to programmers are also interested in the promotion of their platform, and usually are able to establish and maintain them as a standard. OS projects on the other hand generally find it difficult to coordinate on a standard. While this may not be a problem because OS is usually platform independent, it is difficult to keep code operational when there are constant changes to the underlying operating system and compiler platform (Torzynski [1996]).

While the sharing of information may be done less efficiently in OS projects than in proprietary firms, the pool of information that can be shared is expanded. Many contributors to T_EX would probably never have worked in a commercial firm, and even when they were hired in commercial firms, such as Elsevier, they continued contributing their improvements to the wider community. Overall, if it is possible to prove that OS developers would not be able to do what they do in a closed environment and that what they do would not be done by proprietary software, then OSS is beneficial. As an OSSP develops however, it can grow to come into competition with closed source: there is competition at the fringe, when users could use both.

The situation is complicated by the fact that some proprietary software may be based on OS and compete with pure proprietary software. Some work would need to be done to compare publishing firms that use OS software (Hans Hagen's Pragma ADE in the Netherlands, B. Mahesh's Devi Information Systems in India) versus firms that use proprietary software. There is a difference in the nature of upfront cost, maintenance efforts, level of support, possibility of improvements, capabilities, etc.

4.2 The process

4.2.1 Dynamics of the project

Various forces direct the development of \TeX , due to the different concerns, objectives and priorities of its developers who come from different fields and make use of \TeX in different ways. Over time, with the user base changing and the software's environment evolving, different types of priorities have emerged: in the first period, Knuth's own objective was to develop software that could do mathematical typesetting by computer that was worthy of the best manual typesetting tradition. Then, when he felt his objective was achieved, the AMS wanted to make this instrument available to the wider mathematical community, and sponsored the development of \AMS-TeX and its subsequent merging with \LaTeX .

Later on, the objective of the subsequent core developer teams was to make use of new computer capacities and make \TeX more easily extendable with the use of new programming tools (Ω , NTS), while also establishing a standard \LaTeX to prevent forking — $\text{\LaTeX} 2_{\epsilon}$ by the $\text{\LaTeX} 3$ team. At the same time, some work was necessary in making \TeX able to produce not only pdf and html documents, but also Framemaker and Word documents. The work on making \TeX compatible with proprietary standards was first done by commercial companies. Among the priorities, keeping up with competitors' functionality, such as Adobe or WordPerfect, does not seem to have been important, as \TeX developers advocated the use of free source fonts instead of commercial fonts, and mark-up-based document writing instead of Word-like WYSIWYG programs. There are, however, some open source projects trying to achieve greater user friendliness — David Kastrup's preview- \LaTeX package to ease editing, LyX, a document processor using \LaTeX in the background, GNU $\text{\TeX}Macs$, inspired by \TeX and GNU Emacs, etc.

Competition between different development philosophies also worked to determine what works and what doesn't and which way the overall project had to go. An illustration is the difference in philosophy between the NTS project and \pdfTeX : the NTS team wanted to keep compatibility with the initial version of \TeX , while totally rewriting the code — rewrite the WEB Pascal program into the Java programming language. \pdfTeX , on the other hand, was based on the C implementation, less generalized in scope, but easier to work on (Taylor [1998], Hàn Th   Thành [1998]).

4.2.2 Limits of the project

Does the OS development process or the specific OS institutions that support development put limits on the growth and success of open source software?

Growth and success are important because even if the software functions in accordance with the stated aims of the project initiator and the initial user community, it will quickly become obsolete and useless, even to those same people, if it is not maintained to keep up to date with the changing software environment. This can justify changing the aims of the software's community, even in ways not to the advantage of the project initiators, if that can make the software more attractive to new developers.

The pace of development slowed over time. The graph represents the number of bugs found by Knuth in the core program through time. After $\text{\TeX}82$ was released, Knuth stopped implementing general user requests, except for allowing 8-bit input in 1989. Since the whole \TeX system refers back to the core of \TeX , its pace of development is indicative of what is happening in the wider \TeX community.

There is however a difference between development and diffusion. As the software's main tree development is blocked, it can still be adapted to new platforms, translated, and people trained to use it. Nonetheless, it is still true that it will be more difficult to diffuse if there is nobody ready to make the necessary tinkering in the software code to permit adaptation to new usage.

The diffusion of \TeX can be evaluated by looking at the number of requests for support in \TeX -related newsgroups, the number of TUG members, and the number of academic papers written with \TeX . While postings to the English-speaking newsgroup reached a plateau — probably because most questions were already answered in English and referenced in FAQs! — newsgroups in other languages attest to the vitality of the international growth of the user base.

There are some technical limits to the development of an OSSP, and those are different from those that limit the growth of proprietary software. Those limits are due to coordination problems in the development and support. Initial choices in the software programming are hard to change because that requires more coordinated effort over a longer period of time than most OSSPs are able to provide. This means a project can get stuck with outdated standards. There is also difficulty in keeping the original programmers to remain committed to the project.

There are only a limited number of people who may use the software, even if it tries to broaden

Cumulative number of bugs in TeX by day of discovery.

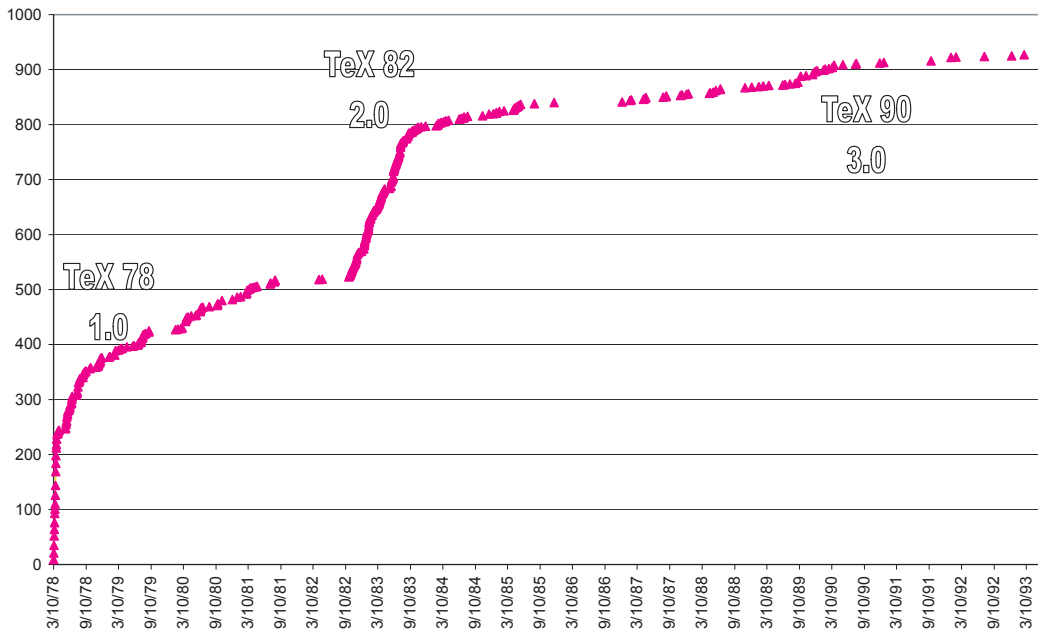


Figure 1: The bugs of TeX.

Number of postings to TeX newsgroups by months.

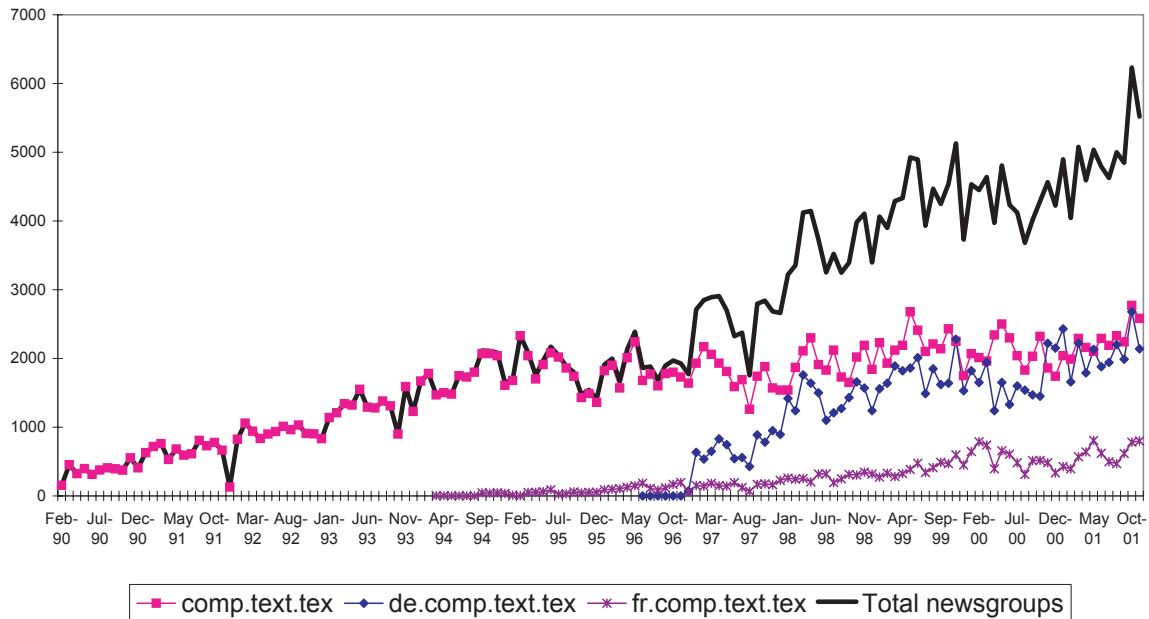


Figure 2: Monthly postings to TeX-related newsgroups.

its appeal. The software progressively reaches all of its intended audience, or is supplanted by other software for that audience.

Some aspects of the software's basic concept are difficult to change, for example its typesetting mark-up language, and this put limitations to its appeal. The concept becomes fatally obsolete, even if it made sense when it was first thought up. Here, other mark-up languages appeared (MathML), and other typesetting engines did not necessitate as much learning — Microsoft Word is less powerful than \TeX but has a more gradual learning curve — or were more tightly integrated with new standards and necessary capabilities — Adobe's InDesign to produce pdf files or to use Quark XPress document files. The Con \TeX t and pdf \TeX projects were attempts to broaden the capabilities of \TeX to keep them up to date with what was necessary for online publishing. \TeX was oriented toward printing, and was not able to easily provide the kind of interactive color complex documents with figures needed for online publishing. It is also based on a document exchange standard (DVI) that has never achieved the popularity of the pdf format. The pdf \TeX and Con \TeX t projects thus had to make changes in \TeX 's conception to adapt it to new needs. Other projects made \TeX XML-compatible.

In summary, \TeX was at first in the forefront of mathematical publishing, but then had to adapt and borrow concepts from new and popular software projects — and this process met with some resistance. The number of people who were interested in those improvements was limited to a fringe, and they found it difficult to advertise their projects beyond the people who already were using \TeX .

Finally, the OS organization imposes some limits: the originator is ready to support only a limited number of people; Knuth had other priorities, the writing of his monumental series *The Art of Computer Programming*. \TeX was in fact originally meant only to typeset those books.

Limited explicit mechanisms (interface specifications, processes, plans, staffing profiles, reviews), extensive reliance on implicit mechanisms (personal relations, customs and habits) and on one-to-one interactions in small teams (communications only mechanism), mean that the development process did not scale easily. Choosing an OS development process put limits on some areas of the software's development.

4.2.3 Leadership

There is a need for a leader in an OSSP. The production of an OSS cannot be described as being peer-

based. Patterns in the history of the projects related to \TeX provide lessons on what constitutes good leadership in an OSSP because they provide a broad sample extended through time. The reason for the projects' successes and failures, which can only be determined through time, can thus be analyzed. The most effective type of leadership seems to consist in first developing independently some implementation of an original idea and only releasing it into the public when it is already well advanced. Projects that began by announcing their goals without backing their ideas with some implementation generally failed because other developers contested their technological decisions or couldn't contribute. There is therefore a limit to the power of consensus building and cooperative development; it is frequently better to go it alone and then ask for help once the project is well along.

Knuth's leadership was characterized by a heavy involvement in the beginning and the choice to leave later development to others. That leadership style was very successful for the beginning of the software's development, but the desire to preserve some stability in the program produces the danger of impeding its development. This could have led to forking if the community built around \TeX had not been so cohesive.

Although there is a need for a leader, there also are problems in coordinating on one leader. An example of a successful leader was Hàn Th   Thành, who initiated the pdf \TeX project to directly output pdf files from \TeX . This is seen as a successful project because Thành released his work only after having done the preliminary groundwork, and was then able to let other developers take the initiative in applying and enhancing his work. The Omega project encountered problems because, while it communicated its goals early, and implemented innovative ideas to enhance the multi-lingual capabilities of \TeX , it did not at first attract developers beyond the initiators and had problems convincing the \TeX community it would one day become fully implemented. That project was first presented in 1995 and it is only now that it is gaining momentum and being supported by the \TeX community.

A large part of the difference between those two projects is often attributed to the leadership style of their initiators; the fact that the Omega developers did not deliver on their claims rapidly made the established leaders in the \TeX community doubt that project was worth getting involved in. As we will see below, however, the main difference between the two projects was perhaps not the difference in the way they were led and in the ability of its program-

mers, but in the acceptability of the project to the existing developers and users. The pdfTeX project did encounter some resistance at its beginning from people who thought other ways to generate pdf files from TeX input were preferable, but the importance of the ultimate goal was not in question. This was not the case of the Omega project.

In short, a project leader will be seen as a good leader depending on whether he is allowed to work within the existing system. If he is not accepted and does not get the support of other developers, then his project may end up badly in a self-fulfilling prophecy. A project will find it difficult to thrive if it doesn't get the support of the establishment, so that most successful projects will serve the needs of existing users and developers, and not those of potential newcomers.

The most consistent leadership was given by organizations: the AMS, which ensured that TeX served the mathematical community, and TUG, which ensured it was user-friendly. The AMS was the main leader of TeX's development. It provided financing for a user group, made propositions to developers, and gathered them to establish objectives. The involvement of the AMS was thought out well in advance, as they wanted to become involved in helping to develop a document preparation system, instead of waiting for a commercial system to be provided to them. They needed a system that was compatible with most hardware, simple, flexible, and cheap; it was to run on mainstream computers. The AMS, as well as other TeX sponsors, were conscious that there was competition between the various possible uses of the software. Because of limited human resources, the product's development could not be led in the way each constituency would like.

There was therefore a need to define an allocation process for development resources. This is how the AMS decided to sponsor the development of a modern system based on TeX that would fit its own use (AMS-TeX), and also sponsored TUG. Its role was to form a group of people who would be able to use the tools recommended by the AMS. The AMS couldn't hope that academics would use the TeX system if it didn't also provide them with the means and training to do so. This sponsoring by the AMS had an impact on the development of TeX that went far beyond the means involved because it served as a signal to other sponsors that TeX was a valuable project that would ultimately, willy-nilly, be a complete system. This is why TUG also attracted sponsorship from various hardware companies and universities. The AMS progressively

lowered its financial contribution to TUG and it is interesting to note that TUG's revenues and membership declined after the release of the final version of TeX in 1990. It would be interesting to look further into the impact AMS's support had on TUG, and the influence TUG had on TeX's development.

Leadership was also provided by users and mediated by user group organizations. This part exposes the many initiatives coming from the TeX users. The identification that is often made between users and developers is not correct, as even users who are not developers have an impact on the development. An OS project is not led through a competition mechanism where the best project wins, it is led by the user who is ready to devote time and effort to the project toward a defined goal, in this case the goal of an association distinct from the TeX community. TeX had no purpose of its own, but TUG did have a mission statement: To be a meeting place for users and developers, of course, but also to use that central position to serve the aims of its sponsors. TUG served as a meeting point between users and developers of TeX, the two not being exclusive. Articles in *TUGboat* were often written by users explaining the use they made of TeX in their respective fields, and outlining the problems they encountered. The various uses of the program led to various pulls (queries to developers) and pushes (independent development) on the program. The problem was then to harness those, integrate interesting contributions into the main distribution, and not have dead-ends. This was done by encouraging independent developers to work with the core developers so as to attain compatibility with the existing TeX system.

Users wanted to protect their investment in the software. Given the weight of legacy, there was reluctance on the part of the user community when faced with changes in the TeX system. Indeed, many of them had written their own modifications of TeX to fit their own use, and were not willing to abandon those in favor of a new system that would provide only minor improvements to them; multilingual typesetting for example was of no use to most users but requires extensive changes. There was no need for change for most users, as any change they needed could be done with macros—even though the TeX macro language itself led to very complicated and badly structured programs. It was very difficult to attract older users on an alternative, and indeed, the main hope for some developers who pursued changes to TeX came from new users in non-Latin countries—the Omega project. New categories of users who did not have the same legacy issues succeeded in breaking the status quo. Various user con-

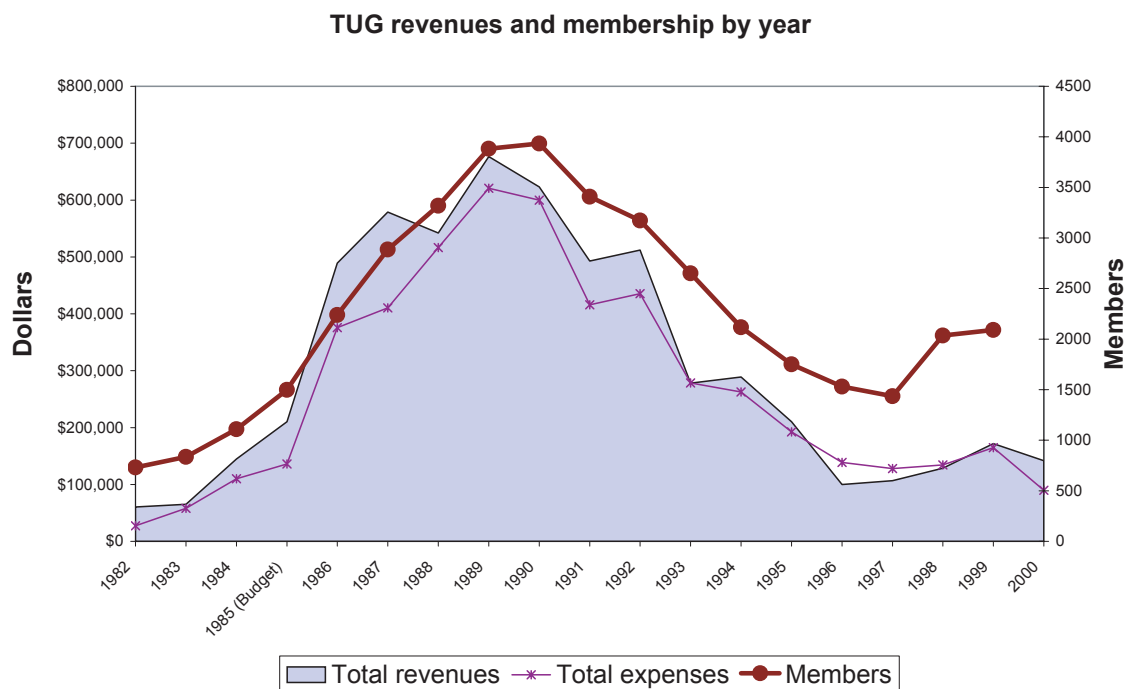


Figure 3: TUG’s revenues and membership.

stituencies, or categories of users, had different requirements: The Europeans came to adopt \LaTeX , while the Americans used \TeX , because \LaTeX had not reached a sufficient stage of development to be interesting to them at the time they adopted the \TeX system. The American organization had been built under the lead of typesetters, publishers, software companies and university institutions who were interested in mathematical publishing, while the European groups were led by individual users, sometimes active in universities or educational publishing. Many of those users’ needs were not satisfied by the user group based in the US. This is how many initiatives were made in some European LUG before being accepted by the main organization.

For example, while the US organization had an established \TeX tape distribution system, and while American developers knew each other well enough to coordinate development on a one-to-one basis or in conventions, the European users saw the need for a \TeX code central repository. This was realized by a group of volunteers at Aston University in the UK, which made it possible for users to download the latest developments in the \TeX system. This group inspired the development of a package classification system, the \TeX Directory Structure, which served as the model for \TeX distribution everywhere; this common system facilitated the installation of the

\TeX system. That initiative led to the creation of the CTAN archives in 1990, which drew contributions from the American organization too.

Another example of user-led initiatives is the initiative by the Netherlands LUG to develop and distribute 4All \TeX in 1993, a \TeX distribution on a CD that was intended to be of use to an end user with no programming background. While the Netherlands LUG could not possibly have taken up the task of delivering a complete user-friendly distribution, it gave the impetus to a wider concerted effort, the \TeX Live project. \TeX Live adopted many of the ideas in the 4All \TeX distribution — choice of programs, organization of packages, and more.

The tensions between \TeX constituencies translate not only in user initiatives, but also in different objectives for groups of developers. There was a conflict between pursuing a standardization of \LaTeX that would fit most users’ needs and allow easy interchange of documents between all users, and going forward without an overriding concern for compatibility, to serve the needs of larger classes of users.

4.3 The framework: The \TeX rules and culture, and how they evolved

The \TeX rules and culture differ from those of other communities, first due to the license terms, but also because Knuth gave authority to some lieutenants

to act on his behalf. Initiators, like Knuth or Lamport, effectively blocked development until pressure built up to a convincing expression of collective choice, when for example users and developers came to Knuth with requirements for \TeX 82 or \TeX 90. That strategy did succeed in reducing the workload on the initiators and in reducing unnecessary developments, but it did not encourage independent development.

The development of the software became more and more decentralized because of two effects. The first one is that the program became more complex, and the number of specialized programs linked to it rose — programs to make figures, indexes, nationalizations of the software, programs to translate \TeX input into outputs other than DVI, various interfaces, various rewriting of the software into other programming languages, etc. The second one is that technology made it possible to coordinate projects one-to-one through electronic means, instead of all contributions going through a central body which then reflected it to all after having filtered the noise (errors, unwanted developments, etc.).

For example, the way submission of changes to the \TeX software were made evolved through time. There is a contrast between new systems and older ones. For \TeX , bugs were submitted to Knuth via filters, people who had to make sure the submission was valid. Changes in the software were suggested to Knuth, who then determined how those changes were to be embodied in the software. In the \LaTeX 3 project team, developers exchanged code via private e-mail and met person to person to discuss changes. Of the newer project, some like Mik \TeX were listed on Sourceforge, and used all the tools now available to coordinate OS projects — CVS files, central repository — and many, like preview- \LaTeX , accepted contributions by totally unrelated volunteers.

One important part of the \TeX community's culture was that it tried to develop user-friendly software based on OSS. This makes \TeX a different case than other well-known OSSPs that have been studied before (Apache, Linux), and will help to determine if OSS can be user-oriented, a mass-market product. From the beginning on, the program was intended to be used by non-programmers: secretaries, researchers, . . . In fact, one of its main 'selling' points at the beginning was how easy it was to install and learn; the syntax was meant to be natural, and Knuth wrote a complete manual for the program at the same time as he developed it. Even the programming language was meant to make the coding easy to understand; the coding was documented along the way using a language and a method, lit-

erate programming, developed by Knuth.

Also, \TeX was developed with non-programming concepts in mind, i.e. concepts that were of no use for the greater programmers' community. Indeed, \TeX was meant to be a translation of the best typesetting practices into a programming system. This served the needs of the typesetters, the publishers, the academics, but not those of the typical programmer who is not involved in typesetting. \TeX came from the lucky coincidence of one person having the need for better typesetting and the ability to follow up on that need.

The problems that have been classically invoked to say that OS programming could not produce user-friendly, mass-market programs are that it cannot generate a good user interface, that a users' orientation requires a different turn of mind than that of OSS developers, that it is not possible to coordinate developers efficiently enough in an OSSP so as to release a fully functional pre-packaged product, and finally, that OSSP developers do not have the means or the will to communicate with end-users.

A good user interface requires a lot of work and is time consuming but is not needed by somebody who already is able to use the software. Ulterior (profit) motives must come into play: this is where commercial organizations have a role, at least until the OS organization is strong enough to be able to produce an easy to install and use program. This is what happened in the \TeX community, where commercial implementations of \TeX appeared in the mid-80s and provided the only user-friendly alternative during at least 10 years. Mik \TeX was the most popular open source user-friendly interface to \TeX , but worked with Windows only, and it is only with the release of the first \TeX Live distributions that a complete easy to install \TeX distribution was made available for Linux.

There were also individual OS initiatives to make the software easier to use; preview- \LaTeX and \TeX view provided graphical interfaces. Other collective initiatives were driven by ideology; LyX is such an attempt at providing a convenient \TeX interface. All those encountered difficulties in providing an interface that would work with all different possible and ever-changing installations of \TeX . They were surprisingly loosely coordinated with the core group of \TeX developers, maybe because the development of those interfaces cannot possibly take account of all the changes in the core program, so that they were based on older versions; their development thus didn't necessitate close coordination with leading core developers.

A stable user-level platform was needed because

TeX documents had to be easy to share for it to be considered a standard and adopted widely. Therefore a core product had to be defined, to which other things were added and with which they had to be compatible. This is where a central authority, and defining limitations in what will be supported or not, came into play. The L^ATeX3 team played that role by defining a ‘core’ L^ATeX package, defined as what the core team thought it had the time to maintain. The TeX Live distribution helped in focusing energies by defining what would be distributed more widely to the end-users, thus guaranteeing to developers that their work would have some impact in the user community.

5 Conclusion: Work to be done

While many participants in the TeX community were interviewed, and a large documentation on the TeX system and the history of its development was gathered, that knowledge still has to be organized in a systematic way. Some salient features will be further developed; for example, a more careful analysis of the determinants in the success of an OSS sub-project could be made. This would allow a better definition of quality, leadership and support in an open source environment.

6 Bibliography

6.1 (L^A)TeX related articles

Bodenheimer B. [1996] “Questions souvent posées sur (L^A)TeX”, Cahiers GUTenberg, 23 April 1996.

Clark M. [1989] “Olde Worlde TeX”, TUGboat 10(4), 1989 Conference Proceedings.

Gaudoul A. [2003] “The (L^A)TeX project: A case study of open source software”, Working Paper, January 2003.

Hàn Thế Thành [1998] “The pdfTeX program”, Cahiers Gutenberg, 28–29 March 1998.

Knuth D.E. [1989] “Notes on the Errors of TeX”, TUGboat 10(4), 1989 Conference Proceedings.

Knuth D.E. [1989] “The Errors of TeX”, Literate Programming, CSLI Lecture Notes, no. 27, 1992.

Knuth D.E. [1989] “The new versions of TeX and Metafont”, TUGboat 10(3), October 1989.

Knuth D.E. [1991] “The future of TeX and Metafont”, TUGboat 11(4), January 1991.

Knuth D.E. [1998] “The Final Errors of TeX”, Digital Typography, CSLI Lecture Notes, no. 78, 1999.

Lammarsch J. [1999] “The History of NTS”, EuroTeX ’99 Proceedings.

L^ATeX3 Project Team [1997] “Modifying L^ATeX”, TUGboat 18(2), June 1997.

Mittelbach F. and C. Rowley [1997] “The L^ATeX3 project”, TUGboat 18(3), 1997 Conference Proceedings.

Skoupy K. [1998] “NTS: A New Typesetting System”, TUGboat 19(3), 1998 Conference Proceedings.

Taylor P. [1996] “A brief history of TeX” in “Computer Typesetting or Electronic Publishing? New trends in scientific publications”, TUGboat 17(4), October 1996.

Taylor P. [1997] “Présentation du projet ε-TeX”, Cahiers Gutenberg, 26 May 1997.

Torzynski M. [1996] “Histoire de TeX sous DOS et Windows à l’ENSP de Strasbourg”, Cahiers Gutenberg, 25 November 1996.

Advogato interview with Donald E. Knuth [2000] <http://www.advogato.org/article/28.html>.

Interview with Leslie Lamport [2000] “How L^ATeX changed the face of Mathematics”, DMV-Mitteilungen, January 2000.

L^ATeX Project Public License at <http://www.latex-project.org/lppl.html>.

The TeX Users Group (TUG) at <http://tug.org>.

The L^ATeX3 project at <http://www.latex-project.org>.

NTG TeX future working group [1998] “TeX in 2003”, TUGboat, 19(3), 1998 Conference Proceedings.

6.2 General articles on open source

Anderson R. [2002] “Security in Open versus Closed Systems—The Dance of Boltzmann, Coase and Moore”, Working Paper.

Behlendorf B. [1999] “Open Source as a Business Strategy”, Open Sources, O’Reilly editors

Benkler Y. [2001] “Coase’s penguin, or, Linux and the Nature of the Firm”, Yale Law Journal, 112, October 2001

Bessen J. [2002] ‘OSS: free provision of a complex public good’, <http://www.researchoninnovation.org/>.

Bezroukov N. [1999] “Open Source Software Development as a Special Type of Academic Research (Critique of Vulgar Raymondism)”, First Monday, 4(10), October 1999.

Brady R., R. Anderson and R. Ball [1999] “Murphy’s law, the fitness of evolving species, and the limits of software reliability”, Cambridge

University Computer Laboratory Technical Report no 471, September 1999.

Crowston K. and B. Scozzi [2002] “Exploring the strengths and Limits of OSS Engineering Processes: A Research Agenda”, Second Workshop on Open Source Software Engineering, 24th International Conference on Software Engineering, Orlando, USA, May 25, 2002.

Dalle J.-M. and N. Jullien [2001] “Open Source vs. Proprietary Software”, Working Paper, October 2001.

Dalle J.-M. and N. Jullien [2002] ‘OS vs. proprietary software’, <http://opensource.mit.edu/>.

Dalle J.-M., P. David and W. Steinmueller [2002] “An agenda for integrated research on the economic organization and efficiency of OS/FS production”, Working Paper, October 2002.

Gaudeul A. [2003] “Open Source Software Development Patterns and License Terms”, Working Paper, February 2003.

Ghosh R. and V. Prakash [2000] “The Orbiten Free Software Survey” at <http://orbiten.org>.

Halloran T.J. and W.L. Scherlis [2002] “High Quality and Open Source Software Practices”, Position Paper, Second Workshop on Open Source Software Engineering, 24th International Conference on Software Engineering, Orlando, USA, May 19, 2002.

Hann I.-H., J. Roberts, S. Slaughter and R. Fielding [2002] “Delayed Returns to Open Source Participation: An Empirical Analysis of the Apache HTTP Server Project”.

Hertel, G., S. Niedner and S. Herrmann [2002] “Motivation of software developers in open source projects: An Internet-based survey of contributors to the Linux kernel”. *Research Policy*, July 2003, vol. 32, iss. 7, pp. 1159–1177(19).

Hippel E. [2002] “Open Source Software as horizontal innovation networks — by and for users”, MIT Sloan School of Management WP No. 4366-02.

Johnson J.P. [2000] “Some Economics of Open Source Software”, <http://opensource.mit.edu/>, December 2000.

Kuan J. [2002] “Open Source Software as Lead User’s Make or Buy Decision: A Study of Open and Closed Source Quality”, 2002 OSS Conference, Toulouse, <http://www.idei.asso.fr/>.

Kuwabara K. [2000] “Linux: A Bazaar at the Edge of Chaos”, *First Monday*, 5(3), March 2000.

Lakhani K. and E. von Hippel [2000] “How Open Source software works: ‘Free’ user-to-user assistance”, MIT Sloan School of Management Working Paper 4117, May 2000.

Lerner J. and J. Tirole [2000] “The Simple Economics of Open Source”, NBER Working Paper 7600.

Lerner J. and J. Tirole [2002] “The Scope of Open Source Licensing”, Draft, 2002.

Mockus A., R. Fielding and J. Herbsleb [2000] “A case study of open source software: The Apache Server”, International Conference on Software Engineering, pp. 263–272, 2000.

Mockus A., R. Fielding and J. Herbsleb [2002] “Two case studies of open source software development: Apache and Mozilla”, Working Paper.

Mustonen M. [2002] “Why do firms support the development of substitute copyleft programs?”, Working Paper, October 2002.

Mustonen M. [2002] “Copyleft—the economics of Linux and other open source software”, Working Paper.

Nakakoji K. et al. [2001] “Toward Taxonomy of Open Source: A Case Study on Four Different Types of Open Source Software Development Projects”.

Peyrache E., J. Crémer and J. Tirole [2001] “Some reflections on Open Source Software”.

Pressman R. [1997] “Software engineering”, 4th edition, Mc-Graw-Hill editors.

Schmidt D.C. and A. Porter [2001] “Leveraging Open Source Communities to Improve the Quality & Performance of Open Source Software”, Position Paper, First Workshop on Open Source Software Engineering, 23rd International Conference on Software Engineering, Toronto, Canada, May 15, 2001.

Scotchmer S. and P. Samuelson [2002] “The Law and Economics of Reverse Engineering”, *Yale Law Journal*, April 2002.

Silverman D. [1999] “Doing Qualitative Research: A Practical Handbook”, Sage.

Varian H. [1993] “Economic Incentives in Software Design”, *Computational Economics*, 6(3–4) pp. 201–17, 1993.

OSS conferences in Toulouse [2001, 2002, 2003], <http://www.idei.asso.fr/Commun/Conferences/Internet/>

GNU General Public License at <http://www.gnu.org/copyleft/gpl.html>

A Polished T_EX story*

Andrzej Odyniec
MacroSoft S.A.
ul. Chrościckiego 49
02-414 Warszawa
Poland
anody@macrosoft.pl

Abstract

Years ago, the birth of T_EX put us, Poles, on the horns of a dilemma: how do we reconcile T_EX's beauty with our attachment to the peculiarities of the Polish language—its multifaceted inflection, a plethora of diacritics and last but not least the prevailing typographical rules?

Whatever the thinking was, enough of us became determined to make The Lion at home in Poland.

This paper presents the story and the people of the 25 adventurous years of the Polish T_EX polishing to not only our but also—we hope—many of our European friends' benefit.



In the beginning

Once upon a time there was a lovely Princess. She lived in the highest room of the tallest tower in a castle guarded by a terrible fire breathing Dragon. Many a brave knight tried—oh no, this should be a different story!

Not so long ago there was Professor Janusz Bień who was the first to typeset a Polish text using T_EX. The text looked more or less like this:¹

* Translated into English by Jerzy B. Ludwichowski, Nicolaus Copernicus University, ul. Gagarina 7, 87-100 Toruń, Poland, Jerzy.Ludwichowski@uni.torun.pl. Drawings by my son, Jędrzej Odyniec.

¹ Written by Julian Tuwim, a famous Polish poet. (Dziela, Tom III, *Jarmark rymów*, Czytelnik 1958, s. 343: *Nowe a skuteczne rymy*; footnote on p. 643: First printed in a section edited by Tuwim *Cicer cum caule, czyli Groch z kapustą*, of "Problemy" 1949, nr 10.) Originally it was typeset in plain T_EX and luckily contained only one eogonek—I don't know how Professor Bień went about it then. There is an interesting story the author gave with this poem: *It happened that we fell in love with a Polytechnic student, a beautiful albeit unfortunately a very serious girl. We started flooding her with poems. To no avail. She mocked our heart—card, vain—plain, tears—hers rhymes. We then came up with an idea which opened the heart of our physicist, mathematician and future engineer. Our poem was...*

Dlaczego sobie Pani ze mnie kpi,
Cierpieniom moim niech nadejdzie kres,
Siła mojej miłości równa się π
Pomnożone przez

$$\sqrt{\frac{2(P+Q)(L^2+a^2)+Gy^2}{g[2(P+Q)a+Cs]}}$$

which translates² to:

Oh, You do deride me, why
Let my sufferings go away
The power of my love is equal to π
Multiplied by

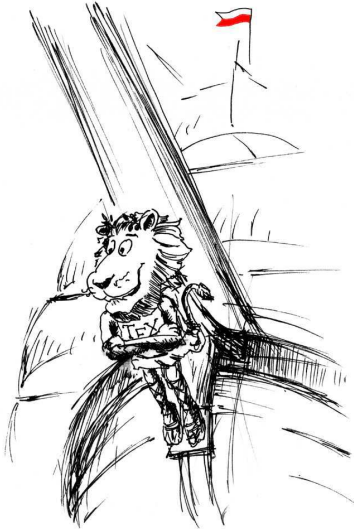
$$\sqrt{\frac{2(P+Q)(L^2+a^2)+Gy^2}{g[2(P+Q)a+Cs]}}$$

There is some charm in this poem. Did Julian Tuwim really express his love through this formula? Was his love bigger than the love of T_EX many a Pole has devoted their professional life to? Many of

² By Andrzej Odyniec and Jerzy B. Ludwichowski.

us still think that our encounter with T_EX has a romantic note to it, or perhaps a sense of an adventure or a fairy tale.³

Pioneers' time When thinking of old, pioneering times, of our adventures with great computers, great programmes and great people I cannot forfeit the impression that similar things happened before.



When I close my eyes, I find myself in the times of great sailors and imagine a big brigantine with sails full of wind. When looking closer, I see a big Lion on a galleon under the bowsprit and the crew on the deck lovingly scrubbing it to shine as *polished*. When I look up, I see the Polish white-red flag flying on the main mast. That is why my story will again and again recall this picture.

The first expedition In the old times one had to use “square” monies⁴ to pay for computing time. To get computing time officially one would need to obtain scientific approval and be allocated those — huge for a private pocket — square sums. Unfortunately, Janusz Bień was unable to spark sufficient interest in the Polish science community. And the times were difficult — martial law ruled. Therefore

³ My eldest son is almost the same age as T_EX. Perhaps some day he will become an architect but even now he remembers the day when I took him to the computing center to show a computer doing calculations on a separate story of a huge building guarded by a fire breathing Dragon, oops, no, not again. A few years ago he recognized the same computer in the Museum of Technology. He still can talk as endlessly about this day as he can about old sailing ships.

⁴ The relation of “square” monies to normal, “round” money was such that “square” money did not exist in a material or visible form. It was only transferred between state companies and the right to manage it was issued to individuals as a kind of recognition or favor from the Communist rule. To be suspected of “improper management” of such monies usually meant an end to one’s career or sometimes even freedom.

the knowledge of T_EX could only be extracted from Stanford University Computer Science Department reports and the *TUGboat* bulletins.

Jan Madey was the pioneer of an overseas expedition and thus the first skipper of the ship with the Polish banner and The Lion at the galleon.



Recently Professor Madey made the headlines as the coach of the winning team of Warsaw University in the ACM International Collegiate Programming Contest 2003. The picture shows him and his team receiving laurels during the Beverly Hills, California, finals (photo by David Hill).

It was he who invested his own money and in 1983 brought to Poland the first tape with T_EX version 0.8.⁵

Back home This of course was not the end of the story but rather its beginning. Heavy fights on almost all fronts had to be won: porting of the Pascal compiler written at the Institute of Computer Science Foundations of the Polish Academy of Sciences to the VM operating system of the IBM 370/148 mainframe, and extending the compiler to the state where it could compile T_EX (done by Piotr Carlson). The IBM 3287 graphic printer output was done by Hanna Kołodziejska. It was she who inherited further adaptation work from Piotr Carlson.

All this was only possible thanks to the then Head of the Computing Center of the Informatics Institute, Dr. Sc. Stanisław Waligórski, a kind⁶ and far-sighted man who got interested in T_EX. It was he who allocated machine time in the Computing Centre under his command. It was he who made it possible for Hanna Kołodziejska — on the suggestion of Professor Bień — to work for many months on the Polish language hyphenation patterns.

⁵ It should be noted that 20 US dollars was worth an average monthly salary in Poland.

⁶ I personally had the pleasure to experience his kindness when he later was my dean.

Professor Bień also proposed the first, quick and dirty, method of adding Polish ogoneks to ‘a’ and ‘e’, a thing Donald Knuth somehow omitted in the fever of the battle. And so in April 1985 the four-liner by Julian Tuwim was typeset with T_EX 0.8 and several months later version 1.1 arrived from Stockholm.

The next planned step was to install T_EX at the then biggest civil computing center in Poland: the Computing Center of Warsaw University, where the RIAD 60 (an IBM 370/165 Russian made clone) ruled, later replaced by a BASF machine (again an IBM copy). I worked there at the time and observed from a distance the T_EX JOBs. It all ended with a series of publications,⁷ as we were all taken by surprise by the microcomputer era.⁸

L_EX and M_EX

Before the world was taken by wordmania T_EX had been used in Poland since 1987 to typeset many publications and books in various areas, even in Braille.



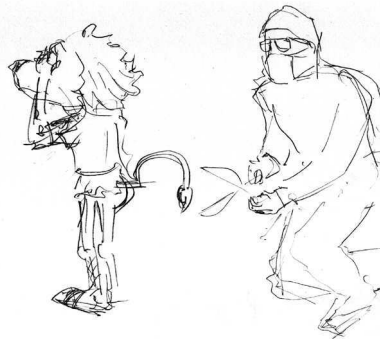
Many a bright man decided to join the crazy crew *polishing* the deck of the ship named T_EX to get it to the Polish language harbour with any amount of spit and polish required. We Poles have something in our veins that drives us always towards Poland,⁹ therefore it would be futile to try to enumerate all those who have been polishing T_EX’s deck. In 1987

⁷ E.g., Janusz Bień, Hanna Kołodziejka, *T_EX for RIAD computers*. In: Dario Lucarella, editor, *Proceedings of the first European Conference on T_EX for Scientific Documentation*, Como, Italy, pages 133–140. Addison-Wesley, Reading, Mass., 16–17 May 1985. Thus we were represented at the first European T_EX Conference.

⁸ Based on Janusz S. Bień, *Co to jest T_EX?*, Instytut Informatyki Uniwersytetu Warszawskiego, Warszawa 1988, <http://www.mimuw.edu.pl/~sbien/publikacje/cttex90.pdf>

⁹ Already in the early 16th century, Mikołaj Rej, the first outstanding Polish writer, advocated the use of the Polish language—as opposed to Latin—for writing by saying that other nations should know that Poles are not geese, that they have their own language: “A niechaj narodowie wždy postronni znają, iż Polacy nie geśi, iż swój język mają.”

Bogusław Jackowski and Marek Ryćko, both looking for a decent typesetting tool, took the steering wheel of the T_EX craft. They were taking turns at the watch of T_EX and METAFONT even before the “eight bit era”. And thus in 1989 we had the first fitting of a Polish T_EX with a set of plain macros under the name L_EX¹⁰ and the CM family of fonts augmented with the Polish ogonek under the shy name of p1 (not yet even pl).



It is worth mentioning that until then no typographically correct ogonek existed in T_EX for the ‘ą’ and ‘ę’ glyphs. There were various attempts to solve this, even by using the French cedilla.

The result was that either the shape was not right, or the direction of the ogonek was wrong or one could not bear looking at it or one could not bear reading it. To keep it short—a surgery on T_EX’s ogonek was required.¹¹ The first operation on the ogonek by Jacko and Marek turned it right, which is... right.¹² The slash notation was also introduced on this occasion (/a, /c, /e, /l, /n, /o, /s, /x, /z) to cater to at least some portability of texts. It is still used by some.



During the polishing Jacko got hurt by some splinters in the deck. As the Admiralty used to compensate for splinters, he received an “acknowledgment” in writing for T_EX, in 1989, at \$327.68 and another one, in 1994, for METAFONT. The Admiral-

¹⁰ The legend has it that there were three brothers, fathers of the Slavic tribes: Lech, Czech and Rus. The name of our forebear—Lech—is pronounced almost as Donald Knuth wants T_EX to be pronounced except of course that ‘L’ replaces ‘T’.

¹¹ Let it be known that “ogonek” is a “small tail” in Polish.

¹² “Let them have an outstanding left, I have an outstanding right, I jump from the right.”, Włodzimierz Wysocki, A high jumper’s song, from the Polish translation by Wojciech Młynarski.

ty—except for these formal written “acknowledgments”—was not overly talkative.

In the beginning not only the ogonek posed problems. It also was not clear what codes should be assigned to our Polish glyphs. And even when the third version of T_EX gave us eight bits of input, we had by then a dozen or so ways of placing the Polish glyphs in the upper half of code pages. The most popular was Mazovia, a standard created by Poles themselves to reconcile our needs with the need to use western European glyphs.

When I met Marek and Jacko a little bit later, they were scrubbing the T_EX deck day and night. They wanted it to become available even under the proverbial thatched roof,¹³ but the situation was such that all modern Poles were using the Mazovia encoding. It became apparent that it would be immensely difficult to popularize T_EX without adapting it in such a way that it would accept eight bit Polish language texts. Thus M_EX¹⁴ and I_AM_EX came into being.

With them, the shape of ogonek got its final polish. A world renowned typographer, Roman Tomaszewski, helped to achieve this.

There are still some who use the Mazovia encoding, although Microsoft has buried it under the 852 and 1250 code pages, and the Internet added another layer with ISO 8859-2. What would the world look like without TCX? Luckily, at the EuroT_EX conference in Cork, the Polish diacriticals have been given the best available places—provided they did not interfere with other nationalities’ diacriticals. . .

The need for good taste aka GUST

Oh, if we had GUST¹⁵ before the “cork expedition!” But we finally understood with the help of Malcolm Clark, who had been talking Marek and Jacko into it, that without our own user group, i.e., our own Admiralty, we will always be perceived as pirates on the T_EX ocean.

The birth of GUST integrated Polish T_EX users—all able crew began scrubbing the deck in concert now. Apart from that, the regular communication enabled by Internet made our voices heard where necessary, sometimes against the will

¹³ “To get something under the thatched roofs”—make it available to everybody.

¹⁴ Standing (a little) for Mazovia–T_EX but some would say Macrosoft–T_EX (Macrosoft, a company which harboured the then most active crew). Besides, it means moss when pronounced similarly to how Don Knuth wants to pronounce T_EX. The M_EX package was awarded the President of the Upper Silesian branch of the Polish Computer Society prize at the Softarg fair in 1992.

¹⁵ Not incidentally, “gust” is Polish for good taste.

of some neighbouring nations.¹⁶ And when the capital of Poland was moved by GUST to Toruń—the town of Copernicus—neither Warsaw nor Gdańsk resented it.

Polish fonts

Polish fonts is the activity where such passionates as Janusz “Uhlan” Nowacki from Grudziądz and our unwavering captain Bogusław “Jacko” Jackowski found their destiny. GUST was also able to subsidize to some extent the public domain work, to which undoubtedly the fonts belong. The first thing to do was supplement the CM fonts in Adobe Type 1 format with Polish diacriticals—DC fonts later inherited those outlines as the source of Polish characters.

But this was not all, by far. We began yearning for a font which our fairy tales and legends had been typeset in: the Antykwa Toruńska. Our beloved Uhlan miraculously persuaded the then still living author, Zygfryd Gardzielewski, to make available his original drawings and meticulously turned them into Type 1 outlines.

Jacko and Uhlan, supported by Piotr Strzelczyk, all united into the JNS team with the aim to overcome the Cork problem by looking for a way to place the 18 Polish diacriticals where we needed them. It was the beginning of the QX font layout.

An attempt at the digitization of yet another font we are attached to because the obituaries of our grandfathers and fathers were typeset with it—Antykwa Póltawskiego—had to be even more systematic as it required reconstructing the font because no original design drawings were preserved. It is now publically available.¹⁷

The TCX battle

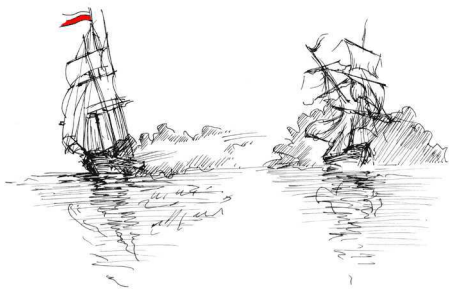
Some of our neighbours were convinced (and possibly still are) that Poles are a messy nation and that they should be taught order because “order must be.” As they themselves in a democratic way

¹⁶ Recalls Marek Ryćko: In Cork there was a moment when representatives of poor countries (those without the Internet) were invited to a place where various T_EX bits and pieces were copied onto diskettes. It was a nice thing that we were catered for, but the joy quickly turned into rage when we found out, that at the same time the other boys had a meeting during which there was the final voting on the encoding for the extended CM family of fonts, now known as the “Cork encoding”. The encoding was arrived at with the use of email communication then unavailable to us. It is unsuitable for Polish but, e.g., catered for the German needs.

¹⁷ Jackowski, B. “Antykwa Póltawskiego: a parameterized outline font”, Proceedings of the EuroT_EX’99 Conference “Paperless T_EX”, Heidelberg, Germany, September, 1999.

finally got rid of diacriticals from their own languages, they could not understand that we Poles have $9 \times 2 = 18$ of them and that because of the order imposed on us by various foreign authors we are forced to place them in various places. The most difficult part was to convince them that we are attached to our ogonek, and on top of it all we like plain $\text{T}_{\text{E}}\text{X}$, which makes it difficult to reconcile both things with the 852, 1250, ISO 8859-2, as well as Mazovia, code pages. We were being pointed to *inputenc* in $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X} 2_{\epsilon}$ without taking into account the difficulties of this method.

And then, in 1995, the animators of the GUST Bulletin, Włodek Bzyl and Staszek Wawrykiewicz found in the code of Web2c a novelty: a piece of code by Karl Berry — an encoding handler called TCX. This was something we liked. Perhaps too much. It looked like we were asking for the moon and at the same time created a storm in which TCX took the role of the Flying Dutchman by alternately showing up and disappearing. TCX needed polishing and at the same time those who likened this approach to dirty hacking tricks had to be nagged and nagged and nagged again.



The TCX battle ensued with salvos exchanged now and then. Masts and ports were broken. The fierce email war was eventually won by Włodek Bzyl, Staszek Wawrykiewicz and Marcin Woliński. Thanks to them even very old Polish texts now compile easily.

$\text{T}_{\text{E}}\text{X}$ Live

No battle and no expedition would succeed without a boatswain. And it's no mean boatswain we have on board. It is StaW¹⁸ who knew from the very beginning where what is and what fits what — i.e., what and where should be installed for $\text{T}_{\text{E}}\text{X}$ et al. to function properly, and whom to shout at if things are not as they should be.

He was the master of distributions and servers. He was also one of those who initiated the good GUST. And last but not least it was he who has

¹⁸ Staszek Wawrykiewicz.

been issuing orders more understandable than those by the Admiralty.¹⁹

After the victorious battle which made TCX famous far and wide,²⁰ his keeping clean of our deck made him famous among others hence today you will not find a $\text{t}_{\text{E}}\text{X}$, $\text{f}_{\text{P}}\text{T}_{\text{E}}\text{X}$ nor $\text{MikT}_{\text{E}}\text{X}$ without Staszek's fingerprints.


Captains visit ships' decks but boatswains are there always, thus it is plainly impossible to list everything for which StaW deserves credit. We are happy that the $\text{T}_{\text{E}}\text{X}$ world appreciates what he does and wants to work with him. In Poland $\text{T}_{\text{E}}\text{X}$ is aLive mainly thanks to his efforts. His arduous work makes every new deckhand feel at home aboard $\text{T}_{\text{E}}\text{X}$. And as Staszek has now become the ambassador to CTAN, I rest assured about the future of $\text{T}_{\text{E}}\text{X}$ archives.

$\text{P}^{\text{L}}\text{A}^{\text{T}}\text{E}^{\text{X}}$

$\text{L}^{\text{A}}\text{M}_{\text{E}}\text{X}$ disappeared with $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X} 2.09$. There was no good reason for fixing the base code to adapt it to the Polish language. This could now be done in the form of a package. Then again our good GUST did bear fruit. Mariusz Olko started with the PoPolsku package in 1994. In 1997 the package began morphing into $\text{P}^{\text{L}}\text{A}^{\text{T}}\text{E}^{\text{X}}$ and its author into Marcin Woliński.

Thanks to their work publications made with $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ have the desired polished look and feel with all 18 Polish diacriticals. And letters written in the Polish language really look Polish and not English or German.

Today this package gives $\text{T}_{\text{E}}\text{X}$ the position it deserves. Many young people reach for it especially when tired of the schizophrenia induced by word-mania. Moreover, the package lives and is constantly being updated. The GUST discussion list attracts many novice users who enter their adventurous path with $\text{P}^{\text{L}}\text{A}^{\text{T}}\text{E}^{\text{X}}$.

A new generation is being born to live on the clean, scrubbed and polished $\text{T}_{\text{E}}\text{X}$ deck. Using the GUST discussion list it seeks the old sea dogs' advice which they patiently give over and over again along with the tips and tricks of the sailors' world whose waters were first charted by Professor Madey. 

¹⁹ Starting from the translation of Michael Doob's "Gentle Introduction to $\text{T}_{\text{E}}\text{X}$ " through translating WinShell up to polishing Eitan Gurari's $\text{T}_{\text{E}}\text{X}4\text{ht}$ into sync with ISO 8859-2.

²⁰ External TCX translation tables have been introduced into all distributions based on Web2c and later into the $\text{MikT}_{\text{E}}\text{X}$ distribution. Now TCX is handled by $\text{pdfT}_{\text{E}}\text{X}$, $\epsilon\text{-T}_{\text{E}}\text{X}$, METAFont and MetaPost which ensures the presence of national glyphs on the screens, in the log files, contents tables, indexes and the like. Thanks to TCX, $\text{T}_{\text{E}}\text{X}$ handles national characters in the same way in formats like $\text{p}^{\text{L}}\text{a}^{\text{T}}\text{e}^{\text{X}}$, $\text{ConT}_{\text{E}}\text{Xt}$, $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$, eplain , or $\mathcal{A}\mathcal{M}\mathcal{S}\text{-T}_{\text{E}}\text{X}$. This approach has been accepted by many users, not only in Poland.

**That pesky lion, or about that lion,
or how a lion came to be**

Duane R. Bibby
Lake Havasu, AZ
drbibby@ctaz.com

Slide show and talk includes early rough sketches of T_EX, Meta, and associated offspring plus other obscure drawings and sketches.

(One sketch is below; a few of the others are scattered throughout this issue — enjoy! *Ed.*)



**The spread of T_EX in India: The role of
outsourced typesetting**

Ajit Ranade
ABN AMRO Bank, India
ajit.ranade@abnamro.com

Unlike in many other countries, the spread of T_EX in India was strongly catalysed by the increasing outsourcing of typesetting, especially by publishers of scientific journals and books. Users from Indian academic institutions played a marginal role in the initial push of T_EX in India. Indeed the Indian T_EX Users Group, now five years old, was the result of the initiative of a commercial typesetting outfit. The outsourcing momentum continues to be strong, fuelled by newer standards and technologies such as XML, and the demand for documents that can be disseminated across different media, such as e-books and internet. There is now a critical mass of T_EX users across different vendors, who depend on T_EX for their livelihood.

But there are significant challenges to harnessing this critical mass and make T_EX even more widespread. Skills spillover outside of typesetting vendors seems to be inhibited by commercial considerations. Training remains an underdeveloped area, especially since the user base in academic institutions is still small.

However, the exciting development of Indic T_EX with a simultaneous increase in literacy and readership of vernacular press offers an opportunity for a much bigger scope for T_EX in India, given the low cost that T_EX implies for publishing. In this paper we analyse these trends, and identify the challenges and opportunities for the greater deployment of T_EX in India.

Calendar

2004

- May 20– Jul 7 In Flight: A traveling juried exhibition of books by members of the Guild of Book Workers. Emory University, Atlanta, Georgia. Sites and dates are listed at <http://palimpsest.stanford.edu/byorg/gbw>.
- Jun 11–16 ALLC/ACH-2004, Joint International Conference of the Association for Computers and the Humanities, and Association for Literary and Linguistic Computing, “Computing and Multilingual, Multicultural Heritage”, Göteborg University, Sweden. For information, visit <http://www.hum.gu.se/allcach2004/> or the organization web site at <http://www.ach.org>.
- Jun 14 GUTenberg Journée L^AT_EX, Paris, France. For information, visit <http://www.gutenberg.eu.org/manifestations/gut2004/>.
- Jun 24–29 2nd International Conference on Typography and Visual Communication: Communication and new technologies, Thessaloniki, Greece. For information, visit <http://www.uom.gr/uompress/>.
- Jul 5– Aug 6 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>.
- Jul 16– Aug 28 In Flight: A traveling juried exhibition of books by members of the Guild of Book Workers. Columbia College, Chicago, Illinois. Sites and dates are listed at <http://palimpsest.stanford.edu/byorg/gbw>.
- Jul 19–22 **Practical T_EX 2004**, San Francisco, California. A user-oriented conference sponsored by TUG. For information, visit <http://www.tug.org/practicaltex2004/>.
- Jul 20–24 SHARP Conference (Society for the History of Authorship, Reading and Publishing), Lyon, France. For information, visit <http://sharpweb.org/>.
- Jul 21–25 TypeCon2004, “Type High”, San Francisco, California. For information, visit <http://www.typecon2004.com/>.
- Aug 2–6 *Extreme* Markup Languages 2004, Montréal, Québec. For information, visit <http://www.extrememarkup.com/extreme/>.
- Aug 8–12 SIGGRAPH 2004, Los Angeles, California. For information, visit <http://www.siggraph.org/calendar/>.
- Aug 14–17 International Conference on Computing, Communications and Control Technologies, University of Texas, Austin, Texas. For information, visit <http://www.iiisci.org/ccct2004/website/>.
- Aug 16–20 Seybold San Francisco, San Francisco, California. For information, visit <http://www.seybold365.com/sf2004/>.
-
- TUG 2004**
Democritus University of Thrace,
Xanthi, Greece.
- Aug 30– Sep 3 The 25th annual meeting of the T_EX Users Group, “XML and Digital Typography”. For information, visit <http://www.tug.org/tug2004/>.
-
- Sep 8–10 IUC26, the 26th Internationalization and Unicode Conference: “Internationalization for an Expanded European Union”, San Jose, California. For information, visit <http://www.unicode.org/iuc/iuc26/>.

Status as of 1 July 2004

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>.

- Sep 13–
Oct 29 In Flight: A traveling juried exhibition of books by members of the Guild of Book Workers. Columbus College of Art & Design, Columbus, Ohio. Sites and dates are listed at <http://palimpsest.stanford.edu/byorg/gbw>.
- Sep 30–
Oct 3 Association Typographique Internationale (ATyPI) annual conference, “Crossroads of Civilizations”, Prague, Czech Republic. For information, visit <http://www.atypi.org/>.
- Oct 2–3 Oak Knoll Book Fest XI, New Castle, Delaware. For information, visit <http://www.oakknoll.com/>.
- Oct 9 First meeting of GuIT (Gruppo utilizzatori Italiani di T_EX), Pisa, Italy. For information, visit <http://www.guit.sssup.it/GuITmeeting/2004/2004.en.html>.
- Oct 18–19 Third Annual St. Bride Conference, “Bad Type”, London, England. For information, visit <http://www.stbride.org/conference.html>.
- Oct 28–30 ACM Symposium on Document Engineering, Milwaukee, Wisconsin. For information, visit <http://www.documentengineering.org>.
- Nov 11–
Dec 31 In Flight: A traveling juried exhibition of books by members of the Guild of Book Workers. Boston Public Library, Boston, Massachusetts. Sites and dates are listed at <http://palimpsest.stanford.edu/byorg/gbw>.
- Nov 15–19 XML Conference & Exposition, “XML—From Syntax to Solutions”, Washington, DC. For information, visit <http://www.xmlconference.org/xmlusa/>.

2005

- Mar 7–11 EuroT_EX 2005, Metz, France. For information, visit <http://www.gutenberg.eu.org/eurotex2005/>.
- Jul 31–
Aug 04 SIGGRAPH 2005, Los Angeles, California. For information, visit <http://www.siggraph.org/calendar/>.

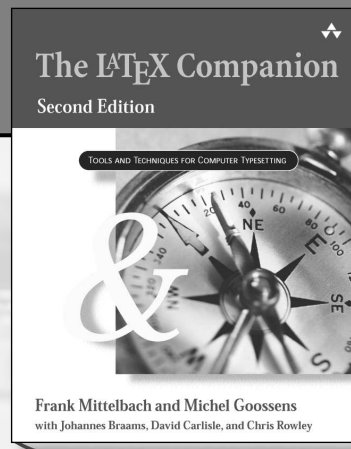
TUG 2005

Wuhan, China.

- Aug 23–25 The 26th annual meeting of the T_EX Users Group. For information, visit <http://www.tug.org/tug2005/>.
-

The L^AT_EX Companion

Second Edition



ISBN: 0-201-36299-6

Frank Mittelbach and Michel Goossens
with *Johannes Braams,*
David Carlisle, and Chris Rowley

The L^AT_EX Companion has long been the essential resource for anyone using L^AT_EX to create high-quality printed documents. This completely updated edition brings you all the latest information about L^AT_EX and the vast range of add-on packages now available—over 200 are covered. Like its predecessor, *The L^AT_EX Companion, Second Edition* is an indispensable reference for anyone wishing to use L^AT_EX productively.

Available at fine bookstores everywhere.


Addison
Wesley

For more information, visit:
[www.awprofessional.com/
titles/0201362996](http://www.awprofessional.com/titles/0201362996)

TUG 2005 Announcement and Call for Papers

TUG 2005 will be held in Wuhan, China, during August 23–25, 2005. CTUG (Chinese T_EX User Group) has committed to undertake the conference affairs, and now announces the call for papers.

Why go to China for TUG 2005?

For fun!

This is the first TUG conference to be held in China. 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.

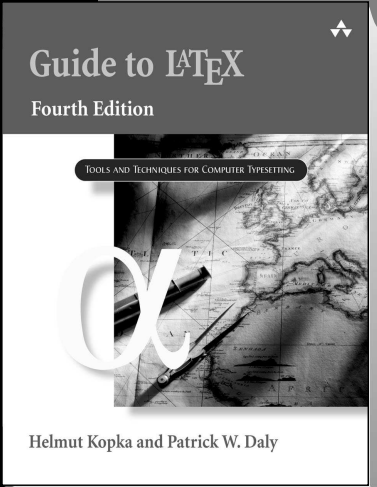
For keeping up with the community!

The T_EX community in China has been growing over the years. China is one of the few countries in the world which has heavily applied free software (including T_EX, GNU/Linux, and more) in industry. The rich human resources and the creative T_EXhackers have become a part of the engine driving the global T_EX community. TUG'05 is a good opportunity to meet them.

For your future!

The growing market is ready to use your expertise. Many libraries, publishing houses, and scientific organizations in China are eager to use your T_EX expertise.

Please submit abstracts for papers to tug2005@tug.org. For more information about TUG 2005, please visit: <http://tug.org/tug2005>



Guide to L^AT_EX

Fourth Edition

TOOLS AND TECHNIQUES FOR COMPUTER DTPSETTING

HELMUT KOPKA AND PATRICK W. DALY

In this completely revised edition, the authors cover the L^AT_EX₂_ε standard and offer more details, examples, exercises, tips, and tricks. They go beyond the core installation to describe the key contributed packages that have become essential to L^AT_EX processing. *Guide to L^AT_EX, Fourth Edition*, will prove indispensable to anyone wishing to gain the benefits of L^AT_EX.

Available at fine bookstores every where.

ISBN: 0-321-17385-6

For more information, visit: www.awprofessional.com/titles/0321173856

Addison Wesley

The T_EX Users Group gratefully acknowledges Apple Computer's generous contributions, especially to the *Practical T_EX 2004* and *TUG 2003* Conferences.

Thank you.

The Apple Store in San Francisco is located at One Stockton Street, San Francisco, CA 94108¹
(This was typeset with the T_EX variant X_YT_EX² created by Jonathan Kew using the Apple System fonts HOEFLER TEXT by Jonathan Hoefler, ZAFFINO by Hermann Zapf and SKIA by Matthew Carter.)
¹<http://www.apple.com/retail/sanfrancisco> ²<http://scripts.sil.org/xetex>

Institutional Members

American Mathematical Society,
Providence, Rhode Island

Banca d'Italia,
Roma, Italy

Center for Computing Science,
Bowie, Maryland

CNRS - IDRIS,
Orsay, France

CSTUG, *Praha, Czech Republic*

Duke University,
Chemistry Library,
Durham, NC

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

Max Planck Institut
für Mathematik,
Bonn, Germany

New York University,
Academic Computing Facility,
New York, New York

Princeton University,
Department of Mathematics,
Princeton, New Jersey

Siemens Corporate Research,
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,
Computing Science Department,
Uppsala, Sweden

Vanderbilt University,
Nashville, Tennessee

T_EX Consulting & Production Services

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 specialty. Custom T_EX macros and L^AT_EX 2_ε document classes and packages. Instruction, support, and consultation for workgroups and authors. Application development in L^AT_EX, T_EX, SGML, PostScript, Java, and C++. Database and corporate publishing. Extensive references.

Veytsman, Boris

2239 Double Eagle Ct.
Reston, VA 20191
(703) 860-0013

Email: borisv@lk.net

I provide training, consulting, software design and implementation for Unix, Perl, SQL, T_EX, and L^AT_EX. I have authored several popular packages for L^AT_EX 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 Unicorn Collaborative, Inc., Ted Zajdel

115 Aspen Drive, Suite K
Pacheco, CA 94553
(925) 689-7442

Email: contact@unicorn-collab.com

We are a technical documentation company, initiated in 1990, which strives for error free, seamless documentation, delivered on time, and within budget. We provide high quality documentation services such as document design, graphic design and copy editing. We have extensive experience using tools such as FrameMaker, T_EX, L^AT_EX, Word, Acrobat, and many graphics programs. One of our specialties is producing technical manuals and books using L^AT_EX and T_EX. Our experienced staff can be trained to use any tool required to meet your needs. We can help you develop, rewrite, or simply copy-edit your documentation. Our broad experience with different industries allows us to handle many types of documentation including, but not limited to, software and hardware systems, communications, scientific instrumentation, engineering, physics, astronomy, chemistry, pharmaceuticals, biotechnology, semiconductor technology, manufacturing and control systems. For more information see our web page: <http://www.unicorn-collab.com>.

The information here comes from the consultants themselves. We do not include any 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 have no opinions and usually no information about the abilities of any specific person. 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 T_EX workers. If you'd like to be included, or run to a larger ad in *TUGboat*, please contact the office or see our web pages:

T_EX 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>