

TUGBOAT

Volume 29, Number 3 / 2008
TUG 2008 Conference Proceedings

TUG 2008	350	Conference program, delegates, and sponsors
	352	Peter Flynn / <i>TUG 2008: T_EX's 30th birthday</i>
L^AT_EX	356	Niall Mansfield / <i>How to develop your own document class — our experience</i>
Software & Tools	362	Jonathan Kew / <i>T_EXworks: Lowering the barrier to entry</i>
	365	Jérôme Laurens / <i>Direct and reverse synchronization with SyncT_EX</i>
	372	Joachim Schrod / <i>Xindy revisited: Multi-lingual index creation for the UTF-8 age</i>
	380	Taco Hoekwater / <i>MetaPost developments: MPlib project report</i>
	383	Hans Hagen / <i>The T_EX–Lua mix</i>
	376	Joe McCool / <i>A newbie's experiences with Lilypond, Lilypond-book, L^AT_EX and Perl</i>
	392	Krisztián Pócsa, Mihály Biczó and Zoltán Porkoláb / <i>docx2tex: Word 2007 to T_EX</i>
	401	Jean-Michel Hufflen / <i>Languages for bibliography styles</i>
Dreamboat	413	Chris Rowley / <i>Vistas for T_EX: liberate the typography! (Part I)</i>
Fonts	418	Dave Crossland / <i>Why didn't METAFONT catch on?</i>
	421	Karel Píška / <i>Creating cuneiform fonts with MetaType1 and FontForge</i>
	426	Ulrik Vieth / <i>Do we need a 'Cork' math font encoding?</i>
	435	Ameer Sherif and Hossam Fahmy / <i>Meta-designing parameterized Arabic fonts for AlQalam</i>
Graphics	444	Manjusha Joshi / <i>Smart ways of drawing PSTricks figures</i>
	446	Hans Hagen / <i>The MetaPost library and LuaT_EX</i>
Philology	454	Mojca Miklavec and Arthur Reutenauer / <i>Putting the Cork back in the bottle — Improving Unicode support in T_EX</i>
	458	Stanislav Jan Šarman / <i>Writing Gregg Shorthand with METAFONT and L^AT_EX</i>
Macros	462	Hans Hagen / <i>The LuaT_EX way: \framed</i>
Electronic Documents	464	Ross Moore / <i>Advanced features for publishing mathematics, in PDF and on the Web</i>
	474	John Plaice, Blanca Mancilla and Chris Rowley / <i>Multidimensional text</i>
	480	Manjusha Joshi / <i>Data mining: Role of T_EX files</i>
Abstracts	482	Abstracts (Fine, Hagen, Henkel, Hoekwater, Høgholm, Küster, Mancilla et al., Mittelbach, Peter, Rahilly et al., Rhatigan, Veytsman & Akhmadeeva, Veytsman)
News	485	Calendar
	486	TUG 2009 announcement
Advertisements	487	T _E X consulting and production services
TUG Business	484	TUG institutional members
	488	TUG 2009 election

TeX Users Group

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

Memberships and Subscriptions

2008 dues for individual members are as follows:

- Ordinary members: \$85.
- Students/Seniors: \$45.

The discounted rate of \$45 is also available to citizens of countries with modest economies, as detailed on our web site.

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

Also, (non-voting) *TUGboat* subscriptions are available to organizations and others wishing to receive *TUGboat* in a name other than that of an individual. The subscription rate is \$95 per year, including air mail delivery.

Institutional Membership

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

TeX is a trademark of the American Mathematical Society.

Copyright © 2008 TeX Users Group.

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

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

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

Board of Directors

Donald Knuth, *Grand Wizard of TeX-arcana*[†]
Karl Berry, *President*^{*}
Kaja Christiansen*, *Vice President*
David Walden*, *Treasurer*
Susan DeMeritt*, *Secretary*
Barbara Beeton
Jon Breitenbucher
Steve Grathwohl
Jim Hefferon
Klaus H"oppner
Dick Koch
Ross Moore
Arthur Ogawa
Steve Peter
Cheryl Ponchin
Philip Taylor
Raymond Goucher, *Founding Executive Director*[†]
Hermann Zapf, *Wizard of Fonts*[†]

^{*}member of executive committee

[†]honorary

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

Addresses

TeX Users Group
P. O. Box 2311
Portland, OR 97208-2311
U.S.A.

Telephone

+1 503 223-9994

Fax

+1 206 203-3960

Web

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

Electronic Mail

(Internet)

General correspondence,
membership, subscriptions:
office@tug.org

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

Technical support for
TeX users:
support@tug.org

Contact the Board
of Directors:
board@tug.org

Have a suggestion? Problems not resolved?

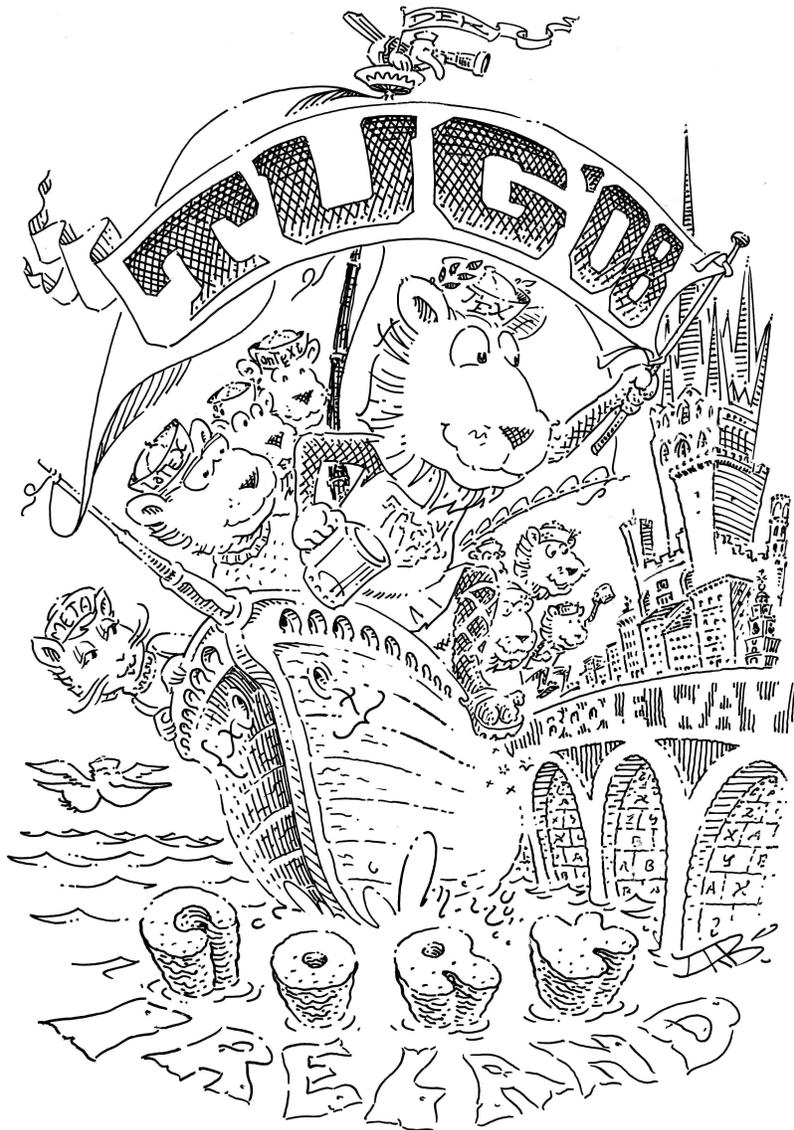
The TUG Board wants to hear from you:
Please email board@tug.org.

[printing date: October 2008]

Printed in U.S.A.

TUGBOAT

The Communications of the TeX Users Group



Volume 29, Number 3, 2008
TUG 2008 Conference Proceedings

TeX Users Group

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

Memberships and Subscriptions

2008 dues for individual members are as follows:

- Ordinary members: \$85.
- Students/Seniors: \$45.

The discounted rate of \$45 is also available to citizens of countries with modest economies, as detailed on our web site.

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

Also, (non-voting) *TUGboat* subscriptions are available to organizations and others wishing to receive *TUGboat* in a name other than that of an individual. The subscription rate is \$95 per year, including air mail delivery.

Institutional Membership

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

TeX is a trademark of the American Mathematical Society.

Copyright © 2008 TeX Users Group.

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

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

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

Board of Directors

Donald Knuth, *Grand Wizard of TeX-arcana*[†]
Karl Berry, *President*^{*}
Kaja Christiansen*, *Vice President*
David Walden*, *Treasurer*
Susan DeMeritt*, *Secretary*
Barbara Beeton
Jon Breitenbucher
Steve Grathwohl
Jim Hefferon
Klaus Höppner
Dick Koch
Ross Moore
Arthur Ogawa
Steve Peter
Cheryl Ponchin
Philip Taylor
Raymond Goucher, *Founding Executive Director*[†]
Hermann Zapf, *Wizard of Fonts*[†]

^{*}member of executive committee

[†]honorary

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

Addresses

TeX Users Group
P. O. Box 2311
Portland, OR 97208-2311
U.S.A.

Telephone

+1 503 223-9994

Fax

+1 206 203-3960

Web

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

Electronic Mail

(Internet)

General correspondence,
membership, subscriptions:
office@tug.org

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

Technical support for
TeX users:
support@tug.org

Contact the Board
of Directors:
board@tug.org

Have a suggestion? Problems not resolved?

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

[printing date: October 2008]

Printed in U.S.A.

TUG 2008 Proceedings

University College Cork

Ireland

July 21–24, 2007

TUGBOAT

COMMUNICATIONS OF THE \TeX USERS GROUP

TUGBOAT EDITOR BARBARA BEETON

PROCEEDINGS EDITOR KARL BERRY

VOLUME 29, NUMBER 3

PORTLAND

•

OREGON

•

•

2008

U.S.A.

TUG 2008 — T_EX's 30th birthday

July 20–24, 2008

University College Cork, Ireland

Sponsors

T_EX Users Group ■ DANTE e.V. ■ Human Factors Research Group, UCC
Carleton Production Centre ■ Malcolm Clark ■ River Valley Technologies

Thanks to the sponsors, and to all the speakers, teachers, and participants, without whom there would be no conference. The session chairs and speakers deserve special recognition for adhering to a extra-tight schedule this year. Special thanks also to Kaveh Bazargan for the video recording, and Duane Bibby for the (as always) excellent drawing.

Conference committee

Anita Schwartz ■ Peter Flynn ■ Robin Laakso ■ Karl Berry

Participants

Leila Akhmadeeva, Bashkir State Medical Univ., Russia
Edd Barrett, T_EX Live / OpenBSD
Kaveh Bazargan, River Valley Technologies
Nelson Beebe, University of Utah
Barbara Beeton, American Mathematical Society
Mihály Biczó, Eötvös Loránd University, Hungary
Johannes Braams, L^AT_EX Project
Jennifer Claudio, St. Lawrence Academy
David Crossland, Wimborne, UK
Sue DeMeritt, Center for Communications Research
Christine Detig, Net & Publication Consultance GmbH
Michael Doob, University of Manitoba
Hossam Fahmy, Cairo University, Egypt
Jonathan Fine, The Open University, UK
Peter Flynn, University College Cork, Ireland
Ralf Gaertner, München, Germany
Michel Goossens, CERN
Steve Grathwohl, Duke University Press
Adelheid Grob, Universität Ulm
Hans Hagen, Pragma ADE
Hartmut Henkel, von Hoerner & Sulger GmbH
Morten Høgholm, L^AT_EX Project
Taco Hoekwater, Elvenkind BV
Klaus Höppner, DANTE e.V.
Jean-Michel Hufflen, Université de Franche-Comté
Jelle Huisman, SIL International
Manjusha Joshi, Bhaskaracharya Institute in Mathematics, India
Jonathan Kew, Thame, UK
Timothy Kew, Thame, UK
Peter Knaggs, Bournemouth, UK
Thomas Koch, Köln, Germany
Harald König, Balingen, Germany
Reinhard Kotucha, Hannover, Germany
Valentinas Kriauciukas, VT_EX
Martha Kummerer, University of Notre Dame
Johannes Küster, typoma GmbH
Robin Laakso, T_EX Users Group
Jérôme Laurens, Université de Bourgogne
Dag Langmyhr, University of Oslo
Olga Lapko, Moscow, Russia
Blanca Mancilla, University of New South Wales
Niall Mansfield, UIT Cambridge Ltd, UK
Patricia Masinyana, UNISA, South Africa
Joe McCool, Southern Regional College, UK
Stephen McCullagh, Dublin Inst. for Advanced Studies
Frank Mittelbach, L^AT_EX Project
Ross Moore, Macquarie University, Australia
Winfried Neugebauer, Bremen, Germany
Manuel Pégourié-Gonnard, T_EX Live
Steve Peter, Pragmatic Programmers
Karel Píška, Academy of Sciences, Czech Republic
John Plaice, University of New South Wales
Krisztián Póczy, Eötvös Loránd University, Hungary
Cheryl Ponchin, Institute for Defense Analyses
Toby Rahilly, University of New South Wales
Arthur Reutenauer, GUTenberg
Daniel Rhatigan, The University of Reading
David Roderick, Carlisle, UK
Chris Rowley, L^AT_EX Project
Stanislav Šarman, Clausthal Univ. of Tech., Germany
Volker RW Schaa, DANTE e.V.
Joachim Schrod, Net & Publication Consultance GmbH
Torsten Schuetze, Möglingen, Germany
Herbert Schulz, Naperville, Illinois
Anita Schwartz, University of Delaware
Heidi Sestrich, Carnegie Mellon University
Martin Sievers, Trier, Germany
Linas Stonys, VT_EX
Sigitas Tolusis, VT_EX
Eva van Deventer, UNISA, South Africa
Marc van Dongen, University College Cork, Ireland
Boris Veytsman, George Mason University
GS Vidhya, River Valley Technologies
Ulrik Vieth, Stuttgart, Germany
Alan Wetmore, US Army Research Laboratory

TUG 2008 — program

Monday, July 21

- 9:00 Peter Flynn, *opening*
9:15 Frank Mittelbach, *Windows of opportunity: A (biased) personal history of two decades of L^AT_EX development — are there lessons to be learned?*
10:00 Steve Peter, *A pragmatic toolchain: T_EX and friends and friends of friends*
10:30 *break*
10:45 Niall Mansfield, *How to develop your own document class — our experience*
11:15 Joe McCool, *A newbie's experiences with Lilypond, Lilypond-book, L^AT_EX, and Perl*
11:45 Jonathan Fine, *Why we need L^AT_EX3*
12:30 *lunch*
2:15 Krisztián Pócza, Mihály Biczó & Zoltán Porkoláb, *docx2tex: Word 2007 to T_EX*
2:45 Jonathan Kew, *T_EXworks: Lowering the barrier to entry*
3:15 *break*
3:30 Manjusha Joshi, *Data mining: Role of T_EX files?*
4:00 Taco Hoekwater, *LuaT_EX: What has been done, what will be done*
4:30 Hans Hagen, *LuaT_EX: The T_EX-Lua mix*
reception

Tuesday, July 22

- 9:00 Mojca Miklavec & Arthur Reutenauer, *Putting the Cork back in the bottle: Improving Unicode support in T_EX extensions*
9:30 Joachim Schrod, *xindy revisited — multilingual index creation for the UTF-8 age*
10:00 Ulrik Vieth, *Do we need a Cork math font encoding?*
10:30 *break*
10:45 Daniel Rhatigan, *Three typefaces for mathematics*
11:15 Johannes Küster, *Minion Math: The design of a new math font family*
11:45 Karel Píška, *Creating cuneiform fonts with MetaType1 and FontForge*
12:15 *lunch*
1:30 Ameer Sherif & Hossam Fahmy, *Meta-designing parameterized Arabic fonts for AlQalam*
2:00 Stanislav Šarman, *Writing Gregg Shorthand with L^AT_EX and Metafont*
2:30 Dave Crossland, *Why didn't Metafont catch on?*
3:00 *break*
3:15 John Plaice, Blanca Mancilla & Toby Rahilly, *Multidimensional text*
3:45 Blanca Mancilla, John Plaice & Toby Rahilly, *Multiple simultaneous galleys: A simpler model for electronic documents*
4:15 Toby Rahilly, John Plaice & Blanca Mancilla, *Parallel typesetting*

Wednesday, July 23

- 9:15 Manjusha Joshi, *Smart ways of drawing PSTricks figures*
9:45 Boris Veytsman & Leila Akhmadeeva, *Medical pedigrees with T_EX and PSTricks: New advances and challenges*
10:30 *break*
10:45 Jonathan Fine, *MathTran and T_EX as a web service*
11:15 Ross Moore, *Advanced features for publishing mathematics, in PDF and on the Web*
11:45 Morten Høgholm, *The galley module, or How I Learned to Stop Worrying and Love the Whatsit*
12:15 Jérôme Laurens, *Direct and reverse synchronization with SyncT_EX*
12:45 *lunch*
Afternoon excursions, coaches expected to leave around 2pm.
7pm *banquet*

Thursday, July 24

- 9:00 Taco Hoekwater, *MPLib: The project, the library and the future*
9:30 Hartmut Henkel, *Image handling in LuaT_EX*
9:30 Hans Hagen, *MPLib: An example of integration*
10:30 *break*
10:45 Hans Hagen, *surprise LuaT_EX talk*
11:15 Jean-Michel Hufflen, *Languages for bibliography styles*
11:45 Boris Veytsman, *Observations of a T_EXnician for hire*
12:15 Anita Schwartz, *closing*
12:30 *lunch*

TUG 2008: T_EX's 30th birthday

Peter Flynn
University College Cork
Ireland
<http://tug.org/tug2008>

'Twas the night before TUGconf and all through the
No computer was stirring, not even my mouse. [house
The bags were all stuffed and in boxes for care
In hopes that the delegates soon would be there.
Attendees were nestled all snug in their planes
While visions of typesetting danced in their brains.

I dare say someone with more poetic license can make a better shot at it, but by the night before the 2008 Cork meeting the bags were indeed all ready, thanks to Anita and Tyler Schwartz, Arthur Reutenauer and Karel Píška who gave up their evening to sort T-shirts, mugs, programs, and the assorted bits and pieces while registering the early arrivals. We finally headed for a much-needed beer, and bumped into my son and his girlfriend in the beer-garden (just to show that Cork is actually a village of 200,000 people).

The workshops (PSTricks and L^AT_EX) were very well attended, almost over capacity in one of the rooms, and in addition to the expected content they covered a lot of the vital but informal tips and hints that you only get in face-to-face tuition.

The “Luck of the Irish” brought us excellent weather and wonderful presenters. Everyone did a fantastic job adhering to the schedule and adjusting to all the last minute changes. Thanks to the session chairs, Cheryl Ponchin, Anita Schwartz, myself and our renowned TUG office manager, Robin Laakso.

As the local organiser I couldn't get to as many of the papers as I wanted to, but meeting old and new faces is one of the benefits of conferences, and I always get a lot out of hearing what people have been doing and what they are using T_EX for. Video recordings of many of the talks were made by the stalwart Kaveh Bazargan, so if you could not be present, or for a reminder if you were, visit <http://www.river-valley.tv/conferences/tug2008>.

The excursions were full of history and beautiful sites. Many would agree that most of us attending TUG conferences do not need to kiss the Blarney

Stone for the gift of gab, especially when it comes to our passion for T_EX. However, many of us did enjoy sharing these passionate discussions over Jameson, Guinness and Beamish at the end of each long day. The banquet provided a relaxing evening to enjoy a nice dinner along with the opportunity to learn and enjoy the talents of our user group members outside of T_EX.

A lot has changed since we hosted the 1990 meeting at Cork: fonts, encodings, packages, versions, features, systems, and people (some of us are older and wiser; some of us just older!). Many of these changes were evident in the presentations, and it was good to see so much new work being done. Many of us have had to fend off the “oh, that old thing” response to mentioning T_EX, and perhaps we don't shout loudly enough about all the shiny new features we get to see at conferences.

And a lot hasn't changed: I had occasion to dig out some of the files I used back around 1990, and by changing `\documentstyle` to `\documentclass` and fixing a couple of package names, they worked fine; a tribute to the stability of the underlying design and the work of the maintainers and developers.

Perhaps embarrassingly, some of what hasn't changed still haunts us, although we'd never tell that to the users of InDesign or QuarkXPress or FrameMaker or (gasp) Word or OpenOffice. L_AT_EX is great, but we still don't have an editing interface that non-technical writers can use. X_EL_AT_EX is wonderful but font installation is still a pain. The M_IK_TE_X-derived package managers are cool, but not yet universal. And we still have people using `\bf` and `\it` after all these years.

Next year we're meeting at Notre Dame, so we have ten months to make some more good changes. And they promise to have visitor wireless access, which UCC didn't, no matter how loudly I screamed. See you there!





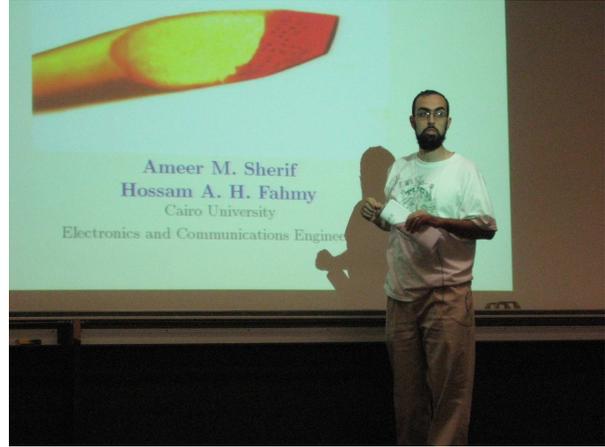
At the reception. *Around the front table:* Hans Hagen, Arthur Reutenauer, Taco Hoekwater, Reinhard Kotucha, Hartmut Henkel, (back of) Volker Schaa. *Behind Hartmut:* Leila Akhmadeeva, Olga Lapko, and Boris Veytsman. *Behind Reinhard:* Heidi Sestrich, Alan Wetmore.



Adelheid Grob, Hans Hagen, Hartmut Henkel, Jean-Michel Hufflen, Harald König, Dag Langmyhr, Michael Doob, Nelson Beebe, Johannes Braams.



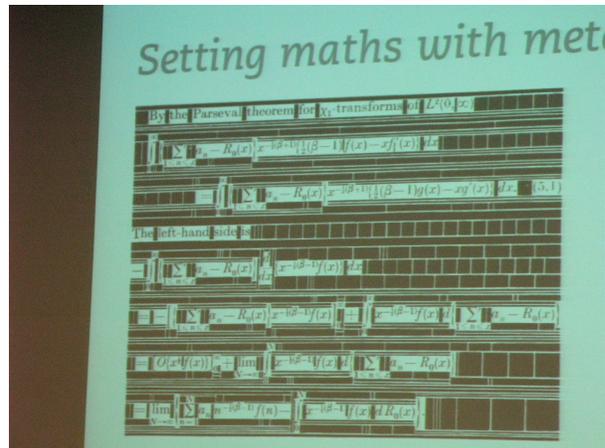
Jonathan Fine.



Hossam Fahmy.



Kaveh Bazargan, recording and listening.



Slide from Daniel Rhatigan's talk.



Klaus Höppner, wielding camera.



At the English Market.

Kneeling: Reinhard Kotucha,
Manuel Pégorié-Gonnard, Dave Crossland.
Standing: Heidi Sestrich, Kaveh Bazargan,
Alan Wetmore, Martha Kummerer,
Barbara Beeton, Manjusha Joshi



Dave Crossland, wielding camera.



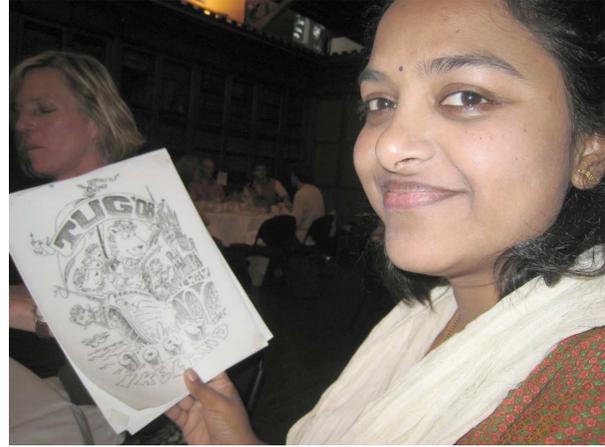
The cliffs of Moher.



Steve Peter, outside the Jameson distillery.



Taco Hoekwater, inside the Beamish brewery.



Manjusha Joshi, winner of the drawing.



Morten Høgholm and family: Evguenia, David, Abigail.



TeX's 30th birthday cake.



Main quad.



Michael Doob, Barbara Beeton, Frank Mittelbach.

Photos courtesy of Jennifer Claudio, Morten Høgholm, Robin Laakso, Steve Peter, and Ulrik Vieth.

How to develop your own document class — our experience

Niall Mansfield

UIT Cambridge Ltd.

PO Box 145

Cambridge, England

tug08 (at) uit dot co dot uk

Abstract

We recently started re-using \LaTeX for large documents — professional computing books — and had to convert an old (1987) \LaTeX 2.09 custom class to work with \LaTeX 2 ϵ . We first tried converting it to a stand-alone `.cls` file, which the documentation seemed to suggest is the thing to do, but we failed miserably. We then tried the alternative approach of writing an “add-on” `.sty` file for the standard `book.cls`. This was straightforward, and much easier than expected. The resulting style is much shorter, and we can use most standard packages to add extra features with no effort.

This paper describes our experience and the lessons and techniques we learned, which we hope will encourage more people to write their own styles or classes.

1 Where we started from

Years ago I wrote a book *The Joy of X* [1], about the X window system. It was in an unusual format called STOP [5], as enhanced by Weiss [6], summarized graphically in Figure 1. In 2008 I wanted to write another book in the same format [2, 3]. It has several interesting features that make it excellent for technical books, although those details are not relevant here. Suffice it to say that STOP required us to change how parts, chapters, sections and sub-sections are handled, and to provide extra sectional units at the beginning and end of each chapter. We also had to provide a summary table of contents, and for each chapter a per-chapter table of contents (TOC) on the first page of the chapter, and use PostScript fonts, which in 1987 was a non-trivial task.

Back in 1987 a colleague of mine, Paul Davis, very kindly wrote the necessary style file for this format, and it worked very well. However, in the meantime the world had moved on from \LaTeX 2.09 to \LaTeX 2 ϵ . The challenge was to provide the functionality of the old style, but under \LaTeX 2 ϵ .

2 First attempt — failure

Where do you start when developing a new style or class? The document *\LaTeX 2 ϵ for class and package writers* says:

if the commands could be used with any document class, then make them a package; and if not, then make them a class.

I took this to mean “We should write a class”. I wrongly went one step further, and thought it also meant we should start our own class from scratch.

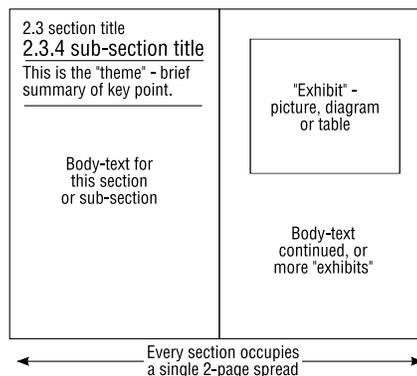


Figure 1: A STOP sub-section or “module”

(Another reason for thinking this was that at least one major publisher seems to have gone this route.)

In fact the same document continues “There are two major types of class: those like `article`, `report` or `letter`, which are free-standing; and those which are extensions or variations of other classes — for example, the `proc` document class, which is built on the `article` document class.” What I ought to have done is started work on an “extension” or “variation” class, but I didn’t realize it.

So I tried to convert the old `.sty` to \LaTeX 2 ϵ , and failed miserably. (This wasn’t surprising, because \LaTeX 2.09 style files consisted of plain \TeX code, which I have always found *very* difficult to understand.) The end-product was something that almost worked, but had lots of small bugs, and when I fixed one problem, the change caused new problems elsewhere.

Another, equally large, disadvantage of this approach was that even if it had worked, the effort to maintain it would have been huge. None of the standard L^AT_EX 2_ε packages would have worked, so if we needed any changes — even “simple” things like changing the page size — we’d have had to code them by hand ourselves.

Lesson A: whether you call your file a class or a style doesn’t matter much — it’s just a matter of a name. What *is* important is not to start from scratch, but to build, as far as possible, using existing code.

At this point we almost gave up and considered using Quark Xpress or InDesign. But luckily someone noticed the `minitoc` package, which looked like it might give us exactly what we needed for our per-chapter TOC, if only we could use it. We decided to try again.

3 Second attempt — a short blind alley

We threw away everything we had and started again from scratch. We tried `book.cls` plus `minitoc`. This addressed one of our most difficult requirements — the per-chapter TOC — and did it so well that we were encouraged to persevere, thank goodness.

We copied the `book.cls` file as `uitbook.cls`, and started adding our own modifications to this. After a few days this became messy, especially when bug-fixing: it wasn’t obvious which was our code (where the bugs were likely to be) and which was the original code.

Lesson B: in L^AT_EX, the way to modify standard code is usually not to modify the original file. Instead, extract just the piece that you want to change, save it as *something.sty* and modify just that little file. Then do `\usepackage{something.sty}`.

4 Third attempt — success!

So we started again, leaving `book.cls` unchanged, and created our own file `uitbook.sty` to contain all our changes. The convention we settled on is:

- If something is just a convenience — e.g. a macro that is merely a shorthand to save typing but doesn’t add any new functionality — we create a small `.sty` for it, and then `\RequirePackage` that. In this way we can re-use the same convenience tools with other classes.

For example, we defined about 12 macros for including graphics or verbatim examples of program code, with or without captions, and with captions in the usual place below the figure or alternatively beside the figure (to save vertical

space). These don’t do anything new, but they all take the same number of arguments in the same order; if a particular variant doesn’t actually need them all, we can just leave the irrelevant ones empty. This makes it easy to change a figure from “no caption” to “side caption” or to “normal caption” with a couple of keystrokes. All these macros are in `uit-figures.sty`.

- Where we make substantial changes, e.g. to the sectioning mechanism or to the format of page headings, we include it directly in our file `uitbook.sty`.

To cheer ourselves up after previous failures, we did all the easy bits first. Those included the convenience macros mentioned above, and the dozens of `\RequirePackage` calls to the packages that we needed:

<code>caption</code>	<code>chnpage</code>	<code>color</code>	<code>colortbl</code>
<code>courier</code>	<code>crop</code>	<code>endnotes</code>	<code>fancyvrb</code>
<code>framed</code>	<code>geometry</code>	<code>graphicx</code>	<code>helvet</code>
<code>hhline</code>	<code>ifthen</code>	<code>latexsym</code>	<code>layout</code>
<code>makeidx</code>	<code>mathpazo</code>	<code>mcaption</code>	<code>minitoc</code>
<code>nextpage</code>	<code>paralist</code>	<code>relsize</code>	<code>showidx</code>
<code>sidecap</code>	<code>ulem</code>	<code>url</code>	<code>wrapfig</code>

At this point things were looking good. We had a style that worked. However, several STOP-specific features were still missing, so that’s what we had to implement next.

The document *L^AT_EX 2_ε for class and package writers* describes the boilerplate for a class or package — analogous to telling a C programmer that he needs a `main()` function, and how to use `#include` statements. What it doesn’t tell you is how the standard classes work, and the common techniques they use. In the following sections we’ll explain the techniques that we came across.

5 The hard bits 1 — over-riding existing functionality

We needed to change the TOC entry for Parts. This is handled in the `\l@part` function in `book.cls`. We copied this function to our `uitbook.sty`, and modified it there. The change involved was only a single line — to use a different font, and insert the word “Part” — but it illustrates a couple of important points:

Lesson C: copying a piece of standard code in L^AT_EX, and changing your own version of it is a bit like over-riding a method in object-oriented programming. Everything that you haven’t changed continues to work as before, but as soon as the relevant function (macro)

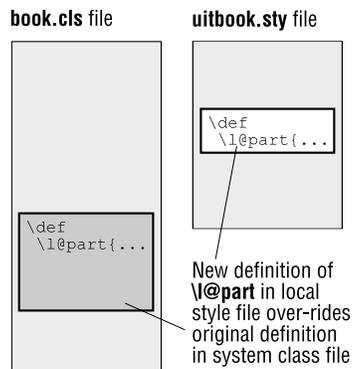


Figure 2: How \LaTeX hooks work

is called — `\l@part` in our example — the new code is used instead of the old (Figure 2).

Lesson D: you don’t have to understand *everything* to make a change to something relatively small. As long as you change as little as possible, you probably won’t break anything else. In our case, even though `\l@part` contains lots of complex stuff, we were confident that our minor format changes would work, because we didn’t modify anything else.

By the way, many functions or macros in the package and class files are defined using the \TeX primitive `\def`, instead of \LaTeX ’s `\newcommand`. If you redefine an existing command with `\def`, you don’t get an error, unlike `\newcommand`’s behavior.

6 The hard bits 2 — \LaTeX hooks

The first clue we got about how \LaTeX 2 packages work with class files, i.e. how they modify their behavior, was reading `ltssect.dtx` — the documented version of the sectioning code in core \LaTeX 2. It has a comment: “Why not combine `\@sect` and `\@xsect` and save doing the same test twice? It is not possible to change this now as these have become hooks!”

What’s a hook? In the Emacs programmable editor, hooks are used to customize the editor’s behavior. For example, `before-save-hook` is a list of Lisp functions that should be run just before a file is saved. By default the list is empty, but by adding your own functions to the list, you can have Emacs perform any special actions you want, such as checking the file into a version control system as well as saving it, etc. Emacs provides about 200 hooks, letting you customize most aspects of its behavior.

In \LaTeX a hook is slightly different. It’s a named function or macro that some other part of the system is going to call. For example, in Section 5 we used `\l@part` as a hook. As we saw, by redefining

`\l@part`, you can change how the TOC entries for your Parts are printed. The hook mechanism and the “over-riding functionality” technique above are more or less the same thing.

Hooks are fundamental to how \LaTeX packages work: they let the package over-ride the standard operation with something different. As an example, consider the `shorttoc.sty` package, which is useful if you want a one-page summary table of contents before the main TOC, for example. The package contains only about 40 lines of code, and in essence, all it does is call the standard table of contents, having first redefined the variable `\c@tocdepth` to a small value to show only the top levels of contents. In effect, `shorttoc.sty` is using all the standard table-of-contents macros as hooks, although it hasn’t changed any of them.

Lesson E: hooks aren’t documented (as far as we’ve been able to see). In fact they can *never* be exhaustively documented, because any package author can just copy any function (as we did with `\l@part` earlier) and over-ride it with their own code, thus using that function as a hook. In real life, the only way you can determine the important hooks is by looking at the important packages, to see which functions they over-ride.

Lesson F: when you copy a chunk of standard code, change the absolute minimum you can get away with. The reason is you don’t really know what parts of it might be used as hooks, or what might happen if you remove a call from it to some other hook. Resist the temptation to tidy or “improve” the code.

7 The hard bits 3 — adding extra functionality

(This section describes a technique that you will often come across, and which you might find useful.)

Let’s say you want to change some function so that it continues to do exactly what it does at present, but does something extra in addition, i.e. the new is a superset of the old. Here’s a real but slightly weird example. The author of a book [4] was using superscripts in his index for a special purpose. We needed a list of the superscripts, and it was difficult to get this from the source files. Using the \TeX primitive `\let`, you can assign a whole function to a new variable, and then call the same old function but with the new name. We used this as follows:

```
\let\origsuper=\textsuperscript
\renewcommand{\textsuperscript}[1]
{\origsuper{XXX(#1)XXX}}
```

This “saves” the original `\textsuperscript` definition as `\origsuper` (Figure 3). Then it redefines `\textsuperscript`, to call the original, unaltered function, but with a modified argument, so that the superscripted text is still superscripted, but is surrounded by the strings `XXX(...)`, which we can then easily search for.¹

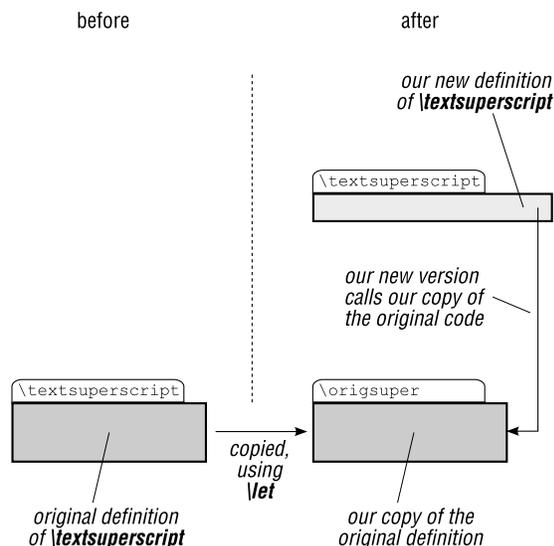


Figure 3: Adding extra functionality to a macro

8 The hard bits 4 — plain \TeX 's syntax

In 1987 I found plain \TeX incomprehensible, and nothing has changed. Using \LaTeX is non-trivial, but it's powerful and the results are more than worth the effort required. For me, the same is not true of plain \TeX : it's too low level, and too complex. Its syntax is weird. Instead of helping you do what you know you need to do, the syntax gets in your way and makes things hard for you. (As an example, we recently found an “off by one” error in a standard package. To fix it, all that was needed was to change ‘if $X > Y$ ’ to ‘if $X \geq Y$ ’, but plain \TeX doesn't let you express things like that, so we had to get someone more experienced in plain \TeX to change the code to do the equivalent.)

So, while plain \TeX is wonderful and is the foundation on which \LaTeX is built, it's not for everyone. (Or, it's for almost no-one?)

Our feeble “solution” to this problem is to avoid it, and when that's not possible, to copy code from packages that work, and hope that $\text{Lua}\TeX$ (www.luaotex.org) will eventually make it easier to code complex or low-level macros.

¹ `catdvi file.dvi | tr -s " \t" "\n" | fgrep 'XXX(' | sort -u`

9 The hard bits 5 — indirection in macro names

(Again, this section describes a common technique that you need to understand, although you might not use it often yourself.)

The \TeX commands `\csname ... \endcsname` let you construct a “control sequence” name, i.e. a macro, programmatically and then invoke it. The following is equivalent to `\textbf{fat cat}`:

```
\newcommand{\mymac}{\textbf}
\csname \mymac \endcsname{fat cat}
```

The first line defines the variable `mymac` to be the string `\textbf`, and the second line uses the variable to construct a macro name and invoke it, passing the argument ‘fat cat’ to it. Being able to invoke a function or macro programmatically like this, instead of having to hard-code its literal name in your `.sty` file, makes it possible to handle many similar but slightly different cases compactly and with little duplication of common code.

The sectioning mechanism uses this technique frequently, to construct names of variables or functions related to the level of the current “sectional unit”² (SU), such as the macros `\l@part`, `\l@chapter`, `\l@section`, etc. We'll look at this in more detail in the next section, but for now, here's a simple but artificial example of how it works. We define a macro `\T`, whose first argument is the style in which its second argument is to be printed:

```
\newcommand{\T}[2]
  {\csname text#1\endcsname{#2}}
Make stuff \T{bf}{heavy}
or \T{it}{slanty}. The end.
```

This produces:

Make stuff **heavy** or *slanty*. The end.

10 The hard bits 6 — changing sectioning

The most difficult thing we had to do was change how sectioning works. (We had to do this because our `STOP` format has to print both section- and sub-section headings on sub-sections.)

For a beginner, sectioning is difficult in three separate ways:

1. There are many functions involved: sections, subsections and lower are defined in terms of `\@startsection`, which then uses `\@sect` (or `\@ssect` if it's a “starred” sectioning command, which in turn calls `\xsect`); all these are complex, and written in plain \TeX , which makes life difficult.

² A sectional unit is a part, chapter, section, subsection, etc.

The way we got over this was by documenting the functions. This is a work in progress, so we've made the rudimentary documentation available on our Web site (uit.co.uk/latex).

- Sectioning uses indirection a lot. Because the same functions (`\@startsection`, etc.) handle many different levels of sectioning, they use indirection to refer to various parameters for the SU being operated on. For example:

- The counters `\c@part`, `\c@chapter`, `\c@section`, `\c@subsection`, ... hold the number of the respective SU.
- The macros `\thepart`, `\thechapter`, `\thesection`, ... specify how the respective counter is formatted. E.g. `book.cls` defines:


```
\renewcommand \thesection
    {\thechapter.\@arabic\c@section}
```

 so that the numbering on a section will be of the form "4.9".
- Similarly, the variables `\l@part`, `\l@chapter`, `\l@section`, ... are what are used to create the table-of-contents entry for the respective SU.

The first argument to the `\@startsection` and `\@sect` functions is the type of the current SU, and the functions use this to construct the relevant item they need, as in:

```
\csname l@#1\endcsname
```

This technique isn't intrinsically difficult, but until you're aware of it, the sectioning mechanism can appear incomprehensible.

- Functions seem to do funny things with their arguments. We cover this in the next section.

11 The hard bits 7 — plain TeX really is a macro processor

The file `book.cls` defines:

```
\newcommand\section
{\@startsection {section} ...
```

i.e. a `\section` is just a call to `\@startsection` with 6 arguments, the first of which is the type of the current SU, as we explained above. However `\@startsection` then calls `\@sect` with 7 arguments, even though `\@sect` is defined to take 8 arguments. And then you realize that `\section` was defined to take no arguments of its own at all! What's happening? Why isn't `\section` defined to take some arguments, like this:

```
\newcommand\section[1]
{...}
```

since `\section` is always called with a name argument, as in `\section{Thanks}`?

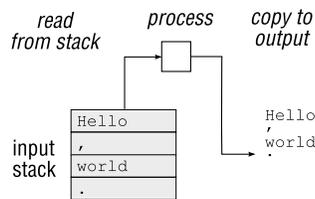


Figure 4: Macro processor — output processing

This starts to make sense only when you realize that plain TeX behaves as a classical, stack-oriented, macro processor (which also typesets!). Initially you can consider the input stack to contain the whole input file. The processor reads input from the file, i.e. removes it from the stack. It just copies the input to the output, unless it's either a macro definition, or a macro invocation, in which case it's evaluated and the result is pushed back onto the input stack, to be re-processed. To make this concrete, let's look at a few simple examples for the `m4` macro processor. The following input has no macros or anything else special, so it's copied to the output without change:

```
% echo 'Hello, world.' | m4
Hello, world.
```

as shown in Figure 4. The slightly more complex:

```
define('showarg', 'my arg is $1')
A showarg(mouse) A
```

defines a simple macro that takes a single argument. Run it and see what you get:

```
% m4 ex1.m4
A my arg is mouse A
```

Now let's have one macro reference another indirectly:

```
define('concat', '$1$2')
define('showarg', 'my arg is $1')
B concat(quick, brown) B
C showarg(fox) C
D concat(sho, warg)(jumps) D
```

and run this:

```
% m4 example.m4
B quickbrown B
C my arg is fox C
D my arg is jumps D
```

The B and C lines are straightforward, but line D is tricky: `concat(sho, warg)` is read from the stack, leaving only:

```
(jumps) D
```

But what we've just read — `concat(sho, warg)` — evaluates to `showarg`, so the string `{showarg}` is

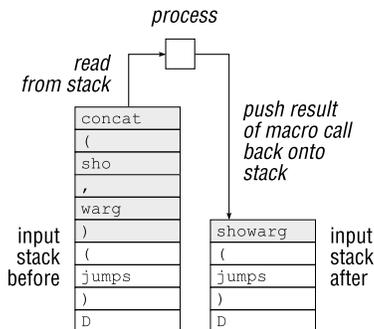


Figure 5: Macro result pushed back onto stack

pushed back onto the stack (Figure 5). The top of the stack now looks like:

```
showarg(jumps) D
```

which is re-evaluated as a call to `showarg` with argument `jumps`. In other words, `(jumps)` was left lying on the stack, and it was picked up as an argument to a macro in due course.

The same thing happens in \LaTeX . In the following code:

```
Foo \textbf{cat} bar. (A)
\newcommand{\Ttwo}[2]
  {\csname text#1 \endcsname{#2}}
Foo \Ttwo{bf}{cat} bar. (B)
\newcommand{\Tone}[1]
  {\csname text#1 \endcsname}
Foo \Tone{bf}{cat} bar. (C)
Foo \newcommand{\Tzero}{\textbf}
  \csname \Tzero \endcsname{cat} bar. (D)
```

each line produces the same output:

```
Foo cat bar. (A)
Foo cat bar. (B)
Foo cat bar. (C)
Foo cat bar. (D)
```

However, in lines (C) and (D) the string `{cat}` is not specified as an argument to the `Txxx` macro we defined — it's left lying around, conveniently surrounded by braces, and is picked up later.

This technique is used in the sectioning code. When you write `\section{Thanks}`, the macro `\section` is invoked with no arguments, leaving the string `{Thanks}` on the stack. `\section` calls `\@startsection`, which calls `\@sect` with 7 arguments; but `\@sect` needs 8 arguments, so it picks up `{Thanks}` from the top of the stack, so it's happy, and we don't get any errors.

12 The results, and lessons learned

Our original \LaTeX 2.09 style file had 1400 lines of code, excluding comments.

Our new \LaTeX 2 ϵ style file has 300 lines of code, excluding comments: 34 are `\RequirePackages`, 150 lines make up 54 `\newcommands` for convenience-type functions that we ought to have isolated in separate files had we been disciplined enough, and 15 lines make up 11 `\newenvironments`. The other large chunk of code is 35 lines for our modified version of `\@sect`.

Not only is our new style file much shorter, it's easier to understand and maintain, and is much more flexible than our old one. We can use most standard packages to add extra features with no effort, because we still provide all the hooks that add-on packages rely on. Moreover, the standard class and style files have had years of debugging and are very robust and reliable. By re-using as much as possible, and minimizing the amount of code changed, we've ended up with a stable system. We've had almost no bugs, and the ones we did have we were able to fix quickly and cleanly.

13 Thanks

Lots of people very kindly helped us over the years, with advice and pieces of code. These include, but are not limited to: Donald Arseneau, Barbara Beeton, Timothy Van Zandt, and lots of other patient and helpful people on the `texhax@tug.org` mailing list.

References

- [1] Niall Mansfield. *The Joy of X*. Addison-Wesley, 1986.
- [2] Niall Mansfield. *Practical TCP/IP — Designing, using, and troubleshooting TCP/IP networks on Linux and Windows (first edition)*. Addison-Wesley, 2003.
- [3] Niall Mansfield. *Practical TCP/IP — Designing, using, and troubleshooting TCP/IP networks on Linux and Windows (second edition)*. UIT Cambridge Ltd., 2008.
- [4] Jan-Piet Mens. *Alternative DNS Servers — Choice and deployment, and optional SQL/LDAP back ends*. UIT Cambridge Ltd., 2008.
- [5] J.R. Tracey, D.E. Rugh, and W.S. Starkey. *STOP: Sequential Thematic Organization of Publications*. Hughes Aircraft Corporation, Fullerton, CA, 1965.
- [6] Edmond H. Weiss. *How to Write a Usable User Manual*. ISI Press, 1985.

TeXworks: Lowering the barrier to entry

Jonathan Kew
21 Ireton Court
Thame OX9 3EB
England
jonathan@jfkew.plus.com

1 Introduction

One of the most successful TeX interfaces in recent years has been Dick Koch’s award-winning TeXShop on Mac OS X. I believe a large part of its success has been due to its relative simplicity, which has invited new users to begin working with the system without baffling them with options or cluttering their screen with controls and buttons they don’t understand. Experienced users may prefer environments such as iTeXMac, AUCTeX (or on other platforms, WinEDT, Kile, TeXmaker, or many others), with more advanced editing features and project management, but the simplicity of the TeXShop model has much to recommend it for the new or occasional user.

Besides the relatively “clean” interface, a second factor in TeXShop’s success is probably the use of a PDF-centric workflow, with pdfTeX as the default typesetting engine. PDF is the de facto standard for fully-formatted pages; every user knows what a PDF file is and what they can do with it. Bypassing DVI reduces the apparent complexity of the overall process, and so reduces the “intimidation factor” for a newcomer. But TeXShop is built on Mac OS X-specific technologies, and is available only to Mac users. There does not seem to be an equivalent tool available on other platforms; there are many TeX editors and environments, but none with this particular focus.

The TeXworks project is an effort to build a similar TeX environment (“front end”) that will be available for all today’s major desktop operating systems—in particular, MS Windows (XP and Vista), typical GNU/Linux distributions, and other X11-based systems, in addition to Mac OS X.

To achieve this, TeXworks is based on cross-platform, free and open source tools and libraries. In particular, the Qt toolkit was chosen for the quality of its cross-platform user interface capabilities, with native “look and feel” for each platform being a realistic target. Qt also provides a rich application framework, facilitating the relatively rapid development of a usable product.

The standard TeXworks workflow will also be PDF-centric, using pdfTeX and XeTeX as typesetting engines and generating PDF documents as the default formatted output. Although it will still be possible to configure a processing path based on DVI, newcomers to the TeX world need not be concerned with DVI at all, but can generally treat TeX as a system that goes directly from marked-up text files to ready-to-use PDF documents.

TeXworks includes an integrated PDF viewer, based on the Poppler library, so there is no need to switch to an external program such as Acrobat, xpdf, etc., to view the typeset output. The integrated viewer also allows it to support source ↔ preview synchronization (e.g., control-click within the source text to locate the corresponding position in the PDF, and vice versa). This capability is based on the “SyncTeX” feature developed by Jérôme Laurens, now integrated into the XeTeX and pdfTeX engines in TeX Live 2008, MiKTeX, and other current distributions.

2 Features for initial release

Figure 1 shows the current TeXworks prototype running on Windows Vista. While this is not a finished interface, it gives an impression of how the application will look. TeXworks version 0 will be an easy-to-install application offering:

1. Simple (non-intimidating — this is *not* emacs or vi!) GUI text editor with
 - i. Unicode support using standard OpenType fonts
 - ii. multi-level undo/redo
 - iii. search & replace, with (optional) regular expressions as well as simple string match
 - iv. comment/uncomment lines, etc.
 - v. (L^A)TeX syntax coloring
 - vi. TeX-aware spell checker
 - vii. auto-completion for easy insertion of common commands
 - viii. templates to provide a starting point for common document types

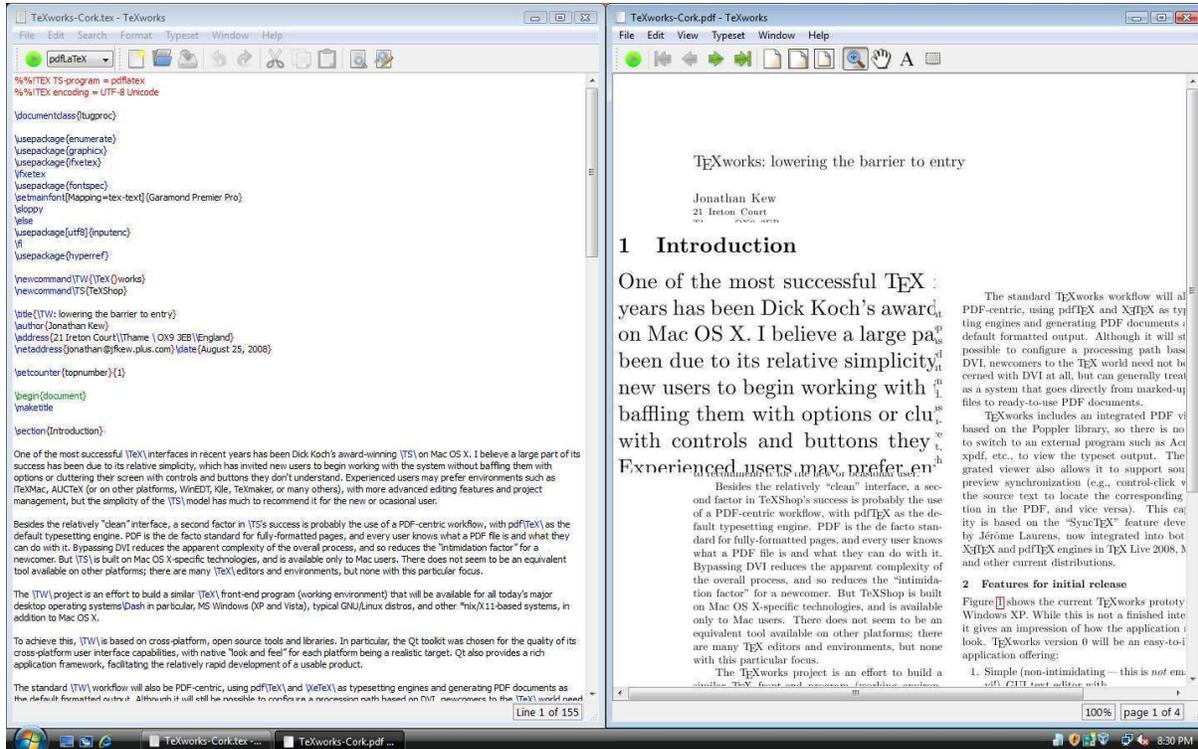


Figure 1: A recent TeXworks build running on Windows Vista: source and preview windows, with the TeXShop-style magnifying glass in use.

2. Ability to run TeX on the current document to generate PDF:
 - i. extensible set of TeX commands (with common commands such as `pdftex`, `pdflatex`, `xelatex`, `context`, etc. preconfigured)
 - ii. also support running BibTeX, Makeindex, etc.
 - iii. any terminal output appears in a “console” panel of the document window; automatically hidden if no errors occur
 - iv. “root document” metadata so “Typeset” works from an `\included` file
3. Preview window to view the output:
 - i. anti-aliased PDF display
 - ii. automatically opens when TeX finishes
 - iii. auto-refresh when re-typesetting (stay at same page/view)
 - iv. TeXShop-like “magnifying glass” feature to examine detail in the preview
 - v. one-click re-typesetting from either source or preview
 - vi. source ↔ preview synchronization based on Jérôme Laurens’ SyncTeX technology

3 Current status

An early TeXworks prototype was demonstrated at the BachTeX conference (April 2008). It became more widely available (though still considered a prototype, not a finished release) when a version was posted in mid-July before the TUG 2008 conference. The current code is available as source (easy to build on typical GNU/Linux systems), and as precompiled binaries for Windows and Mac OS X.

At this time, the application supports text editing and PDF viewer windows, and has the ability to run a typesetting job and refresh the output view, etc. There is not yet any documentation, and many potential “power user” features do not yet exist, but it is a usable tool in its current state. In addition to Windows (XP and Vista), it runs on Mac OS X (see figure 2) and GNU/Linux systems (figure 3).

A few features remain to be implemented before a formal release of “version 0”, including “single instance” behavior, and various options for window positioning; appropriate installer packages for Mac OS X and Windows are also needed, to simplify setup.

More information may be found online via the TeXworks home page at <http://texworks.org/>.

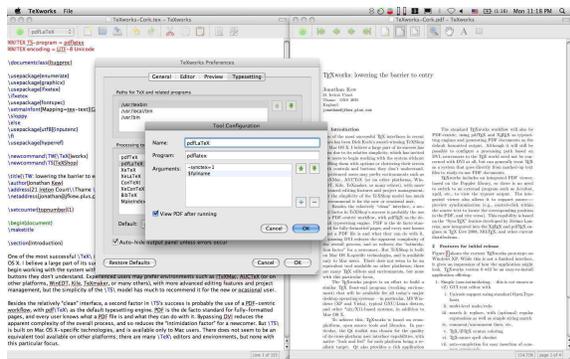


Figure 2: TeXworks running on Mac OS X: using the Preferences dialog to configure a typesetting tool.

4 Future plans

After the release of version 0, several major additional features are planned; some ideas high on the priority list include:

- intelligent handling of TeX errors
- assistance with graphics inclusion and format conversions
- text search and copy in the PDF preview
- support rich PDF features such as transitions, embedded media (sound, video), annotations, etc.
- customizable palettes of symbols, commands, etc.
- TeX documentation lookup/browser
- interaction with external editors and other tools
- additional support for navigating in the source, e.g., “folding” sections of text, recognizing document structure tags such as `\section`, etc.

I expect development priorities to be guided by user feedback as well as developer interest, once the initial version 0 release is available.

5 Invitation to participate

TeXworks is a free and open source software project, and all are welcome to participate and contribute to its development. This does not necessarily mean writing code; many other roles are equally important. Some possible ways to participate include the following.

- Use the prototype for some real work, and give feedback on what’s good, what’s bad, what’s broken:
 - if there’s a current binary download available for your platform, try that;
 - get the code and try building it on your platform; provide bug reports (and fixes!) for whatever problems show up.

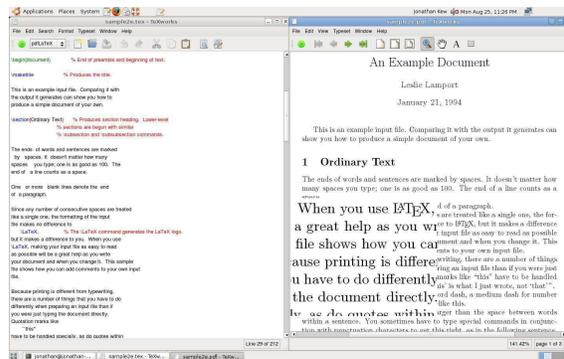


Figure 3: TeXworks running on a typical GNU/Linux system (Ubuntu).

- Dig in to the code, and submit patches to implement your favorite missing features.
- Write on-line help, documentation and tutorials for newcomers to TeXworks and TeX.
- Review and enhance the command completion lists available for the integrated editor.
- Provide well-commented templates for various types of documents.
- Design icons for the toolbars, etc.; TeXworks has some nice icons from Qt and the Tango project, but others are merely rough placeholders.
- Use the Qt Linguist tool to localize the user interface for your language.
- Package TeXworks appropriately for your favorite GNU/Linux or BSD distribution, or create an installer for Windows or Mac OS X.

There is a TeXworks mailing list for questions and discussions related to the project; see <http://lists.tug.org/texworks/> to subscribe, for the list archives, etc.

The TeXworks source itself is maintained in a Google Code project at <http://code.google.com/p/texworks/>. Resources available through this site include the Subversion source repository, precompiled binaries for Windows and Mac OS X, and an issue tracker for bug reports and feature suggestions.

6 Thanks

The TeXworks project arose out of discussions at several recent TeX Users Group meetings, and has received generous support from TUG’s TeX development fund and its contributors, and from UK-TUG. Special thanks to Karl Berry for his encouragement and support, and to Dick Koch for showing us the potential of a clean, simple TeX environment for the average user.

Direct and reverse synchronization with SyncTeX

Jérôme Laurens

Département de mathématiques

Université de Bourgogne

21078 Dijon Cedex

FRANCE

jerome (dot) laurens (at) u-bourgogne (dot) fr

<http://itexmac.sourceforge.net/SyncTeX.html>

Abstract

We present a new technology named SyncTeX used to synchronize between the TeX input and the output.

1 What is synchronization?

Creating documents with the TeX typesetting system often requires two windows, one for entering the text, the other one for viewing the resulting output. In general, documents are too long to fit in the visible frame of a window on screen, and what is really visible is only some part of either the input or the output. We say that the input view and the output view are synchronized if they are displaying the “same” portion of the document. Forwards or direct synchronization is the ability, for an output viewer, to scroll the window to a part corresponding to a given location in the input file. Backwards or reverse synchronization is the ability, for a text editor, to scroll the text view to a part corresponding to a given location in the output file.

Figure 1 is a screenshot illustrating SyncTeX supported in *iTeXMac2*, the TeX front end developed by the author on Mac OS X. The top window is a text editor where an extract of the “Not so short introduction to L^AT_EX 2_ε” is displayed. The word “lscommand” has been selected and the viewer window at the bottom automatically scrolled to the position of this word in the output, highlighting it with a small red arrow. The grey background was added afterwards in the bottom window for the sake of visibility on printed media.

2 What is SyncTeX?

This is a new technology embedded in both pdfTeX and XeTeX, available in the 2008 TeX Live and corresponding MiKTeX distributions. When activated, it gives both text editors and output viewers the necessary information to complete the synchronization process. It will be available in LuaTeX soon.

In order to activate SyncTeX, there is a new command line option:

```
pdftex -synctex=1 foo.tex
```

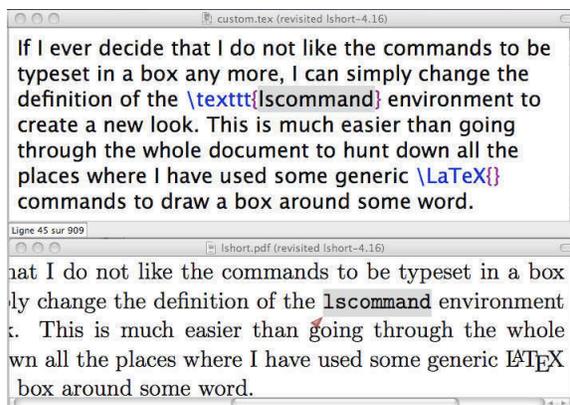


Figure 1: Synchronization in *iTeXMac2* based on SyncTeX technology with text analysis.

(or `--synctex=1`) and the same for `xetex`. With this option, a new informational line is printed at the end of the command output and reads `SyncTeX written on foo.synctex.gz`. The new (compressed) auxiliary file named `foo.synctex.gz` is used by applications for the synchronization process; this is the *SyncTeX output file*.

Setting the `synctex` option to `-1` creates an uncompressed `foo.synctex` auxiliary file, more suitable for certain operations. Setting it to `0` definitively prevents SyncTeX activation.

There is also an eponymous new TeX primitive that you can set to 1 for SyncTeX activation from the source file: `\synctex=1`. It can be used, for example, to temporarily disable SyncTeX operations for some input file by properly using `\synctex=0`. This primitive has no effect if the `-synctex=0` command line option was given, or if the first page has already been shipped out (it is then too late to activate SyncTeX).

3 Other synchronization technologies

The commercial software *Visual T_EX* available on Windows has had the PDF synchronization capability embedded in its T_EX engine since 1999. The commercial software *T_EXtures* available on Mac OS X has had embedded synchronization since 2000, but between the text source and the DVI output. Neither implementation is freely available to the public and will not be considered in the remainder of this article.

Turning to the T_EX macro level, Aleksander Simonic, the author of the *WinEdt* T_EX shell on Windows, wrote before 1998 the `srcltx` macro package to enable synchronization between the text source and the DVI output. It is based on the powerful `\special` command and was later integrated into the T_EX engine as “source specials”. Heiko Oberdiek wrote `vpe` in 1999 where PDF technologies are used for reverse synchronization from the PDF output to the text input. In 2003, the author wrote the `pdfsync` package discussed in [5], [6] and [7], to allow synchronization between the PDF output and the text input, following ideas from Piero d’Ancona. This was based on the use of pdfT_EX commands similar to `\special`, with the same limitations, namely an incompatibility with very useful packages and unwanted changes in the layout.

None of these solutions is satisfying, being either incomplete or unsafe, as we shall see.

4 Solving the synchronization problem

4.1 Stating the problem

The problem is to define a mapping between an *input record* given by an input file name and a line number in that file, and an *output record* given by a page number and a location in that page of the output file. The input record describes a certain part of the input text whereas the output record describes the corresponding location where this text appears in the output. The original T_EX engine does not provide any facility for such a correspondence, except the debugging information used to report syntax errors (we call it the *current input record*). More precisely, T_EX does not know at the same time both the input records and their corresponding output records. In short, T_EX parses each different line of the input text file and expands the macros with its “eyes” and “mouth” (according to [2], page 38), then it sends a list of tokens into its “stomach”. In turn, the stomach creates lines of text, stacks them into paragraphs, and stacks the paragraphs into pages. Once a page is complete with the objects properly positioned, it is shipped out to the output file. During this process, T_EX keeps the input record information until macro

expansion only (in its head), and it does not know the corresponding output record until ship out time which occurs later (in its stomach). The problem is to force T_EX to remember the input record information until ship out time.

4.2 Partial solutions using macros

The first idea, developed in the `srcltx` package, is to use the `\special` macro to keep track of the input record information until ship out time. By this method, it inserts in the text flow invisible material that dedicated DVI viewers can understand. The main problem is that this invisible material is not expected to be there and can alter significantly the line breaking mechanism or cause other packages to malfunction, which is extremely troublesome.

The second idea, developed in the `pdfsync` package, is also to use macros, but in a different way because it is more difficult to manage PDF contents than DVI contents. This package automatically adds in the input source some macros that act in two steps. At macro expansion time, they write to an auxiliary file the input record information with a unique identifying tag. They also insert in the text flow invisible material to prepare T_EX to write the output record information at ship out time, with exactly the same identifying tag. In this design, the problems concerning line breaking and package incompatibility remain. Moreover, the mapping between input and output records is not one to one, which renders synchronization support very hard to implement for developers.

In these two different solutions, we see the inherent limits of synchronization using macros. More generally, we can say that those macros are *active observers* of the input records. In fact, by inserting invisible material in the text flow they interact with the typesetting process. On the contrary, SyncT_EX is a *neutral observer* that never interacts with the typesetting process.

4.3 How SyncT_EX works

In fact, the only object that ever knows both the input and output records is the T_EX engine itself, so it seems natural to embed some synchronization technology into it.

We first have to determine what kind of information is needed to achieve synchronization. For that purpose, we follow [2] at page 63: “T_EX makes complicated pages by starting with simple individual characters and putting them together in larger units, and putting these together in still larger units, and so on. Conceptually, it’s a big paste-up job.

The TeXnical terms used to describe such page construction are boxes and glue.” The key words are “characters”, “boxes” and “glue”. But since an individual character requires a considerable amount of extra memory, only horizontal boxes and vertical boxes are taken into account at first. For these boxes, we ask TeX to store in memory, at creation time and during their entire lifetime, the current input record. At ship out time, we ask TeX to report for each box the stored file name, the stored line number, the horizontal and vertical positions, and the dimensions as computed during typesetting. This information will be available for synchronization: for example when the user clicks on some location of the PDF document, we can really find out in which box the click occurred, and then deduce the corresponding file name and line number. We have here the design for a neutral observer.

But if this new information is sufficient for locating, it cannot be used for synchronization due to the way TeX processes files. In fact, boxes can be created in TeX’s mouth where the current input record is accurate, but in general, they are created in the stomach when breaking lines into paragraphs, for text that was parsed a long time ago and no longer corresponds to the current input record. This is particularly obvious when a paragraph spans many lines of the input text: the line breaking mechanism is engaged after the last line is parsed, and every horizontal box then created will refer to the last input line number even if the contained material comes from a previous input line. For that reason, we also ask TeX to store input records for glue items, because they are created in TeX’s mouth, when the current input record is still accurate.

By combining boxes and glue management, we have accurate information about positions in the output and the correspondence with positions in the input file. In fact, things are slightly more complicated because of TeX internals: kern, glue and math nodes are more or less equivalent at the engine level, so SyncTeX must treat them similarly, but this is better for synchronization due to the supplemental information provided.

SyncTeX does other sorts of magic concerning file name management, the magnification and offset, but these are implementation details.

4.4 The benefits of SyncTeX

Embedding the synchronization technology deeply inside the TeX engine solves many problems and improves the feature significantly.

The most visible improvements are connected with accuracy: with SyncTeX, the synchronization

process reaches in general a precision of ± 1 line. With additional technologies such as implemented in *iTeXMac2*, we can even synchronize by words (see figure 1), essentially always finding an exact word correspondence between input and output.

The next improvements are a consequence of the overall design of SyncTeX. Since synchronization is deeply embedded into the TeX engines, there is no TeX macro involved in the process. As a straightforward consequence, there cannot be any incompatibility with macro packages. Moreover, no extra invisible material is added to the text flow, thus ensuring that the layout of the documents is exactly the same whether SyncTeX is activated or not. As a matter of fact, it is absolutely impossible to determine if the output was created with SyncTeX activated by examining its contents. Finally, no assumptions are made about external macros or output format, so that synchronization works the same for Plain, L^ATeX or ConTeXt as well as DVI, XDV or PDF.

Of course, all this needs extra memory and computational time but this is in no way significant. *In fine*, we can say that with SyncTeX, the synchronization has become safe and more precise.

5 Limits and improvements

It is indisputable that abandoning the use of macros and choosing an embedded design is a great advance for synchronization. But still it is not perfect! Some aspects of the implementation are not complete due to a lack of time, but others will prevent us from reaching the ultimate synchronization comparable to *wysiwyg* (an acronym for “What You See Is What You Get”) as discussed in [7].

5.1 The DVI to PDF filters

When producing a PDF document from a DVI or XDV output, we apply a filter like `dvitopdf` or `xdv2pdfmx`. But those filters can alter the geometry of the output by changing the magnification or the offset of the top left corner of the text. In that case, the SyncTeX information, which is accurate for the DVI file, is not accurate for the PDF file. This problem is solvable by post-processing the SyncTeX output file with the new `synctex` command line tool available in the distributions, *eg*

```
xdv2pdfmx -m MAG -x XXX -y YYY foo
```

should be followed by

```
synctex update -m MAG -x XXX -y YYY foo
```

But this is not a good design. Instead, the post-processing should be embedded into the various DVI to PDF filters so that no further user interaction is required.

5.2 Using \TeX 's friends

Some documents are created with a complex typesetting flow where \TeX is just one tool amongst others. In such circumstances, the \TeX file is automatically generated from possibly many original input files by some processor. Then, synchronization should occur between the output and the original input files rather than the \TeX file, which is just an intermediate step. For example, a bibliography in \LaTeX is generally built by $\text{BIB}\TeX$ based on a bibliography database with a `bib` file extension using an intermediate auxiliary \LaTeX file with a `bb1` file extension. At present, the synchronization occurs between the PDF output and the `bb1` file and not the original `bib` file, as one would prefer.

Improving $\text{Sync}\TeX$ to properly manage this situation is not extremely complicated: we first have to define a $\text{Sync}\TeX$ map file format for the mapping between the lines of the original input files and the lines of the produced \TeX files, then we have to provide facilities to merge this mapping information into the $\text{Sync}\TeX$ output file. Then the processor could produce the map file, and a supplemental step in the typesetting flow would update the $\text{Sync}\TeX$ information with that map.

Sometimes it might not be appropriate to simply bypass the intermediate file. In that case, the viewer should synchronize with the auxiliary file using a text editor which in turn should synchronize with the original input file using the map file.

5.3 Accuracy and the column number

As described above, we only take into account whole lines in the input files and jump from or to lines in the text. This can suffice for textual files, but does not when mathematical formulas are involved — we would like to have a more precise position in the input. Unfortunately, when parsing the input files, the original \TeX engine does not handle column positions at all. And it seems that adding support for this supplemental information might need a great amount of work, probably much greater than the eventual benefits.

5.4 For non-Latin languages

$\text{Sync}\TeX$ has been designed with a Latin language background: it relies on the fact that \TeX automatically creates kern and glue nodes at parse time to manage interword spacing. For languages that do not have a comparable notion of word, the synchronization will not be sufficiently accurate and will most certainly need further investigations. This question is

open and the author welcomes test files, suggestions and advice.

5.5 A question of design

The two preceding limitations are consequences of a conceptual default in the actual synchronization design. With $\text{Sync}\TeX$, the \TeX engine has been modified to export some observed information useful for synchronizers. The problem is that we are able to observe only what \TeX allows us to, and this is not always the best information we would like to have. It would be more comfortable and efficient if \TeX already provided synchronization facilities from the very beginning. In that case, all the macros packages would have to be compatible with the synchronization code and not the opposite. That would require more work for the package maintainers but would also prevent any kind of layout and compatibility problems due to special nodes.

In a different approach, a supplemental step could be to store synchronization information for each individual character, thus increasing the memory requirements of the engine in a way similar to how $\text{X}\TeX$ handles multi-byte characters. This idea was originally proposed by Hàn Thế Thành, but it was abandoned because the $\text{Sync}\TeX$ output file was unbearably huge. With the new design, this idea can certainly be revisited with more success.

Anyway, further investigations into the arcana of the \TeX program would certainly lead to a better synchronization accuracy but if we want to avoid huge changes in \TeX and keep compatibility with existing macro packages, we must admit that we have almost reached some insuperable barrier.

6 Implementation in \TeX Live

Without entering into great detail, we explain how the implementation of $\text{Sync}\TeX$ is carefully designed to ease code maintenance and enhancements, as far as possible.

6.1 A segmented implementation

All the $\text{Sync}\TeX$ related code is gathered in only one directory named `synctexdir`, in which the code is split into different source files. The separation is organized in order to share the maximum amount of code between the different engines, and to clearly identify the different tasks involved in the information management. All in all, we end up with 14 different change files. When building the binaries, the partial make file `synctex.mk` has the duty to manage which change file should apply to which engine and when.

6.2 An orthogonal implementation

One of the key concepts in modern code design is separation, whose purpose is to ease code management and maintenance. WEB Pascal does not offer facilities for code separation, nevertheless, it is possible to build all the engines with or without the SyncTeX feature, as explained in the `synctex.mk` file. It will be useful for developers whenever a problem is caused by the SyncTeX patches.

7 Which software supports SyncTeX

Up to now, we have focused on the technological aspects of synchronization, and we have described in detail the foundations. It is time to look at the concrete implementations of synchronization with different methods, because this feature is useless if it is not adopted by applications. SyncTeX not only consists of changes to the TeX engine, but gives developers tools to easily support the technology.

7.1 The SyncTeX parser library

The main tool is a library written in C, whose purpose is to help developers implement direct and reverse synchronization in their application. It consists of one file named `synctex_parser.c` and its header counterpart `synctex_parser.h`, meant to be included as-is in application sources. Both are available on the SyncTeX web site [4]. The source file takes care of all the ancillary work concerning SyncTeX information management and the header file contains all the necessary help for an optimal usage of the programming interface.

At this writing, *TeXworks* (presented by J. Kew in [1]), Sumatra PDF on the Windows platform, and Skim and *iTeXMac2* on Mac OS X, all support SyncTeX by including this parser library. For other applications, TeX users are encouraged to send a feature request to the developers of their favorite PDF or DVI viewer.

7.2 Remark about the document viewer

It should be noticed that the tricky part of direct and reverse synchronization should be handled by the viewer only. The SyncTeX parser library is meant not for text editors but for viewers. In a normal direct synchronization flow, the user asks the text editor to synchronize the viewer with a given line in a given input file, the text editor forwards the file name and the line number to the viewer, the viewer asks the SyncTeX parser for the page number and location corresponding to the information it has received, then it scrolls its view to the expected page and highlights the location. In a normal reverse

synchronization flow, the user asks the viewer to synchronize the text editor with a given location in a given page of an output file, the viewer asks the SyncTeX parser for the input file name and line number corresponding to the location; it then asks the text editor to display the named input file and highlight the numbered line.

7.3 The new `synctex` command line tool

There are cases when the inclusion of the parser library is not possible or even improbable (consider for example Adobe's Acrobat reader). For such situations, the `synctex` command line tool is the alternative designed to allow synchronization. It is just a wrapper over the SyncTeX parser library that acts as an intermediate controller between a text editor and a viewer. The description of its usage is obtained via the command line interface running `synctex help view` for direct synchronization and `synctex help edit` for reverse synchronization.

Provided that the text editor and the viewer have some scripting facilities, here is how this tool should be used. For direct synchronization, the user asks the text editor to synchronize the viewer with a given line in a given input file, the text editor forwards this file name and line number to the `synctex` tool together with some scripting command to activate the viewer, the `synctex` tool transparently asks the SyncTeX parser for the page number and location corresponding to the information it has received, then it asks the viewer to proceed with the help of the scripting command.

For reverse synchronization, the user asks the viewer to synchronize the text editor with a given location in a given page of an output file, the viewer forwards this information to the `synctex` tool together with some scripting command to activate the text editor, the `synctex` tool transparently asks the SyncTeX parser for the input file name and line number corresponding to the information it has received, then it asks the text editor to proceed according to the received scripting command.

Before this tool was available, developers had no solution other than directly parsing the contents of the SyncTeX output file. This was generally made in continuation of the implementation of `pdfsync` support. Comparatively, it is more comfortable to work with a `.synctex` file than a `.pdfsync` file because the new syntax is extremely clear and straightforward, consequently reverse engineering was unexpectedly encouraged. But this practice should be abandoned for two reasons: it is certainly not compatible with forthcoming enhancements of SyncTeX, and it generally does not work when changing the magnification

and the offset in the DVI to PDF conversion, as discussed above. In order to convince developers to prefer the `syncTeX` tool, the specifications of the SyncTeX output file are considered private and will not be widely published.

More details concerning usage and implementation are available on the SyncTeX web site [4].

8 Applications

There are a variety of ways to use the newly available information in the SyncTeX output file. Some were considered while designing this feature, others suggested by people at the conference. No doubt this list is not exhaustive.

8.1 Better typesetting mechanisms

TeX is well known for its high quality page breaking mechanism, but the hardware constraints that were crucial 30 years ago imposed some choices and deliberate barriers. The limitation in memory usage led to a page by page design, where memory is freed each time a page is shipped out. In that situation, a page breaking algorithm cannot perform optimization in a document as a whole, but only on a small number of consecutive pages.

In order to have global optimization algorithms, one can keep everything in memory until the end of the TeX run, but that would require a big change in the engine. From another standpoint, SyncTeX has demonstrated that it is possible to trace geometrical information throughout the typesetting process. It is clear that the information actually contained in the SyncTeX output file is not suitable for typesetting purpose because it was designed for synchronization only. But with some additional adaptations, there is no doubt that SyncTeX can help in designing global optimization algorithms for even better typesetting.

8.2 Debugging facilities

During his presentation at the conference (see [3]), the author used a lightweight PDF viewer to demonstrate SyncTeX. This viewer was primarily designed as a proof of concept and as such, was meant to remain private. But one of its features might be of great interest to the TeX community, as suggested by different people at the conference, namely the ability to display over the text all the boxes, either horizontal or vertical, created during the typesetting process. As it happens, this feature was already implemented in an unknown modest PDF viewer for Mac OS X (whose name I have unfortunately lost) by parsing the result of the `\tracingall` macro in the log file.

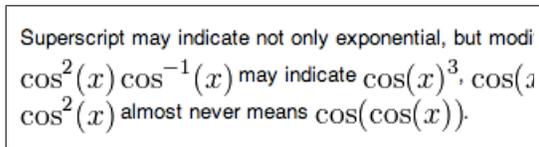


Figure 2: TeX output embedded in HTML, detail of http://en.wikipedia.org/wiki/Special_functions (2008/08/11)

The interest is at least twofold. It can serve debugging purposes for publishers who want to elaborate complicated page layouts, and it can also serve pedagogical purposes during TeX training sessions. For these reasons, this viewer will be available on the SyncTeX web site [4] once it has been properly factored for distribution. Unfortunately, this benefits Mac OS X users only, but adding this feature to the new cross-platform *TeXworks* will eventually be considered.

8.3 Embedding TeX output into HTML or running text

In web pages, it is rather common to include mathematical formulas as embedded images built with TeX, to compensate for the limitations of web browsers. The example given in figure 2 is particularly ugly, not only because the size of the mathematical text does not conform to the size of the running text, but also because the base lines of the formulas and the running text are not properly aligned. In fact, the included images contain no information concerning the base line, and this is where SyncTeX can come into play. The synchronization information contains the dimensions of each box containing a mathematical formula, in particular its height and depth, hence the exact position of the base line. We just have to raise the image by the proper amount to obtain a correct vertical alignment.

9 Concluding remarks

9.1 Synchronizing by word

In *iTeXMac2*, synchronization is enhanced to attain the precision of a word or even a character, by combining SyncTeX with some text analysis. This was rather easy to accomplish because *iTeXMac2* manages both the text input and the PDF output, and also because the PDF library on Mac OS X has text facilities. But this does not mean that only an integrated TeX environment is able to reach such a level of accuracy. It is in fact more a matter of communication between different applications.

In fact, the text editors and the viewers allow some inter-process communication through the command line, for example `mate -l LINE foo.tex` asks the *TextMate* text editor to edit `foo.tex` at line `LINE`. Only the line number is used and this is perfectly suitable for programmers because in general, compilers and debuggers only need line numbers. But TeX users are not programmers. In *iTeXMac2* this kind of practice has been reconsidered for reverse synchronization and the information passed to the text editor contains not only the file name and the line number, but also an extract of text surrounding the character under the mouse. This supplemental information is the hint used by the text editor to have a better focus on the synchronization target. For direct synchronization, the same idea applies and the PDF viewer is asked to highlight a location in a given page, with the help of a similar textual hint.

In order to achieve synchronization by word or character, text editors and viewers should use a textual hint, both as senders and receivers. Of course this requires some coding effort because input text and output text are not exactly the same (due to line breaking for example), but the expense is affordable as soon as efforts are combined. Once again, users are encouraged to submit feature requests to the developers of their favorite tools.

By the way, the new `synctex` command line tool anticipates the use of a textual hint by editors or viewers through its `-h` command line option.

9.2 An historical standpoint

Synchronization with SyncTeX appears for the 30th anniversary of TeX; we can legitimately wonder why and whether such a long period of gestation was necessary. In order to explain this delay, let us review the ingredients that made SyncTeX possible.

As in many situations of software design, a favorable context comes concurrently from available technologies and available workers. Regarding technological aspects, we can say in a reduction not very far from reality that SyncTeX is nothing but a clever usage of the Web2C implementation of TeX. Of course, developing on MacOS X was rather easy and very efficient, but any other environment would certainly provide the same result at the price of more programming work.

Concerning people, the author has claimed since the beginning of `pdfsync` that some synchronization should definitely be embedded in the TeX engine, in the hope that someday, someone *else* would do it. Hàn Thế Thành was aware of the problem three years

ago (not one year ago as claimed by the author during his presentation [3]), but he could only take some time for coding this in summer 2007, probably under the friendly pressure of some users dissatisfied with the limits of `pdfsync`. Although his first attempt was hardly usable and finally abandoned, it introduced the author to the minutiae of the Web2c implementation of TeX. Initially, SyncTeX was targeted at pdfTeX but Jonathan Kew helped in adapting it to XeTeX and also with the integration into TeX Live.

This short review seems to indicate that technologies like SyncTeX could easily have become available many years ago. One can attribute the delay to a lack of effort devoted to the human interface of TeX, which is highly regrettable. With SyncTeX and tools like *TeXworks*, first steps are made in the right direction, because TeX really deserves a good human interface, not just a user interface.

10 Acknowledgements

The author gratefully thanks Hàn Thế Thành without whom this work would never have started and Jonathan Kew without whom this work would not have reached the present stage. He received important remarks and valuable help from members of the pdfTeX, XeTeX, *iTeXMac2* and TeX Live development teams; thanks to all of them.

References

- [1] Jonathan Kew. *TeXworks*: Lowering the barrier to entry. In this volume, pages 362–364.
- [2] D. Knuth. *The TeXbook*. Addison Wesley, 1983.
- [3] Jérôme Laurens. *SyncTeX presentation at TUG 2008*. <http://www.river-valley.tv/conferences/tug2008/#0302-Jerome-Laurens>.
- [4] Jérôme Laurens. *SyncTeX web site*. <http://itexmac.sourceforge.net/SyncTeX.html>.
- [5] Jérôme Laurens. *iTeXMac*, an integrated TeX environment for Mac OS X. In *TeX, XML, and Digital Typography*, volume 3130/2004 of *Lecture Notes in Computer Science*, pages 192–202. Springer Berlin / Heidelberg, 2004.
- [6] Jérôme Laurens. The TeX wrapper structure: A basic TeX document model implemented in *iTeXMac*. In *EuroTeX 2005, 15th Annual Meeting of European TeX Users*, 2005.
- [7] Jérôme Laurens. Will TeX ever be *wysiwyg* or the PDF synchronization story. *The PracTeX Journal* 2007(3), 2007.

Xindy revisited: Multi-lingual index creation for the UTF-8 age

Joachim Schrod

Net & Publication Consultance GmbH

Kranichweg 1

63322 Rödermark, Germany

jschrod (at) acm dot org

<http://www.xindy.org/>

Abstract

Xindy is a flexible index processor for multi-lingual index creation. It handles 44 languages with several variants out of the box. In addition, some indexes demand special sort orders for names, locations or different target audiences; xindy can handle them as well. Raw index files may have several encodings beyond ASCII. In particular, L^AT_EX's standard output encoding is supported directly, as is X_YL^AT_EX's UTF-8 output. With the new *xindy TUG30 release*, support for Windows is added; previously xindy was available only for GNU/Linux, Mac OS X, and other Unix-like systems.

1 What is xindy?

Xindy is an index processor. Just like MakeIndex, it transforms raw index information into a sorted index, made available as document text with markup that may be processed by T_EX to produce typeset book indexes. Unlike MakeIndex, it is multi-lingual and supports UTF-8 encoding, both in the raw index input and in the tagged document output.

Overall, xindy has five key features:

1. *Internationalization* is the most important feature at all and was originally xindy's raison d'être: the standard distribution knows how to handle many languages and dialects correctly out of the box.
2. *Markup normalization* and *encoding support* is the ability to handle markup in the index keys in a transparent and consistent way, as well as different encodings.
3. *Modular configuration* enables the reusability of index configurations. For standard indexing tasks, L^AT_EX users do not have to do much except use available modules.
4. *Location references* go beyond page numbers. Locations may also be book names, section numbers or names, URLs, etc.
5. *Highly configurable markup* is another cornerstone. While this is usually not as important for L^AT_EX users, it comes in handy if one works with other author systems.

The focus of this paper is the current state of multi-lingual and encoding support that's available for xindy. The paper's scope does not include other features which I'll mention just in passing:

Locations are more than page numbers: Most index processors can work only with numbers, or maybe sequences of numbers such as "2.12". Going further, xindy features a generalized notion of structured location references that can be book names, music piece names, law paragraphs, URLs and other references. You can index "Genesis 3:16" and "Exodus 3:16" and Genesis will be in front of Exodus since they are not alphabetically sorted names any more, but terms in an enumeration.

Such location references may be combined into a range, such as 6, 7, 8, and 9 becoming "6–9". Also well-known are range specifications in the humanities, such as 6f or 6ff. With xindy, location ranges can also be formed over structured references, but some knowledge about the domain of the reference components must be available.

xindy is configured with a declarative style language, where declarations look like

```
(some-clause argument1 argument2 ...)
```

A file with such declarations is called an xindy module, and an xindy run may use several of these modules. This allows making available predefined modules for common indexing tasks, e.g., the f- and ff-range designation illustrated above. Xindy declarations are also used to configure output markup.

Last, but not least, xindy is the practical result in research about a theoretical model of index creation. Even if one does not use xindy the program, the model itself can provide valuable input for the creation of future index creation programs.

2 Multi-lingual sorting

Sorting is a multi-layered process where characters are first determined, placed into categories that are

albanian	greek	norwegian
belarusian	gypsy	polish
bulgarian	hausa	portuguese
croatian	hebrew	romanian
czech	hungarian	russian
danish	icelandic	serbian
dutch	italian	slovak
english	klinton	slovenian
esperanto	kurdish	spanish
estonian	latin	swedish
finnish	latvian	turkish
french	lithuanian	ukrainian
general	lower-sorbian	upper-sorbian
georgian	macedonian	vietnamese
german	mongolian	

Table 1: Predefined languages in xindy

sorted the same for now (collation classes), and sorted either left-to-right or right-to-left. If this results in index entries that are sorted the same but are not identical, characters are reclassified with different rules and sorted again to resolve the ambiguities. Words from most languages can be sorted with this process. It is standardized as ISO/IEC 14651 (*International String Ordering*).

2.1 Predefined languages

Xindy provides the ability to sort indexes in different languages; 44 of them are already prepackaged in the distribution and are listed in Table 1. For some of these languages there are multiple sorting definitions: e.g., German has two different kind of sort orders, colloquially called DIN and Duden sorting (more on that later).

While the sorting of all predefined languages may be expressed in terms of the ISO standard 14651 named above, xindy’s abilities go beyond that. The standard language modules are usable for indexes where index entries all belong to one language or where foreign terms are sorted as if they would be local. But if one mixes several languages in one index, e.g., in an author index, one is able to define the sort rules that should be used individually, just for this text. While this is some work, of course, xindy at least makes it possible to create indexes for such real international works that go beyond mere multi-lingualism.

2.2 Complexity of index sorting

One might ask if this paper doesn’t make a mountain out of a molehill, and what’s the big deal with all this supposed complexity of index sorting and creation

To address that valid question, I’d like to present a few peculiarities that show why index sorting is more complex than just sorting a few strings and why an ISO standard on string ordering is a good start but not the end of ordering index entries.

Cultural peculiarities For some languages, index sort order depends on context, or the term’s semantics. German is a good example for this complexity: there are two sort orders in wide usage and they differ in the sorting the “umlauts”. These are the vowels with two dots above: ä, ö, and ü.

One sort order sorts them as if there were a following ‘e’, i.e., ‘ä’ is sorted as ‘ae’, ‘ö’ is sorted as ‘oe’, and ‘ü’ is sorted as ‘ue’. That is the official sort order, and is defined in an official German standard, DIN 5007. This sort order is used for indexes in publications for the domestic market, for an audience that knows German and is thus expected to know that these characters are true letters on their own and not just vowels with accents. Such a domestic audience is also expected to have learned the sort order of umlauts in their first school year and will be able to cope with that cultural peculiarity.

A second way of index sorting drops that idiosyncratic German feature and sorts umlauts as if they were vowels with accents, i.e., ‘ä’ is sorted just like ‘a’, and so on. This sort order is used in indexes of publications for an international market, or where an international audience is expected to read this publication regularly. Especially dictionaries and phone books use this non-standard way of sorting; we want to give our foreign visitors a chance to look up the phone number of any Mr. or Ms. Müller they want to visit. This sort order has no official name, but is colloquially known as phone book sorting or “Duden sorting”, after the most important dictionary of the German language that uses this sort order.

Legacy rules Some special and non-obvious sort orders are so old that the reason behind them is not known (at least, not to me). An interesting example is French, where additional complexity has been introduced in some previous time when it comes to sorting names with accented characters: when two words have the same letters but differ in accents, the existence of accents decides the final sort order — but backwards, from right to left!

The most prominent example is the four words

cote côte coté côté

In the first pass of sorting a French index, these four words are sorted the same. In the second pass, they are still sorted the same — the second pass sorts uppercase letters before lowercase letters. The third pass then sorts from right to left and puts

non-accented letters before accented letters:

```

→
cote
←
côte
coté
côté

```

This finally results in the sort order shown in the table above.

Character recognition Sometimes legacy representations in files on computers introduce complexities: In Spanish, ‘ch’ and ‘ll’ are *one* letter and sorted accordingly, whereas in all other European languages they are two letters. Traditionally, these Spanish letters are represented by two characters in a file; an index sort processor has to recognize them as such.

There is also the problem of what to do when a character appears in an index that does not exist in that language, e.g., there is no ‘w’ in Latin. Should one be pragmatic and sort it like modern languages of the Roman family, between ‘v’ and ‘x’? Or should one place it somewhere in the non-letter group? The order might very well depend on the target audience and intent of the respective index.

In the \TeX world, character recognition is arguably less an issue for non-Latin scripts — \TeX authors are used to specifying their characters with exact encodings or transliteration. Especially the rise of Unicode text editors and their enhanced input support for non-Latin scripts make identification of characters easier than the supposedly ASCII-like representation in traditional encodings.

Beyond Europe The examples so far were “just” about European languages. (Admittedly, because I know most about them . . .) Some languages use phonetic sorting where one needs additional information about words that are used in the sort algorithm. This does not change the algorithm itself, but available authoring systems often do not support that aspect at all. (xindy does not support it out-of-the-box either, but it has the functionality to describe such sorting in its language modules.)

Other languages use aspects of glyphs such as strokes or number of strokes for sorting. Diacritics may or may not influence sorting; sometimes they are vowels, sometimes they just denote special emphasis and can be ignored.

3 One sort order is not enough

For multi-lingual index creation it is not sufficient to define sort orders for languages. Having defined a language module with the default sort order of

German, French, or any other language is a good and necessary start, and many index processors stop at that. But it is not sufficient for production of actual indexes where sorting rules appear that are not covered by standards.

For example, in author indexes some languages handle parts of nobility names differently, depending on whether they are part of the name or a true peerage title. In registers of places, city names might be sorted differently than spelled. Transliteration must be taken into account, just like combination of alphabets within one index.

This boils down to the requirement that project- or document-specific sort rules are needed. While one book might sort ‘St.’ as it is written, Gault Millau will need a different sort order for its register — they sort it as ‘Saint’. MakeIndex introduced a way to do that by specifying print and sort keys explicitly, as in `\index{Saint Malo@St. Malo}`: It demands from the author that this explicit specification must be used every time that term appears.

Xindy goes a step further and allows the user to specify sort orders in a separate style file that may be used just for one document or reused for all documents in a project. It still allows using an explicit sort and print key in your \TeX document — but experience has shown that it is much less error-prone to declare it once in an external file for a whole group of index terms than to write it explicitly in each occurrence of that group.

4 Examples for xindy style declarations

Let’s have a look at how such document-specific declarations are done. We demonstrate their use for two purposes: index entry normalization and entry sorting.

Markup normalization is the process to decide if two raw index entries denote the same term and should be combined into one processed index entry, i.e., if they should be *merged*.

Especially with \TeX , it might be that the same term appears differently in the raw index. This is caused by macro expansions, especially when one produces part of the index entries automatically. Depending on the state of \TeX ’s processing, macros in the raw index are sometimes expanded and sometimes not expanded.

Here comes into play a point where xindy differs from MakeIndex: it ignores \TeX or \LaTeX tags (macro names and braces) by default. With xindy, you can write `\index{\textbf{term}}` and this will be the index entry “term”, `\textbf` and the braces will be ignored. (Such index entries are usually not

input manually, but are generated by other macros.)

So, how would one index METAFONT, written in the Metafont logo font? The MakeIndex way would be to use `\index{METAFONT@MF}` for every to-be-indexed occurrence of METAFONT, and that way still works with xindy. But in addition, one can use an xindy style file with the declaration

```
(merge-rule "\MF" "METAFONT")
```

and just use `\index{MF}` in the document. No risk to add typos to one's index entries; these index entries will be sorted as 'METAFONT' but output as METAFONT.

Merge rules may influence whole index terms or just parts of them. One can also use regular expressions to normalize large classes of raw terms.

Sort specifications Sort orders are specified with very similar declarations:

```
(sort-rule "ä" "a~e")
```

tells xindy to place 'ä' between 'a' and 'b'. (The special notion '~e' means "at the end"; there is also '~b' for "at beginning".)

This is the low-level way to specify sort orders, and it is used to create special document- or project-specific sort orders as mentioned above. It is possible to create whole language sort-order modules with that method as well — we did so at the start of the xindy project.

But then Thomas Henlich wrote *make-rules*, a preprocessor to create xindy language modules for languages where sort order can be expressed with the ISO 14651 concept. For that preprocessor, one describes alphabets and sort orders over collation classes with multiple passes, and xindy modules with sort rules as shown above are created as a result.

5 Encoding of raw index files — LICR and UTF-8

At the moment, the most often used encoding for raw index files is the \LaTeX output of `\index` commands. That encodes non-ASCII characters as macros; the representation is called *\LaTeX Internal Character Representation* or LICR, as described in section 7.11 of *The \LaTeX Companion*, 2nd ed. Xindy knows about LICR: xindy modules exist with merge rules to recognize these character representations. A special invocation command for \LaTeX , `texindy`, picks them up automatically, so authors have no need to think about them.

At the moment, LICR is mapped to an ISO-8859 encoding that's appropriate for the given language, and that encoding is then the base for xindy's sort rules. Please note that this is completely unrelated to the encoding used in the author's \LaTeX document.

You can use UTF-8 there with the `inputenc` package, but that encoding doesn't matter for index sorting. When the raw index arrives at xindy, that original encoding is not visible any more; we see only LICR. And we just need ISO-8859 encodings for sorting languages that are supported by \LaTeX 's standard setup, which mostly use European scripts.

While this is appropriate and useful for European languages, it won't help authors with documents in Arabic, Hebrew, Asian, or African languages. But they also won't use LICR much anyhow and will probably be better served by new programs like $X_{\text{F}}\TeX$ or Omega/Aleph. For these users, all language modules are supplied in a variant that knows about UTF-8 encodings as output by $X_{\text{F}}\TeX$ or Omega's low-level output of (Unicode) characters. If one has a raw index file that was produced by these systems, one can use xindy; it will "just work".

Looking beyond UTF-8 is still not necessary in the \TeX world; we have no \TeX engine that will output UTF-16 or even wider characters to a raw index file or expect such encodings in a processed index file. That's good, because xindy can't handle UTF-16 input — yet. This will probably be an enhancement of one the next major releases and shall help to open up xindy's appeal beyond \TeX -based authoring environments.

6 Availability

Through release 2.3, xindy was available only for GNU/Linux and other Unix-like systems. At the time of writing, release 2.4 has been prepared which is nicknamed the *TUG30 release*, to honor TUG's 30th birthday. This release adds support for Windows (2000, XP, and Vista), thus widening the potential user base considerably. For now, installation of a Perl system is needed to use xindy; this should not be a big obstacle.

The TUG30 release is available for download at xindy's Web site www.xindy.org. Currently, it is there in source form; binary distributions for several operating systems will be added as time permits.

While release 2.3 is included in \TeX Live 2008, with executables for nearly all the platforms except Windows, including Mac OS X, we were not able to finish the new release 2.4 in time to make it to the DVD. Hopefully, it will become available via the new on-line update mechanism soon. Eventually, full support for xindy will be available in \TeX Live 2009.

The best place to look for user documentation about xindy is *The \LaTeX Companion*, 2nd ed., section 11.3. Documentation on the Web site is technically more complete, but improvements of its organization and accessibility are high on our to-do list.

A newbie's experiences with Lilypond, Lilypond-book, L^AT_EX and Perl

Joe M^cCool

Southern Regional College, Ireland

mccoolj (at) src dot ac dot uk

<http://benburb.demon.co.uk/apache2-default/joe.html>

Abstract

The author is an active Irish traditional musician. He is also a keen inland boater. He is having a lot of fun composing a book on “Traditional Music for Boaters”.

In this paper he describes his successes and frustrations using Lilypond, Lilypond-book, L^AT_EX and ABC musical notation. Lilypond and L^AT_EX have a lot in common. Neither are WYSIWYG, neither demand GUI's. Both compile simple flat files to produce beautiful graphical output.

Lilypond's original manifestations produced output directly for L^AT_EX, but of late users writing books have been encouraged to use Lilypond-book. This looks for Lilypond code within L^AT_EX source files and produces graphics and associated instructions which can then be processed by L^AT_EX.

Most joy has come from automating these processes via GNU/Linux and Perl.

1 What's out there

By definition classical music has an inherent connection with books and text. Traditional music does not. Classical music is written down in scores. Historically, traditional music is not. It is largely an aural medium and tunes are learnt by ear. Having said that, musical scores do now have an important role to play. More and more young traditional musicians are learning how to read scores. Printed material allows us to store pieces that would otherwise be lost and pencil and paper is a useful aid to composition.

Musicians like to share pieces and the arrival of the personal computer has made this easier than ever. But, where years ago we popped manuscripts in the post, we now need to share scores electronically. This has brought about a plethora of programs and systems that enable us to do so.

1.1 ABC notation

In the 1980's Chris Walshaw, then at the University of Cambridge, began writing out fragments of folk/traditional tunes using letters to represent the notes. This became gradually formalised into what is known as the ABC “standard”.¹ Numerous small programs have appeared to convert ASCII files of ABC to printed scores.² There are also programs to convert ABC code to midi.

Here's an example of an ABC file:

```
X: 1
T:The trout
```

¹ www.walshaw.plus.com/abc

² for example, moinejf.free.fr

```
C:Franz Schubert
O:Austria
M:C|
L:1/8
Q:1/4=160
K:C
G2|"C"c2c2e2e2|"C"c4G2G2|"G"G3G dcBA|"G"
G4 z2G2|"C"c2c2 e2e2|"C"c4G2c2|"G"B2AB
....
```

The K: field represents the tune's key. All lines below this contain the music. Lines above are header fields, with X: representing an index for this particular tune in a file of other ABC's. Notes in quotes represent accompaniment chords.

At first sight ABC seems ideal for what I wish to do. It has a simple input format. It can output PostScript files that I can incorporate into L^AT_EX documents. ABC programs are open source and, most importantly, there are huge collections of ABC source files available on the internet.

But, I have a few issues with ABC:

- there is a gross lack of standardisation. What standardisation exists is often ignored by the authors of ABC files.
- I need a system comparable to L^AT_EX in terms of typesetting quality. ABC does not have the fine grained control of L^AT_EX.
- It is difficult to avoid clashing problems: note heads clashing with bar numbers, or grace notes clashing with accidentals.
- At the time I started my project, the ABC mailing list seemed to vanish!

1.2 Commercially available software

Under Microsoft Windows, several commercial programs are available for typesetting music. Finale,³ Sibelius,⁴ and Cakewalk⁵ are well known. Noteworthy Composer is available as shareware.⁶

1.3 MusicT_EX and MusiX_TE_X

Both of these, based on what I could find out about them,⁷ appeared too complex for me. They did not seem to have an active development or support community.

1.4 Lilypond

I rejected the commercial products and Noteworthy because they use proprietary file formats and they rely on GUI interfaces. I am a traditionalist in more ways than one. Long ago I realised the power, elegance and beauty of plain ASCII files under GNU/Linux. Hence my final choice of Lilypond.^{8,9}

- Lilypond's originators have objectives very similar to those of L^AT_EX: "to print music in the best traditions of classical engraving with minimum fuss".
- Lilypond uses plain ASCII, not dissimilar to ABC.
- Lilypond enjoys ongoing development.
- Its documentation is excellent.
- Very active user support via mailing lists.
- Very fast keying of source files:
 - Note durations need stating once only. In the input `a4 b`, the notes `a` and `b` have the same duration.
 - Notes can be raised an octave using a following `'` or lowered by a following `,`. Lilypond also provides a `relative` mode in which it will position notes on the scale in a common sense, reasonable fashion.

```
\relative c'' {
  b c d c b c bes a
}
```

These notes will all be placed within the scale, rather than climbing higher upward.

- key transposition is easy: `\transpose d e ...`
- The excitement and delight I found putting together my first Lilypond scripts was matched only by that of my first L^AT_EX scripts. Lilypond and L^AT_EX really are first cousins!

³ www.finalemusic.com

⁴ www.sibelius.com

⁵ www.cakewalk.com

⁶ www.noteworthysoftware.com

⁷ www.tex.ac.uk/cgi-bin/texfaq2html?label=music

⁸ www.lilypond.org

⁹ In rural Ireland we have a saying: "If you think a donkey will do the job, use a horse!"

1.5 Example Lilypond file

```
\version "2.11.33"
\header {
  composer = "Joe Mc Cool"
  title = "The Eight Lock"
  dedication = ""
}
voicedefault = {
  \relative c'
  \clef treble
  \key g \major
  \time 4/4
  \repeat volta 2 {
    \time 4/4
    \clef treble
    d'4 d8 e d c b4
    ....
  }
  \repeat volta 2 {
    a'4 a8 a b c4.
    ....
  }
}
\include "../new.score.ly"
```

Notice the indentation of code, similar to programming languages. `voicedefault` is a musical object that will subsequently process as a score (see below), or a midi file.¹⁰

As my project gathered weight I got tired of having to edit individual Lilypond files in order to change the overall look of my book, hence my use of the include statement. `new.score.ly` consists of:

```
\score{
<<
{
\voicedefault
}
>>
\layout{
  #(layout-set-staff-size 20)
}
}
```

In order to change the overall look of all my pieces, for example to change the staff size, all I have to do is edit the above.

2 My approach

My approach is constrained by the following goals:

- Each page should contain an integer number of tunes. Classical musicians are happy (or at least willing) to turn a page in the middle of a piece, traditional musicians are not.
- Traditional music is often played in sets. Two or three jigs will be played one after the other,

¹⁰ though midi support is regrettably not good in Lilypond.

or a group of hornpipes. When new tunes are added to the collection, they must be kept as close together as possible to other members of their set, ideally on the same page, or on the same spread.

- Brief texts and footnotes must appear on the page of the tune to which they refer.
- Index entries must have the form '*name : type : page number*'. This reveals the page number and the tune type for each entry.
- The build process should produce midi files.

3 Combining Lilypond and L^AT_EX

Here is a simple example of a L^AT_EX file (`small.ly`) containing Lilypond code:

```
\documentclass{article}
\begin{document}
\noindent
Some text before a musical snippet.\\
\begin[quote,fragment]{lilypond}
{
  c' e' g' e'
}
\end{lilypond}
Another snippet:\\
\begin[quote,fragment]{lilypond}
{
  f' g' a' b'
}
\end{lilypond}
Some more text.\\
\end{document}
```

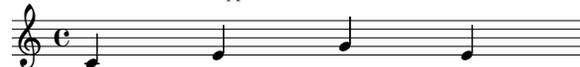
This is processed by the command line:

```
lilypond-book -f latex --psfonts
               --output OUTPUT small.tex
```

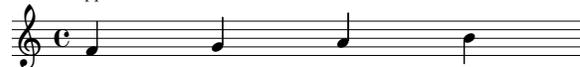
And in the OUTPUT directory Lilypond creates the following files:

```
lily-2b589ef505-1.eps
lily-2b589ef505.eps
lily-2b589ef505.ly
lily-2b589ef505-systems.tex
lily-2b589ef505-systems.texti
lily-2b589ef505.txt
lily-eb070afcf7-1.eps
lily-eb070afcf7.eps
lily-eb070afcf7.ly
lily-eb070afcf7-systems.tex
lily-eb070afcf7-systems.texti
lily-eb070afcf7.txt
small.dep
small.tex
snippet-map.ly
snippet-names
tmpMQ9ShM.aux
```

Some text before a musical snippet.



Another snippet:



Some more text.

Figure 1: `small.pdf`

Here the file `small.tex` constitutes the final output. This is then processed by L^AT_EX in the normal way, with the result shown in Fig. 1.

Lilypond-book creates a graphic for each line of music. It also creates a graphic for the whole snippet—in our small example, `2b589ef505-1.eps` and `2b589ef505.eps`. My project has currently gathered about 200 tunes and the number of small files in OUTPUT hovers around 3500!

Contents of `OUTPUT/small.tex`:

```
\documentclass{article}
\usepackage{graphics}
\begin{document}
\noindent
Some text before a musical snippet.\\
{%
\parindent 0pt%
\ifx\preLilyPondExample \undefined%
  \relax%
\else%
  \preLilyPondExample%
\fi%
\def\lilypondbook{%
\input lily-2b589ef505-systems.tex%
\ifx\postLilyPondExample \undefined%
  \relax%
\else%
  \postLilyPondExample%
\fi%
}
Another snippet:\\
{%
\parindent 0pt%
\ifx\preLilyPondExample \undefined%
  \relax%
\else%
  \preLilyPondExample%
\fi%
\def\lilypondbook{%
\input lily-eb070afcf7-systems.tex%
\ifx\postLilyPondExample \undefined%
  \relax%
\else%
  \postLilyPondExample%
\fi%
}
Some more text.\\
\end{document}
```

If L^AT_EX can fit the snippet into a page in its entirety it uses the whole graphic, otherwise it uses individual lines, placing some of the lines on the following page. This breaks my first requirement: pages should contain only an integer number of snippets.

4 Overcoming the first limitation

My first approach to this problem was to wrap the snippet in a Figure environment:

```
\begin{figure}
....
lilypond code
....
\end{figure}
```

An integer number of tunes then appeared on a page, but I found that the positioning of the graphics was inconsistent, particularly at the end of chapters. Google reported that lots of people had suffered from this same problem, but I could find no solutions.

Indeed the suggestion was that I abandon L^AT_EX altogether and use only lilypond and a particular stylesheet.¹¹

I also tried using the standard utility *grep* to find a relation between my L^AT_EX file, the Lilypond file and the *lilypond-book*-generated *eps* files. I intended then to use conventional `\includegraphics` commands to position the graphics manually. This proved too cumbersome.

My final code ended up as:

```
\noindent
\begin{minipage}{\columnwidth}
\index{mytune:reels}
\lilypondfile{mytune.ly}\\\\
\include{mytune.tex}
\end{minipage}
```

Here *mytune.tex* contains notes pertaining to this particular piece. Wrapped within the minipage, it was guaranteed to appear on the same page. It might also contain footnotes.

This example also shows that if the Lilypond script is long, it can be stored in a file and referred to with `\lilypondfile`; *lilypond-book* then processes its argument.

5 Clever includes

Ideally I would have liked code such as:

```
\newcommand{\lily}[1]{
  \lilypondfile{#1}{#1}
  ....
\lily{lilys/my.tune.ly}
```

¹¹ lsr.dsi.unimi.it/LSR/Item?id=368

but *lilypond-book* complained about not being able to find files. It is just not that clever. It is not able to process my `\newcommand`.

6 Source collections

Ironically the largest collections of traditional music from all over the world are held in ABC files and there are quite a few search engines tuned specifically for searching ABC sites.¹² There is also a Python script available that converts ABC to Lilypond (*abc2ly*).

Again, possibly because of the lack of ABC standards, *abc2ly* does not produce very tidy code and sometimes gets the repeats plain wrong. It is often brought to its knees by idiosyncratic ABC.

7 And then there was Perl

Perl is ideal for processing text. Both L^AT_EX and Lilypond are text based, so the marriage is obvious. My collecting of ABC files and their subsequent placement in my book is now almost completely automatic:

- ABC file arrives in target folder (often via email)
- A Perl daemon:
 1. makes a backup
 2. cleans up ABC code
 3. creates an index entry
 4. merges text to precede or follow this item
 5. runs *abc2ly*
 6. adds name of lily file to compilation list
- a *make* invocation puts together book version

I think of this process as resembling a trout: the Perl daemon watches for an ABC file arriving as a result of an Internet search — just as a trout watches for minnows! It is not perfect, but I have good error reporting in place and mistakes are easily fixed by hand.

When sufficient new tunes have been added to the repository, another Perl script employs *lilypond-book*, *latex*, *dvips* and *ps2pdf* to produce the final copy.

8 Subversion

Small changes are made to this project daily and sometimes the editing is done using machines on different sites. A small change can have a disastrous effect on the end product. For this reason the whole project is controlled using the *Subversion*¹³ version control system.

¹² for example, trillian.mit.edu/~jc/cgi/abc/tunefind
¹³ subversion.tigris.org

MetaPost developments: MPlib project report

Taco Hoekwater

Elvenkind BV

Dordrecht, The Netherlands

<http://tug.org/metapost>

Abstract

The initial stage of the MPlib project has resulted in a library that can be embedded in external programs such as Lua \TeX , and that is also the core of the `mpost` program. This paper presents the current state of affairs, the conversion process of the MetaPost source code, and the application interface to the library.

1 MPlib project goals

The MPlib project is a logical consequence of the transfer of MetaPost development from its author John Hobby to the MetaPost development team. It originates from a desire to update MetaPost for use in the 21st century. The first thing that needed to be done to make that happen was updating the program source code and infrastructure to be closer to today's programming standards.

These days, programs are often written in the form of shared libraries, with a small frontend application. When written in this way, a program can not only be used as a standalone program, but can also easily and efficiently be (re)used as a plugin inside other programs, or turned into a multi-user system service.

With current MetaPost, such alternate uses are impossible because of the internals of the code. For example, MetaPost uses many internal global variables. This is a problem because when two users would be accessing a 'MetaPost' library at the same time, they would alter each other's variables. For another example, MetaPost has static memory allocation: it requests all the computer memory it will ever use right at startup. It never bothers to free that memory, because it counts on the operating system to clean up automatically after it exits. And one file example: MetaPost not only opens files at will, but it also writes to and, even more problematically, reads from the terminal directly.

A large part of updating MetaPost is therefore fixing all these issues. But while doing this, there are other weirdnesses to take care of at the same time.

The present subsystem for typesetting labels (`btex ... etex`) is pretty complicated, requiring an array of external programs to be installed on top of the normal `mpost` executable. And from a system viewpoint, the error handling of MetaPost is not very good: it often needs user interaction, and

in most other cases it simply aborts. Finally, the whole process of installing the program is complicated: a fair bit of the \TeX Live development tree is needed to compile the executable at all.

2 Solutions

Many of the problems mentioned above are a side-effect of the age of the source code: the source is largely based on METAFONT, and therefore written in Pascal WEB. And the bits that are not in Pascal WEB are an amalgam of C code borrowed from other projects, most notably pdf \TeX .

Not wanting to lose the literate programming documentation, we had only one practical way to proceed: using the CWEB system. CWEB has the same functionality that Pascal WEB has, except that it uses C as the programming language instead of Pascal, and it has some extensions so that it does not get in the way of the 'normal' C build system.

Using CWEB, a single programming language now replaces all of the old Pascal and C code. The code has been restructured into a C library, the label generator `makempx` has been integrated, and compilation now depends only on the `ctangle` program and the normal system C compiler, so that a simple Autoconf script can be used for configuration of the build process.

3 Code restructuring

Whereas converting the C code of the font inclusion and label processing subsystems to CWEB was a fairly straightforward process, converting the Pascal WEB core of MetaPost was a more elaborate undertaking.

In the first stage, the Pascal code within the WEB underwent an automatic rough conversion into C code. Afterwards, the generated C code was manually cleaned up so that it compiled properly using `ctangle`. This part took roughly one month, and the end result was an executable that was 'just like'

Pascal MetaPost, but using CWEB as source language instead of Pascal WEB.

After that was done, the real work started:

- All of the global variables were collected into a C structure that represents an ‘instance’ of MetaPost.
- The Pascal string pool was stripped away so that it is now used only for run-time generated strings. Most internal functions now use normal C strings.
- The PostScript backend was isolated from the core of the program so that other backends can be added much more easily.
- All of the exported functions now use a C namespace.
- Where it was feasible to do so, MPlib uses dynamic memory allocation. There are a few exceptions that will be mentioned later.
- All input and output now uses function pointers that can be configured through the programming interface.
- The MPlib library never calls `exit()` itself but instead returns to the calling program with a status indicator.

4 Using the library from source code

Using the MPlib library from other program code is pretty straightforward: first you set up the MPlib options, then you create an MPlib instance with those options, then you run MetaPost code from a file or buffer, and finally finish up.

The options that can be controlled are:

- various command-line options that are familiar from `mpost`, such as whether MetaPost starts in INI mode, the `mem_name` and `job_name`, ‘troff’ mode, and the non-option part of the command line,
- the size of the few remaining statically allocated memory arrays,
- various function pointers like those for input and output, file searching, the generator function for typeset labels, and the ‘editor escape’ function,
- the start value of the internal randomizer,
- and finally a ‘userdata’ pointer that is never used by MPlib itself but can be retrieved by the controlling program at any time.

The application programming interface at the moment is very high-level and simplistic. It supports two modes of operation:

- emulation of the command-line `mpost` program, with traditional I/O and interactive error handling,

- an interpreter that can repeatedly execute individual string chunks, with redirected I/O and all errors treated as if `nonstopmode` is in effect.

For the string-based interpreter, there are a few extra functions:

- the runtime data can be fetched; this comprises the logging information and the internal data structure representation of any generated figures,
- the instance’s error state can be queried,
- the userdata pointer can be retrieved,
- some statistics can be gathered,
- PostScript can be generated from the image output,
- and some glyph information can be retrieved; this is useful if you want to create a backend yourself.

4.1 Examples

Here is a minimalistic but complete example that uses the `mpost` emulation method in C code:

```
#include <stdlib.h>
#include "mplib.h"
int main (int argc, char **argv) {
    MP mp;
    MP_options *opt = mp_options();
    opt->command_line = argv[1];
    mp = mp_initialize(opt);
    if (mp) {
        int history = mp_run(mp);
        mp_finish(mp);
        exit (history);
    } else {
        exit (EXIT_FAILURE);
    }
}
```

Given the basic library functionality now available, it is reasonably straightforward to create bindings for other languages. We have done this for Lua, and here is a second example that uses these Lua language bindings. The Lua bindings are always based on string execution, and the option setting and instance creation are merged into a single `new` function:

```
local mplib, mp, l, chunk
mplib = require('mplib')
mp = mplib.new ({ini_version = false,
                mem_name = 'plain'})

chunk = [[
    beginfig(1);
        fill fullcircle scaled 20;
    endfig;
]]
if mp then
```

```

l = mp:execute(chunk)
if l and l.fig then
  print (l.fig[1]:postscript())
end
mp:finish()
end

```

5 Using the command-line program

On the command line very little has changed. The executable `mpost` still exists. Now it is merely a thin wrapper that is much like the C code example shown earlier, except with a few hundred more lines because it has to set up the command-line properly.

As mentioned already, the `makempx` functionality has also been converted into a small library that is used by `mpost` to emulate the old label creation system. The programs `makempx`, `dvitomp`, `mpto`, and `dmp` have been merged into this library and no longer exist as separate programs. For backward compatibility, a user-supplied external label generation program will be called if the `MPXCOMMAND` environment variable is set, but normally `mpost` sets up the MPlib library to use the new embedded code.

In the normal case, the only external program that will be run is the actual typesetter (\TeX or Troff). The command-line of `mpost` is extended to allow the specification of which typesetter to use.

6 Planning and TODO

Most development took place at the beginning of 2008, after which we entered a period of extensive testing. This way we were relatively confident that the first version of the library was basically usable from the start.

The first beta release (1.091) was presented at the TUG 2008 conference. The distribution contains the MPlib library source, the code for the ‘`mpost`’ frontend, code for the Lua bindings, and the C and Lua API documentation.

The final MPlib 1.100 release will be released later in 2008, and the MPlib-based distribution will replace the Pascal MetaPost distribution from that point forward.

After this release, work on the TODO list will continue. Items already on the wishlist:

- Start using dynamic memory allocation for the remaining statically allocated items: the main memory, the number of hash entries, the number of simultaneously active macro parameters, and the maximum allowed input nesting levels.
- An extension is being planned under the working name ‘MegaPost’ that will extend the range and precision of the internal data types.
- In the future, we want to use MPlib to generate (OpenType) fonts. This requires support from the core engine like overlap detection and calculation of pen envelopes.
- Error strategies are planned so that the behaviour of the string-chunk based interface can be configured properly.
- There are desires to expand the API. For instance, it would be nice if applications were able to use the equation solver directly.

7 Acknowledgements and contact

The MPlib project could not have been done without funding by the worldwide \TeX user groups, in particular: DANTE, TUG India, TUG, NTG, CSTUG, and GUST. A big thank you goes to all of you for giving us the opportunity to work on this project.

The general contact information for MetaPost and MPlib has not changed:

- Web site and portal:
<http://tug.org/metapost>
- User mailing list:
<http://lists.tug.org/metapost>
- Source code and bug tracker:
<http://foundry.supelec.fr/projects/metapost>

The T_EX–Lua mix

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

Abstract

An introduction to the combination of T_EX and the scripting language Lua.

1 Introduction

The idea of embedding Lua into T_EX originates in some experiments with Lua embedded in the SciTE editor. You can add functionality to this editor by loading Lua scripts. This is accomplished by a library that gives access to the internals of the editing component.

The first integration of Lua in pdfT_EX was relatively simple: from T_EX one could call out to Lua and from Lua one could print to T_EX. My first application was converting math written in a calculator syntax to T_EX. Following experiments dealt with MetaPost. At this point integration meant as little as: having some scripting language as an addition to the macro language. But, even in this early stage further possibilities were explored, for instance in manipulating the final output (i.e. the PDF code). The first versions of what by then was already called LuaT_EX provided access to some internals, like counter and dimension registers and the dimensions of boxes.

Boosted by the Oriental T_EX project, the team started exploring more fundamental possibilities: hooks in the input/output, tokenization, fonts and nodelists. This was followed by opening up hyphenation, breaking lines into paragraphs and building ligatures. At that point we not only had access to some internals but also could influence the way T_EX operates.

After that, an excursion was made to MPlib, which fulfilled a long standing wish for a more natural integration of MetaPost into T_EX. At that point we ended up with mixtures of T_EX, Lua and MetaPost code.

As of mid-2008 we still need to open up more of T_EX, like page building, math, alignments and the backend. Eventually LuaT_EX will be nicely split up in components, rewritten in C, and we may even end up with Lua gluing together the components that make up the T_EX engine. At that point the interoperation between T_EX and Lua may be even richer than it is now.

In the next sections I will discuss some of the

ideas behind LuaT_EX and the relationship between Lua and T_EX and how it presents itself to users. I will not discuss the interface itself, which consists of quite a number of functions (organized in pseudo-libraries) and the mechanisms used to access and replace internals (we call them callbacks).

2 T_EX vs. Lua

T_EX is a macro language. Everything boils down to either allowing stepwise expansion or explicitly preventing it. There are no real control features, like loops; tail recursion is a key concept. There are only a few accessible data structures, such as numbers, dimensions, glue, token lists and boxes. What happens inside T_EX is controlled by variables, mostly hidden from view, and optimized within the constraints of 30 years ago.

The original idea behind T_EX was that an author would write a specific collection of macros for each publication, but increasing popularity among non-programmers quickly resulted in distributed collections of macros, called macro packages. They started small but grew and grew and by now have become pretty large. In these packages there are macros dealing with fonts, structure, page layout, graphic inclusion, etc. There is also code dealing with user interfaces, process control, conversion and much of that code looks out of place: the lack of control features and string manipulation is solved by mimicking other languages, the unavailability of a float datatype is compensated by misusing dimension registers, and you can find provisions to force or inhibit expansion all over the place.

T_EX is a powerful typographical programming language but lacks some of the handy features of scripting languages. Handy in the sense that you will need them when you want to go beyond the original purpose of the system. Lua is a powerful scripting language, but knows nothing of typesetting. To some extent it resembles the language that T_EX was written in: Pascal. And, since Lua is meant for embedding and extending existing systems, it makes sense to bring Lua into T_EX. How do they compare? Let's give some examples.

About the simplest example of using Lua in \TeX is the following:

```
\directlua { tex.print(math.sqrt(10)) }
```

This kind of application is probably what most users will want and use, if they use Lua at all. However, we can go further than that.

3 Loops

In \TeX a loop can be implemented as in the plain format (editorial line breaks, but with original comment):

```
\def\loop#1\repeat{\def\body{#1}\iterate}
\def\iterate{\body\let\next\iterate
\else\let\next\relax\fi\next}
\let\repeat=\fi % this makes \loop..\if..\repeat
% skippable
```

This is then used as:

```
\newcount \mycounter \mycounter=1
\loop
...
\advance\mycounter 1
\ifnum\mycounter < 11
\repeat
```

The definition shows a bit how \TeX programming works. Of course such definitions can be wrapped in macros, like:

```
\forloop{1}{10}{1}{some action}
```

and this is what often happens in more complex macro packages. In order to use such control loops without side effects, the macro writer needs to take measures to permit, for instance, nested usage and avoid clashes between local variables (counters or macros) and user-defined ones. Above we used a counter in the condition, but in practice expressions will be more complex and this is not that trivial to implement.

The original definition of the iterator can be written a bit more efficiently:

```
\def\iterate{\body \expandafter\iterate \fi}
```

And indeed, in macro packages you will find many such expansion control primitives being used, which does not make reading macros easier.

Now, get me right, this does not make \TeX less powerful, it's just that the language is focused on typesetting and not on general purpose programming, and in principle users can do without that: documents can be preprocessed using another language, and document specific styles can be used.

We have to keep in mind that \TeX was written in a time when resources in terms of memory and CPU cycles were far less abundant than they are now. The 255 registers per class and (about) 3000

hash slots in original \TeX were more than enough for typesetting a book, but in huge collections of macros they are not all that much. For that reason many macro packages use obscure names to hide their private registers from users and instead of allocating new ones with meaningful names, existing ones are shared. It is therefore not completely fair to compare \TeX code with Lua code: in Lua we have plenty of memory and the only limitations are those imposed by modern computers.

In Lua, a loop looks like this:

```
for i=1,10 do
...
end
```

But while in the \TeX example, the content directly ends up in the input stream, in Lua we need to do that explicitly, so in fact we will have:

```
for i=1,10 do
tex.print("...")
end
```

And, in order to execute this code snippet, in Lua \TeX we will do:

```
\directlua 0 {
for i=1,10 do
tex.print("...")
end
}
```

So, eventually we will end up with more code than just Lua code, but still the loop itself looks quite readable and more complex loops are possible:

```
\directlua 0 {
local t, n = { }, 0
while true do
local r = math.random(1,10)
if not t[r] then
t[r], n = true, n+1
tex.print(r)
if n == 10 then break end
end
end
}
```

This will typeset the numbers 1 to 10 in randomized order. Implementing a random number generator in pure \TeX takes a fair amount of code and keeping track of already defined numbers in macros can be done with macros, but neither of these is very efficient.

4 Basic typesetting

I already stressed that \TeX is a typographical programming language and as such some things in \TeX are easier than in Lua, given some access to inter-

```
\setbox0=\hbox{x}\the\wd0
```

In Lua we can do this as follows:

```
\directlua 0 {
  local n = node.new('glyph')
  n.font = font.current()
  n.char = string.byte('x')
  tex.box[0] = node.hpack(n)
  tex.print(tex.wd[0]/65536 .. "pt")
}
```

One pitfall here is that T_EX rounds the number differently than Lua. Both implementations can be wrapped in a macro resp. function:

```
\def\measured#1{\setbox0=\hbox{#1}\the\wd0\relax}
```

Now we get:

```
\measured{x}
```

The same macro using Lua looks as follows:

```
\directlua 0 {
  function measure(chr)
    local n = node.new('glyph')
    n.font = font.current()
    n.char = string.byte(chr)
    tex.box[0] = node.hpack(n)
    tex.print(tex.wd[0]/65536 .. "pt")
  end
}
\def\measured#1{\directlua0{measure("#1")}}
```

In both cases, special tricks are needed if you want to pass for instance a # character to the T_EX implementation, or a " to Lua; namely, using \# in the first case, and Lua’s “long strings” marked with double square brackets in the second.

This example is somewhat misleading. Imagine that we want to pass more than one character. The T_EX variant is already suited for that, but the Lua function will now look like:

```
\directlua 0 {
  function measure(str)
    if str == "" then
      tex.print("0pt")
    else
      local head, tail = nil, nil
      for chr in str:gmatch(".") do
        local n = node.new('glyph')
        n.font = font.current()
        n.char = string.byte(chr)
        if not head then
          head = n
        else
          tail.next = n
        end
        tail = n
      end
      tex.box[0] = node.hpack(head)
      tex.print(tex.wd[0]/65536 .. "pt")
    end
  end
}
```

```
end
end
}
```

And still it’s not okay, since T_EX inserts kerns between characters (depending on the font) and glue between words, and doing that all in Lua takes more code. So, it will be clear that although we will use Lua to implement advanced features, T_EX itself still has quite a lot of work to do.

5 Typesetting stylistic variations

In the following examples we show code, but it is not of production quality. It just demonstrates a new way of dealing with text in T_EX.

Occasionally a design demands that at some place the first character of each word should be up-percase, or that the first word of a paragraph should be in small caps, or that each first line of a paragraph has to be in dark blue. When using traditional T_EX the user then has to fall back on parsing the data stream, and preferably you should then start such a sentence with a command that can pick up the text. For accentless languages like English this is quite doable but as soon as commands (for instance dealing with accents) enter the stream this process becomes quite hairy.

The next code shows how ConT_EXt MkII defines the \Word and \Words macros that capitalize the first characters of a word or words. The spaces are really important here because they signal the end of a word.

```
\def\doWord#1%
  {\bgroup\the\everyuppercase\uppercase{#1}%
  \egroup}

\def\Word#1%
  {\doWord#1}

\def\doprocesswords#1 #2\od
  {\doifsomething{#1}{\processword{#1} % space!
  \doprocesswords#2 \od}}

\def\processwords#1%
  {\doprocesswords#1 \od\unskip}

\let\processword\relax

\def\Words
  {\let\processword\Word \processwords}
```

The code here is not that complex. We split off each word and feed it to a macro that picks up the first token (hopefully a character) which is then fed into the \uppercase primitive. This assumes that for each character a corresponding uppercase variant is defined using the \uccode primitive. Excep-

tions can be dealt with by assigning relevant code to the token register `\everyuppercase`. However, such macros are far from robust. What happens if the text is generated and not input as is? What happens with commands in the stream that do something with the following tokens?

A Lua-based solution could look as follows:

```
\def\Words#1{\directlua 0
  for s in unicode.utf8.gmatch("#1", "([^\ ])") do
    tex.sprint(string.upper(
      s:sub(1,1)) .. s:sub(2))
  end
}
```

But there is no real advantage here, apart from the fact that less code is needed. We still operate on the input and therefore we need to look to a different kind of solution: operating on the node list.

```
function CapitalizeWords(head)
  local done = false
  local glyph = node.id("glyph")
  for start in node.traverse_id(glyph,head) do
    local prev, next = start.prev, start.next
    if prev and prev.id == kern
      and prev.subtype == 0 then
      prev = prev.prev
    end
    if next and next.id == kern
      and next.subtype == 0 then
      next = next.next
    end
    if (not prev or prev.id ~= glyph)
      and next and next.id == glyph then
      done = upper(start)
    end
  end
  return head, done
end
```

A node list is a forward-linked list. With a helper function in the `node` library we can loop over such lists. Instead of traversing we can use a regular while loop, but it is probably less efficient in this case. But how to apply this function to the relevant part of the input? In Lua \TeX there are several callbacks that operate on the horizontal lists and we can use one of them to plug in this function. However, in that case the function is applied to probably more text than we want.

The solution for this is to assign attributes to the range of text which a function is intended to take care of. These attributes (there can be many) travel with the nodes. This is also a reason why such code normally is not written by end users, but by macro package writers: they need to provide the frameworks where you can plug in code. In Con \TeX t we have several such mechanisms and therefore

in MkIV this function looks (slightly simplified) as follows:

```
function cases.process(namespace,attribute,head)
  local done, actions = false, cases.actions
  for start in node.traverse_id(glyph,head) do
    local attr = has_attribute(start,attribute)
    if attr and attr > 0 then
      unset_attribute(start,attribute)
      local action = actions[attr]
      if action then
        local _, ok = action(start)
        done = done and ok
      end
    end
  end
  return head, done
end
```

Here we check attributes (these are set on the \TeX side) and we have all kind of actions that can be applied, depending on the value of the attribute. Here the function that does the actual uppercasing is defined somewhere else. The `cases` table provides us a namespace; such namespaces need to be coordinated by macro package writers.

This approach means that the macro code looks completely different; in pseudo code:

```
\def\Words#1{\setattribute<cases>
  <somevalue>#1}}
```

Or alternatively:

```
\def\StartWords{\begingroup\setattribute<cases>
  <somevalue>}
\def\StopWords {\endgroup}
```

Because starting a paragraph with a group can have unwanted side effects (such as `\everypar` being expanded inside a group) a variant is:

```
\def\StartWords{\setattribute<cases><somevalue>}
\def\StopWords {\resetattribute<cases>}
```

So, what happens here is that the user sets an attribute using some high level command, and at some point during the transformation of the input into node lists, some action takes place. At that point commands, expansion and the like no longer can interfere.

In addition to some infrastructure, macro packages need to carry some knowledge, just as with the `\uccode` used in `\uppercase`. The `upper` function in the first example looks as follows:

```
local function upper(start)
  local data, char = characters.data, start.char
  if data[char] then
    local uc = data[char].uccode
    if uc and
      fonts.tfm.id[start.font].characters[uc]
    then
```

```

    start.char = uc
    return true
end
end
return false
end

```

Such code is really macro package dependent: LuaT_EX provides only the means, not the solutions. In ConT_EXt we have collected information about characters in a `data` table in the `characters` namespace. There we have stored the uppercase codes (`uccode`). The `fonts` table, again ConT_EXt specific, keeps track of all defined fonts and before we change the case, we make sure that this character is present in the font. Here `id` is the number by which LuaT_EX keeps track of the used fonts. Each glyph node carries such a reference.

In this example, eventually we end up with more code than in T_EX, but the solution is much more robust. Just imagine what would happen when in the T_EX solution we would have:

```
\Words{\framed[offset=3pt]{hello world}}
```

It simply does not work. On the other hand, the Lua code never sees T_EX commands, it only sees the two words represented by glyph nodes and separated by glue.

Of course, there is a danger when we start opening T_EX’s core features. Currently macro packages know what to expect, they know what T_EX can and cannot do, and macro writers have exploited every corner of T_EX, even the darkest ones. While the dirty tricks in *The T_EXbook* had an educational purpose, those of users sometimes have obscene traits. If we just stick to the trickery introduced for parsing input, converting this into that, doing some calculations, and the like, it will be clear that Lua is more than welcome. It may hurt to throw away thousands of lines of impressive code and replace it by a few lines of Lua but that’s the price the user pays for abusing T_EX. Eventually ConT_EXt MkIV will be a decent mix of Lua and T_EX code, and hopefully the solutions programmed in those languages are as clean as possible.

Of course we can discuss until eternity whether Lua is the best choice. Taco, Hartmut and I are pretty confident that it is, and in the couple of years that we have been working on LuaT_EX nothing has proved us wrong yet. We can fantasize about concepts, only to find out that they are impossible to implement or hard to agree on; we just go ahead using trial and error. We can talk over and over how opening up should be done, which is what the team does in a nicely closed and efficient loop, but at some

points decisions have to be made. Nothing is perfect, neither is LuaT_EX, but most users won’t notice it as long as it extends T_EX’s life and makes usage more convenient.

6 Groups

Users of T_EX and MetaPost will have noticed that both languages have their own grouping (scope) model. In T_EX grouping is focused on content: by grouping the macro writer (or author) can limit the scope to a specific part of the text or have certain macros live within their own world.

```
.1. \bgroup .2. \egroup .1.
```

Everything done at 2 is local unless explicitly told otherwise. This means that users can write (and share) macros with a small chance of clashes. In MetaPost grouping is available too, but variables explicitly need to be saved.

```
.1. begingroup; save p; path p; .2. endgroup .1.
```

After using MetaPost for a while this feels quite natural because an enforced local scope demands multiple return values which is not part of the macro language. Actually, this is another fundamental difference between the languages: MetaPost has (a kind of) functions, which T_EX lacks. In MetaPost you can write

```
draw origin for i=1 upto 10: ..(i,sin(i)) endfor;
```

but also:

```
draw some(0) for i=1 upto 10: ..some(i) endfor;
```

with

```
vardef some (expr i) =
  if i > 4 : i = i - 4 fi ;
  (i,sin(i))
enddef ;
```

The condition and assignment in no way interfere with the loop where this function is called, as long as some value is returned (a pair in this case).

In T_EX things work differently. Take this:

```
\count0=1
\message{\advance\count0 by 1 \the\count0}
\the\count0
```

The terminal will show:

```
\advance \count 0 by 1 1
```

At the end the counter still has the value 1. There are quite a few situations like this, for instance when data such as a table of contents has to be written to a file. You cannot write macros where such calculations are done, hidden away, and only the result is seen.

The nice thing about the way Lua is presented to the user is that it permits the following:

```
\count0=1
\message{\directlua0{%
  tex.count[0] = tex.count[0] + 1}%
  \the\count0}
\the\count0
```

This will report 2 to the terminal and typeset a 2 in the document. Of course this does not solve everything, but it is a step forward. Also, compared to \TeX and MetaPost, grouping is done differently: there is a `local` prefix that makes variables (and functions are variables too) local in modules, functions, conditions, loops, etc. The Lua code in this article contains such locals.

7 An example: XML

In practice most users will use a macro package and so, if a user sees \TeX , he or she sees a user interface, not the code behind it. As such, they will also not encounter the code written in Lua that handles, for instance, fonts or node list manipulations. If a user sees Lua, it will most probably be in processing actual data. Therefore, in this section I will give an example of two ways to deal with XML: one more suitable for traditional \TeX , and one inspired by Lua. It demonstrates how the availability of Lua can result in different solutions for the same problem.

7.1 MkII: stream-based processing

In Con \TeX t MkII, the version that deals with pdf \TeX and X \TeX , we use a stream-based XML parser, written in \TeX . Each `<` and `&` triggers a macro that then parses the tag and/or entity. This method is quite efficient in terms of memory but the associated code is not simple because it has to deal with attributes, namespaces and nesting.

The user interface is not that complex, but involves quite a few commands. Take for instance the following XML snippet:

```
<document>
  <section>
    <title>Whatever</title>
    <p>some text</p>
    <p>some more</p>
  </section>
</document>
```

When using Con \TeX t commands, we can imagine the following definitions:

```
\defineXMLenvironment[document]
  {\starttext} {\stoptext}
\defineXMLargument [title]
  {\section}
\defineXMLenvironment[p]
  {\ignorespaces}{\par}
```

When attributes have to be dealt with, for in-

stance a reference to this section, things quickly start looking more complex. Also, users need to know what definitions to use in situations like this:

```
<table>
  <tr><td>first</td> ... <td>last</td></tr>
  <tr><td>left</td> ... <td>right</td></tr>
</table>
```

Here we cannot be sure that a cell does not contain a nested table, which is why we need to define the mapping as follows:

```
\defineXMLnested[table]{\bTABLE} {\eTABLE}
\defineXMLnested[tr]   {\bTR}   {\eTR}
\defineXMLnested[td]   {\bTD}   {\eTD}
```

The `\defineXMLnested` macro is rather messy because it has to collect snippets and keep track of the nesting level, but users don't see that code, they just need to know when to use what macro. Once it works, it keeps working.

Unfortunately mappings from source to style are never that simple in real life. We usually need to collect, filter and relocate data. Of course this can be done before feeding the source to \TeX , but MkII provides a few mechanisms for that too. For instance, to reverse the order you can do this:

```
<article>
  <title>Whatever</title>
  <author>Someone</author>
  <p>some text</p>
</article>
\defineXMLenvironment[article]
  {\defineXMLsave[author]}
  {\blank author: \XMLflush{author}}
```

This will save the content of the `author` element and flush it when the end tag `article` is seen. So, given previous definitions, we will get the title, some text and then the author. You may argue that instead we should use for instance XSLT but even then a mapping is needed from the XML to \TeX , and it's a matter of taste where the burden is put.

Because Con \TeX t also wants to support standards like MathML, there are some more mechanisms but these are hidden from the user. And although these do a good job in most cases, the code associated with the solutions has never been satisfying.

Supporting XML this way is doable, and Con \TeX t has used this method for many years in fairly complex situations. However, now that we have Lua available, it is possible to see if some things can be done more simply (or differently).

7.2 MkIV: tree-based processing

After some experimenting I decided to write a full blown XML parser in Lua, but contrary to the

stream-based approach, this time the whole tree is loaded in memory. Although this uses more memory than a streaming solution, in practice the difference is not significant because often in MkII we also needed to store whole chunks.

Loading XML files in memory is very fast and once it is done we can have access to the elements in a way similar to XPath. We can selectively pipe data to T_EX and manipulate content using T_EX or Lua. In most cases this is faster than the stream-based method. An interesting fact is that we can do this without linking to existing XML libraries, and as a result we are pretty independent.

So how does this look from the perspective of the user? Say that we have the simple article definition stored in `demo.xml`.

```
<?xml version = '1.0'?>
<article>
  <title>Whatever</title>
  <author>Someone</author>
  <p>some text</p>
</article>
```

This time we associate so-called setups with the elements. Each element can have its own setup, and we can use expressions to assign them. Here we have just one such setup:

```
\startxmlsetups xml:document
  \xmlsetsetup{main}{article}{xml:article}
\stopxmlsetups
```

When loading the document it will automatically be associated with the tag `main`. The previous rule associates the setup `xml:article` with the `article` element in tree `main`. We register this setup so that it will be applied to the document after loading:

```
\xmlregistersetup{xml:document}
```

and the document itself is processed with (the empty braces are an optional setup argument):

```
\xmlprocessfile{main}{demo.xml}{}
```

The setup `xml:article` can look as follows:

```
\startxmlsetups xml:article
  \section{\xmltext{#1}{/title}}
  \xmlall{#1}{!(title|author)}
  \blank author: \xmltext{#1}{/author}
\stopxmlsetups
```

Here `#1` refers to the current node in the XML tree, in this case the root element, `article`. The second argument of `\xmltext` and `\xmlall` is a path expression, comparable to XPath: `/title` means: the `title` element anchored to the current root (`#1`), and `!(title|author)` is the negation of (complement to) `title` or `author`. Such expressions can be more complex than the one above, for instance:

```
\xmlfirst{#1}{/one/(alpha|beta)/two/text()}
```

which returns the content of the first element that satisfies one of the paths (nested tree):

```
/one/alpha/two
/one/beta/two
```

There is a whole bunch of commands like `\xmltext` that filter content and pipe it into T_EX. These are calling Lua functions. This is no manual, so we will not discuss them here. However, it is important to realize that we have to associate setups (consider them free formatted macros) with at least one element in order to get started. Also, XML inclusions have to be dealt with before assigning the setups. These are simple one-line commands. You can also assign defaults to elements, which saves some work.

Because we can use Lua to access the tree and manipulate content, we can now implement parts of XML handling in Lua. An example of this is dealing with so-called Cals tables. This is done in approximately 150 lines of Lua code, loaded at runtime in a module. This time the association uses functions instead of setups and those functions will pipe data back to T_EX. In the module you will find:

```
\startxmlsetups xml:cals:process
  \xmlsetfunction {\xmlldocument} {cals:table}
  {lxml.cals.table}
\stopxmlsetups

\xmlregistersetup{xml:cals:process}
\xmlregisterns{cals}{cals}
```

These commands tell MkIV that elements with a namespace specification that contains `cals` will be remapped to the internal namespace `cals` and the setup associates a function with this internal namespace.

By now it will be clear that from the perspective of the user Lua is hardly visible. Sure, he or she can deduce that deep down some magic takes place, especially when you run into more complex expressions like this (the `@` denotes an attribute):

```
\xmlsetsetup
  {main} {item[@type='mpctext' or @type='mrtext']}
  {questions:multiple:text}
```

Such expressions resemble XPath, but can go much further, just by adding more functions to the library.

```
item[position() > 2 and position() < 5
  and text() == 'ok']
item[position() > 2 and position() < 5
  and text() == upper('ok')]
item[@n=='03' or @n=='08']
item[number(@n)>2 and number(@n)<6]
item[find(text(),'ALSO')]
```

Just to give you an idea, in the module that implements the parser you will find definitions that match the function calls in the above expressions.

```
xml.functions.find = string.find
xml.functions.upper = string.upper
xml.functions.number = tonumber
```

So much for the different approaches. It's up to the user what method to use: stream-based MkII, tree-based MkIV, or a mixture.

8 T_EX–Lua in conversation

The main reason for taking XML as an example of mixing T_EX and Lua is in that it can be a bit mind-boggling if you start thinking of what happens behind the scenes. Say that we have

```
<?xml version = '1.0'?>
<article>
  <title>Whatever</title>
  <author>Someone</author>
  <p>some <b>bold</b> text</p>
</article>
```

and we use the setup shown before with `article`.

At some point, we are done with defining setups and load the document. The first thing that happens is that the list of manipulations is applied: file inclusions are processed first, setups and functions are assigned next, maybe some elements are deleted or added, etc. When that is done we serialize the tree to T_EX, starting with the root element. When piping data to T_EX we use the current catcode regime; linebreaks and spaces are honored as usual.

Each element can have a function (command) associated and when this is the case, control is given to that function. In our case the root element has such a command, one that will trigger a setup. And so, instead of piping content to T_EX, a function is called that lets T_EX expand the macro that deals with this setup.

However, that setup itself calls Lua code that filters the title and feeds it into the `\section` command, next it flushes everything except the title and author, which again involves calling Lua. Last it flushes the author. The nested sequence of events is as follows:

```
lua: Load the document and apply setups and
the like.
lua: Serialize the article element, but since
there is an associated setup, tell TEX to
expand that one instead.
tex: Execute the setup, first expand the
\section macro, but its argument is a
call to Lua.
lua: Filter title from the subtree un-
```

```
der article, print the content to
TEX and return control to TEX.
tex: Tell Lua to filter the paragraphs i.e.
skip title and author; since the b
element has no associated setup (or
whatever) it is just serialized.
lua: Filter the requested elements and
return control to TEX.
tex: Ask Lua to filter author.
lua: Pipe author's content to TEX.
tex: We're done.
lua: We're done.
```

This is a very simple case. In my daily work I am dealing with rather extensive and complex educational documents where in one source there is text, math, graphics, all kind of fancy stuff, questions and answers in several categories and of different kinds, to be reshuffled or not, omitted or combined. So there we are talking about many more levels of T_EX calling Lua and Lua piping to T_EX, etc. To stay in T_EX speak: we're dealing with one big ongoing nested expansion (because Lua calls `expand`), and you can imagine that this somewhat stresses T_EX's input stack, but so far I have not encountered any problems.

9 Final remarks

Here I discuss several possible applications of Lua in T_EX. I didn't mention yet that because LuaT_EX contains a scripting engine plus some extra libraries, it can also be used purely for that. This means that support programs can now be written in Lua and that we need no longer depend on other scripting engines being present on the system. Consider this a bonus.

Usage in T_EX can be categorized in four ways:

1. Users can use Lua for generating data, do all kind of data manipulations, maybe read data from file, etc. The only link with T_EX is the print function.
2. Users can use information provided by T_EX and use this when making decisions. An example is collecting data in boxes and use Lua to do calculations with the dimensions. Another example is a converter from MetaPost output to PDF literals. No real knowledge of T_EX's internals is needed. The MkIV XML functionality discussed before demonstrates this: it's mostly data processing and piping to T_EX. Other examples are dealing with buffers, defining character mappings, and handling error messages, verbatim . . . the list is long.
3. Users can extend T_EX's core functionality. An

example is support for OpenType fonts: Lua-T_EX itself does not support this format directly, but provides ways to feed T_EX with the relevant information. Support for OpenType features demands manipulating node lists. Knowledge of internals is a requirement. Advanced spacing and language specific features are made possible by node list manipulations and attributes. The alternative `\Words` macro is an example of this.

4. Users can replace existing T_EX functionality. In MkIV there are numerous examples of this, for instance all file I/O is written in Lua, including reading from zip files and remote locations. Loading and defining fonts is also under Lua control. At some point MkIV will provide dedicated splitters for multicolumn typesetting and probably also better display

spacing and display math splitting.

The boundaries between these categories are not set in stone. For instance, support for image inclusion and MPlib in ConT_EXt MkIV sits between categories 3 and 4. Categories 3 and 4, and probably also 2, are normally the domain of macro package writers and more advanced users who contribute to macro packages. Because a macro package has to provide some stability it is not a good idea to let users mess around with all those internals, due to potential interference. On the other hand, normally users operate on top of a kernel using some kind of API, and history has proved that macro packages are stable enough for this.

Sometime around 2010 the team expects Lua-T_EX to be feature complete and stable. By that time I can probably provide a more detailed categorization.

docx2tex: Word 2007 to T_EX

Krisztián Pócza

Eötvös Loránd University, Faculty of Informatics, Department of Programming Languages and Compilers,
Pázmány Péter sétány 1/C. H-1117, Budapest, Hungary

kpocza (at) kpocza dot net

<http://kpocza.net/>

Mihály Biczó

Eötvös Loránd University, Faculty of Informatics, Department of Programming Languages and Compilers,
Pázmány Péter sétány 1/C. H-1117, Budapest, Hungary

mihaly.biczo (at) t-online dot hu

<http://avalon.inf.elte.hu/personal/hdbiczo/>

Zoltán Porkoláb

Eötvös Loránd University, Faculty of Informatics, Department of Programming Languages and Compilers,
Pázmány Péter sétány 1/C. H-1117, Budapest, Hungary

gsd (at) elte dot hu

<http://gsd.web.elte.hu/>

Abstract

Docx2tex is a small command line tool to support users of Word 2007 to publish documents when typography is important or only papers produced by T_EX are accepted. Behind the scenes, docx2tex uses common technologies to interpret Word 2007 OOXML format without utilizing the API of Word 2007. Docx2tex is published as a free and open source utility that is accessible and extensible by everyone. The source code and the binary executable of the application can be downloaded from <http://codeplex.com/docx2tex/>. This paper was originally written in Word 2007 and later converted to T_EX using docx2tex.

1 Introduction

There are two general methods to produce human readable and printable digital documents:

1. Using a WYSIWYG word processor
2. Using a typesetting system

Each of them has its own advantages and disadvantages; therefore each of them has many use cases where one is better than the other.

WYSIWYG [1] is an acronym for the term *What You See Is What You Get* that originates from the late '70s. WYSIWYG editors are usually favored by everyday computer users whose aim is to produce good-looking documents in a fast and straightforward way exploiting the rich formatting capabilities of such systems. WYSIWYG editors and word processors ensure that the printed version of the document will be the same as the document that is visible on the screen during editing. The first WYSIWYG word processor called Bravo was created at Xerox by Charles Simonyi, who is the inventor of intentional programming. In 1981 Simonyi left Xe-

rox and joined Microsoft where he created Microsoft Word [2, 3], the first and still the most popular word processor. Word is capable of producing simple and also complex documents, including those with many mathematical symbols. Another important feature of Word is *Track Changes* that supports team work. Using Track Changes any of the team members can modify the document while these modifications are tracked and can be accepted or refused by the team leader.

Typesetting is the process of putting characters of different types in their correct place on the paper or screen. Before electronic typesetting systems became widely used, printed materials had been produced by compositors who worked by hand or using special machines. The aim of typesetting systems is to create high-quality output of materials that may contain complex mathematical formulae and complex figures. Similarly, electronic typesetting systems follow this goal and produce high quality, device independent output. The most popular

typesetting system is T_EX [4] created by Donald E. Knuth. T_EX is mainly used by researchers and individuals whose aim is to achieve the best quality output without sacrificing platform or device independence. The users of T_EX use a special and extensible DSL (Domain Specific Language) that was designed to solve complex typesetting problems, produce books containing hundreds of pages, and more.

There is a big gap between these systems because each tries to satisfy different demands, namely: produce common documents quickly even in a group setting vs. achieve the best quality and typographically correct printout. To bridge this difference there are both commercial and non-commercial tools that support conversion from Word or other WYSIWYG formats to T_EX (and back). The first direction, converting from WYSIWYG (Word) formats to T_EX, has perhaps more frequent usage because many users edit the original text in Word for the sake of simplicity and efficiency, and later convert it to T_EX by hand in order to ensure quality.

The problems with present conversion applications include the following:

1. many of them are available only as proprietary tools;
2. thus they have limitations (running times or page limit) when not purchased;
3. they support only the old, binary Word or Rich Text document format (*.doc*, *.rtf*); and/or
4. they use the Word's COM API to process documents, which makes them complex.

In this paper we present an open source and free solution that is capable of handling the new and open Word 2007 *.docx* format natively by using standard technologies without using the COM API of Word and without even installing Word. In this article the current features are presented along with further development directions.

2 Existing solutions

It has always been challenging to convert proprietary, binary or any other document formats to T_EX. Because Word is the most common editor, many tools try to convert from Word documents. One of these tools is the proprietary *Word2TeX* [5], which makes Word capable of saving documents in T_EX format. This tool is embedded into Word, has an evaluation period and can be purchased in different license packages. A similarly featured tool named *Word-to-Latex* [6] does not provide sources, though it is available at no cost.

Another possibility is to use OpenOffice.org [7], which is capable of reading Word documents and

also saving them in T_EX format. It is a free and open source application; however, it interprets the binary data of Word documents.

Rtf2latex2e [8] is the most recent solution that translates *.rtf* files to T_EX. It is a free and open source application.

3 Technology

In this section we will enumerate and then briefly review the technologies that are used in docx2tex and show how they cooperate.

The technologies used are the following:

1. Office Open XML (ECMA 376 Standard [9], recently approved as an ISO Standard), the default format of Word 2007. In brief: OOXML.
2. Microsoft .NET 3.0 (CLI is ECMA 335 Standard [10] and ISO/IEC 23271:2006 Standard [11]).
3. ImageMagick [12] to convert images.

OOXML files are simply XML and media files compressed using ZIP. Docx2tex uses Microsoft .NET 3.0 to open and unzip OOXML Word 2007 (*.docx* extension) documents. Microsoft .NET 3.0 has some special classes in the *System.IO.Packaging* namespace that facilitate opening and unzipping OOXML files and also provide an abstraction to represent the included XML and media files as packages. The operations performed by this component are described by the object line called *OOXML depackaging* in Figure 1.

The most important component of docx2tex is the *Core XML Engine* that implements the basic conversion from XML files to T_EX. The Core Engine is responsible for reading and processing the XML data of the OOXML documents that is served by the OOXML depackaging component. The Core Engine identifies parts of the OOXML document and processes the contents of these parts (paragraphs, runs, tables, image references, numberings, ...). It is not responsible for processing parts of the OOXML document that are available through a relation. For those, docx2tex has a set of internal *helper functions* that are responsible for driving the processing of related entities such as image conversion, special styling and resolving the properties of numbered lists.

When an image reference is found in the XML, ImageMagick is called to produce EPS files from the original image files. The resulting EPS files can be embedded easily into T_EX documents.

We support the exact output produced by Word 2007; other output variations saved by third party applications that may differ from the ECMA 376 Standard are not supported.

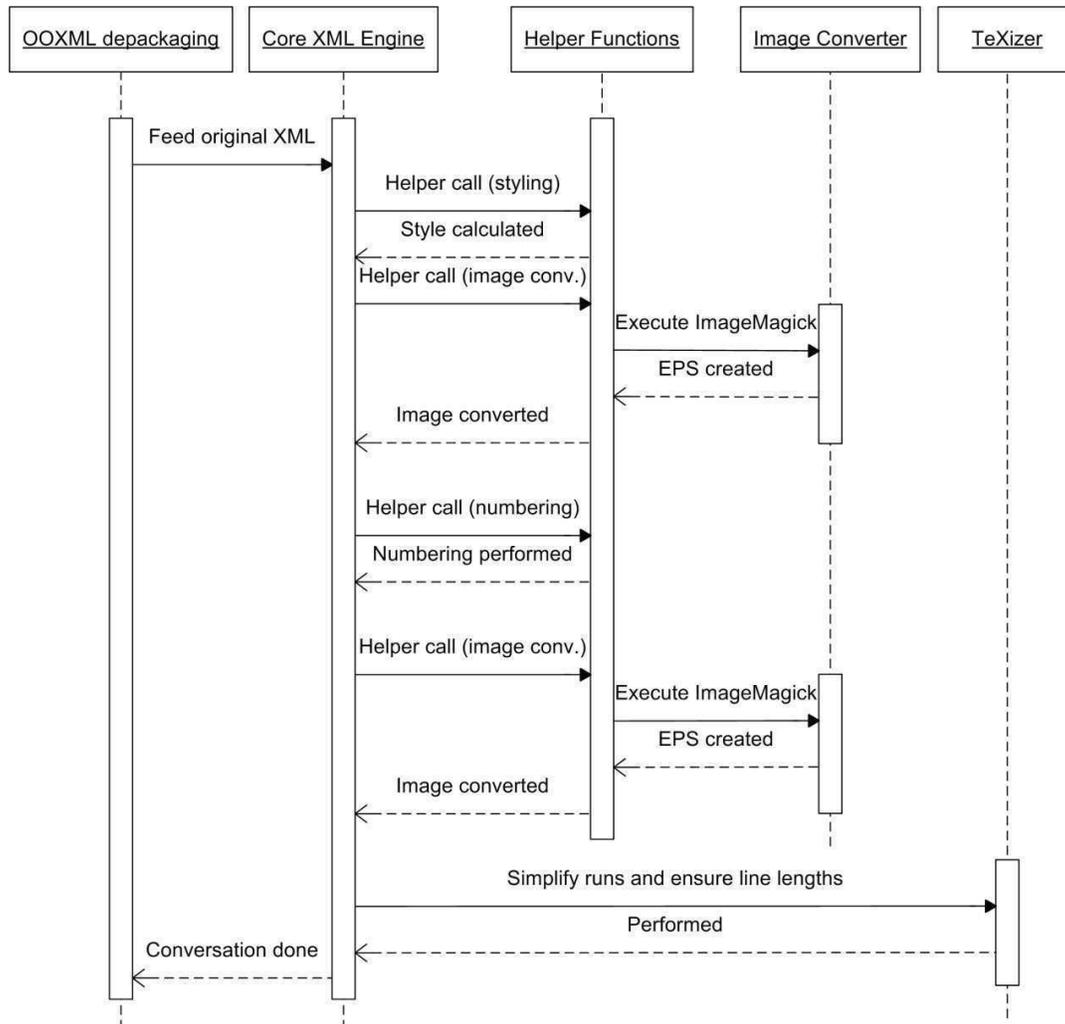


Figure 1: UML sequence diagram

In the next sections the structure of OOXML will be briefly discussed; first, let us review the idea of *runs*. A *run* is a piece of text which also has some style specification. Runs are placed and removed dynamically while the Word document is edited. A sentence or even a word can be divided into more than one run with the same style. The component called *TeXizer* is responsible to join runs having the same style to a simple run in the outgoing \TeX code and break the source line length at some predefined value (default is 72).

The previous description is illustrated by the UML sequence diagram in Figure 1.

4 Features of docx2tex

In this section we list the supported and the unsupported features of docx2tex.

4.1 Supported features

Docx2tex supports the following features of Word 2007 and \TeX :

1. Normal text
2. Italic, bold, underlined, stroked, small capitals, ...
3. Left, right, center aligned text
4. Headings and sections, three levels
5. Verbatim text
6. Style mapping
7. Simple tables
8. Line and page breaks
9. Numbered and bulleted lists
10. Multilevel lists and continuous numbered lists
11. Figure, table and listing captions
12. Cross references to captions and headings

13. Image conversion from various formats (including .png, .jpeg, .emf, etc.) to .eps
 14. Substitution of special characters (e.g. \, #, {, }, [,], %, &, ~, ...)
 15. Text boxes
 16. Basic math formulae, Word Equations support
- docx2tex supports normal and special text styles and also text alignments. We support heading styles *Heading1*, *Heading2*, and *Heading3* that convert to `\section`, `\subsection`, and `\subsubsection` respectively. Word does not support verbatim text while T_EX does. To work around this deficiency, text marked with *Verbatim* style is converted to verbatim text surrounded by `\begin{verbatim}` and `\end{verbatim}`. There are many cases where we are required to use different styles for headings or even verbatim.

Only simple, left-aligned tables are supported. Both numbered and bulleted lists are supported, including mixing and nesting. Continuous lists are also supported using the `\setcounter`, `\enumi`, and `\theenumi` commands. Figure, table and listings captions are recognized and we support referencing them; likewise with heading references. Image references are resolved and the images (mainly .png and .jpeg) embedded in the OOXML documents are converted to .eps. The width and height properties are queried and the same properties are used in the resulting T_EX documents. Some special T_EX characters are also resolved and escaped in the resulting T_EX document. Text found in *Text Boxes* of Word documents are also processed and inserted in place of the resulting T_EX document.

4.2 Unsupported features

Docx2tex has only basic Word Equations (mathematical formulae) support at the time of publishing this paper. We plan to add more support for Word 2007 Equations and Drawings that can be converted to T_EX mathematical formulas and xfigs respectively. Both of them are described in XML format therefore our standard solution can be extended without introducing other technologies.

5 A complex example

In this section we will show a complex example broken into significant parts that introduces the most important features of docx2tex.

5.1 The structure of the OOXML ZIP package

To inspect the content of an OOXML ZIP package we first unzip the contents of our Word 2007 document to a directory and get a recursive directory listing:

```
PS C:\Phd\conf\2008_4_tex\example.docx>
ls -Recu |% {$_ .FullName.SubString(30)}
example.docx\customXml
example.docx\docProps
example.docx\word
example.docx\_rels
example.docx\[Content_Types].xml
example.docx\customXml\_rels
example.docx\customXml\item1.xml
example.docx\customXml\itemProps1.xml
example.docx\docProps\app.xml
example.docx\docProps\core.xml
example.docx\word\media
example.docx\word\theme
example.docx\word\_rels
example.docx\word\document.xml
example.docx\word\fontTable.xml
example.docx\word\numbering.xml
example.docx\word\settings.xml
example.docx\word\styles.xml
example.docx\word\webSettings.xml
example.docx\word\media\image1.jpeg
example.docx\word\theme\theme1.xml
example.docx\word\_rels\document.xml.rels
example.docx\_rels\.rels
```

The most important part is the `document.xml` file that contains the document itself and references to external items. The `numbering.xml` file specifies the style of the numbered or bulleted lists contained in `document.xml`. The `styles.xml` file specifies information about the styles used in the document. Under the `media` subdirectory the embedded images can be found (`image1.jpeg` in our example).

5.1.1 Structure of the document

The text in the `document.xml` file is grouped into *paragraphs*. Every segment of the document is a paragraph (normal text, heading texts, images, etc.) except for some special elements like tables. Paragraphs are further divided into *runs*. A *run* is a piece of text that also has some style specification.

5.2 Text conversion

The most fundamental feature of tools like docx2tex is the ability to interpret text runs with many basic styling properties and convert them to T_EX format. Consider the following example sentence: This is a sentence *that contains* text *with* ~~different~~ *formatting*.

This sentence looks like the following in OOXML:

```
<w:p w:rsidR="004F5706" w:rsidRDefault=
    "004F5706" w:rsidP="004F5706">
  <w:r w:rsidRPr="0030655B">
    <w:t xml:space="preserve">
```

```

                This is a </w:t>
</w:r>
<w:r w:rsidRPr="0030655B">
  <w:rPr>
    <w:i/>
    <w:vertAlign w:val="superscript"/>
  </w:rPr>
  <w:t>sentence</w:t>
</w:r>
<w:r w:rsidRPr="0030655B">
  <w:rPr>
    <w:b/>
    <w:i/>
  </w:rPr>
  <w:t xml:space="preserve"> that</w:t>
</w:r>
<w:r w:rsidRPr="0030655B">
  <w:t xml:space="preserve"> </w:t>
</w:r>
<w:r w:rsidRPr="0030655B">
  <w:rPr>
    <w:u w:val="single"/>
  </w:rPr>
  <w:t>contains</w:t>
</w:r>
<w:r w:rsidRPr="0030655B">
  <w:t xml:space="preserve"> text </w:t>
</w:r>
<w:r w:rsidRPr="0030655B">
  <w:rPr>
    <w:b/>
    <w:i/>
    <w:u w:val="single"/>
  </w:rPr>
  <w:t>with</w:t>
</w:r>
<w:r w:rsidRPr="0030655B">
  <w:t xml:space="preserve"> </w:t>
</w:r>
<w:r w:rsidRPr="0030655B">
  <w:rPr>
    <w:strike/>
  </w:rPr>
  <w:t>different</w:t>
</w:r>
<w:r w:rsidRPr="0030655B">
  <w:t xml:space="preserve"> </w:t>
</w:r>
<w:r w:rsidRPr="0030655B">
  <w:rPr>
    <w:vertAlign w:val="subscript"/>
  </w:rPr>
  <w:t>formatting</w:t>
</w:r>

```

```

  <w:r w:rsidRPr="0030655B">
    <w:t>.</w:t>
  </w:r>
</w:p>

```

The XML node `<w:p>` and `</w:p>` encloses a paragraph while `<w:r>` and `</w:r>` encloses a run. A run contains a range of text (between `<w:t>` and `</w:t>`) and may contain some formatting between `<w:rPr>` and `</w:rPr>` (e.g. `<w:b/>` means bold, while `<w:i/>` means italic font style).

The \TeX output generated by `docx2tex` of the previous sentence looks like the following:

```

This is a \textit{${sentence}}\textbf{%
\textit{that}} \underline{contains} text
\textbf{\textit{\underline{with}}}\ \sout{%
different} $_{formatting}$.
```

5.3 Headings and verbatim

Headings and verbatim are handled the same way because they can be identified in the source document by examining paragraph level styles.

Consider the following OOXML fragment that describes a first level heading:

```

<w:p w:rsidR="004F5706" w:rsidRPr=
  "0030655B" w:rsidRDefault="004F5706"
  w:rsidP="000136DF">
  <w:pPr>
    <w:pStyle w:val="Heading1"/>
  </w:pPr>
  <w:bookmarkStart w:id="0" w:name=
    "_Ref186547407"/>
  <w:r w:rsidRPr="0030655B">
    <w:t>Heading text</w:t>
  </w:r>
  <w:bookmarkEnd w:id="0"/>
</w:p>

```

The `<w:pStyle w:val="Heading1"/>` node specifies that a first level heading begins, while the contained node `<w:bookmarkStart w:id="0" w:name="_Ref186547407"/>` identifies a unique internal reference (bookmark) to the heading that can be cross-referenced from any part of the document. For each referenceable item Word generates an ugly unique number prefixed with `_Ref` as an identifier (in our example, `_Ref186547407`).

The generated \TeX output is the following (line break is editorial):

```

\section{Heading text}\label{section:
_Ref186547407}
```

It is possible to map custom styles to certain \TeX elements. The special mappings are loaded from a file with the same name having the extension `.paraStyleName` (that is, a file `example.docx` has

the mapping file `example.paraStyleName`). The Word 2007 styles appearing on the right side of these equations have to be the `w:styleId` attribute of one of the styles found in the `styles.xml` file (names are case sensitive).

Here is a listing to help understand the format of the `.paraStyleName` files:

```
section=Myheading1
subsection=Myheading2
subsubsection=Myheading3
verbatim=Myverbatim
```

5.4 Images and cross references

In OOXML, images are described in a very complex and loose way; there is no space here to show the original XML fragment. Instead we show only the generated T_EX code:

```
\begin{figure}[h]
\centering
\includegraphics[width=10.52cm,height=
8.41cm]{media/image1.eps}
\caption{\label{figure:_Ref186544261}:
Figure caption}
\end{figure}
```

The image is centered and the width and the height of the image are preserved. `image1.jpeg` was converted to `image1.eps` and the file was saved in the `media` subdirectory. When the image has a caption then it is also added to the output so that it can be referenced.

Reference to the previous figure is described in OOXML in the following form:

```
<w:p w:rsidR="004F5706" w:rsidRPr="
0030655B" w:rsidRDefault="
004F5706" w:rsidP="004F5706">
  <w:pPr>
    <w:keepNext/>
  </w:pPr>
  <w:r w:rsidRPr="0030655B">
    <w:t xml:space="preserve">Reference to
      the figure: </w:t>
  </w:r>
  <w:r w:rsidR="007A289D">
    <w:fldChar w:fldCharType="begin"/>
  </w:r>
  <w:r w:rsidR="006B4DA8">
    <w:instrText xml:space="preserve">
      REF _Ref186544261 \h </w:instrText>
  </w:r>
  <w:r w:rsidR="007A289D">
    <w:fldChar w:fldCharType="separate"/>
  </w:r>
  <w:r w:rsidR="006B4DA8">
```

```
<w:t xml:space="preserve">Figure</w:t>
</w:r>
<w:r w:rsidR="006B4DA8">
  <w:rPr>
    <w:noProof/>
  </w:rPr>
  <w:t>1</w:t>
</w:r>
<w:r w:rsidR="007A289D">
  <w:fldChar w:fldCharType="end"/>
</w:r>
</w:p>
```

The generated T_EX code is simple (it can be seen that the *Figure 1* text has been omitted from the output because it is internal to Word):

Reference to the figure: `\ref{figure:_Ref186544261}`.

Referencing tables is the same as for figures and sections.

5.5 Lists and tables

There are two main categories of lists supported by OOXML and Word 2007: numbered and bulleted. Both types of lists are allowed to have multiple levels, and numbered lists can be continuous, meaning that the list can be interrupted by some other content and then continued at the same number.

The first item of a numbered list looks like the following:

```
<w:p w:rsidR="004F5706" w:rsidRPr="
0030655B" w:rsidRDefault="004F5706"
w:rsidP="004F5706">
  <w:pPr>
    <w:pStyle w:val="ListParagraph"/>
    <w:keepNext/>
    <w:numPr>
      <w:ilvl w:val="0"/>
      <w:numId w:val="1"/>
    </w:numPr>
  </w:pPr>
  <w:r w:rsidRPr="0030655B">
    <w:t>First</w:t>
  </w:r>
</w:p>
```

The nodes enclosed in `<w:numPr>` and `</w:numPr>` specify that we have a list. The `w:val` attributes of `w:numId` and `w:ilvl` specify numbering identifier and level parameters (style 1 at level 0 in our example). It may seem strange that the `w:numPr` nodes describe both numbered and bulleted lists. It is the numbering identifier and the level parameter that distinguishes between the two categories of lists. These parameters are defined in the `numbering.xml`

file that is processed by docx2tex. The above element is part of a complex multilevel numbered list:

```
\newcounter{numberedCntA}
\begin{enumerate}
\item First
\item Second
\item Third
\begin{enumerate}
\item First
\item Second
\item Third
\end{enumerate}
\end{enumerate}
\setcounter{numberedCntA}{\theenumi}
\end{enumerate}
```

When a previous list is continued, `\setcounter{enumi}{\thenumberedCntA}` is inserted after the `\begin{enumerate}` by docx2tex.

We omit showing a bulleted example since it differs only in the \TeX output: the *itemize* keyword is used instead of *enumerate*, and we do not have to maintain counters for continuous lists.

Docx2tex supports only simple tables, so no merged, divided or differently aligned table cells are possible, but the current features still allow docx2tex to be able to convert most tables.

Again there is not enough space here to show the OOXML version of a simple table. The generated \TeX output of a table with four cells is the following:

```
\begin{tabular}{|1|1|}
\hline
1 & 2 \\
\hline
3 & 4 \\
\hline
\end{tabular}
\caption{\label{table:_Ref186545972}:
caption}
\end{table}
```

5.6 Special characters

\TeX uses some special characters to place formatting commands to structure or change the appearance of text. When we want to place these special characters in running text they have to be described in a special way.

Consider the following set of special characters:

```
< > as'q ... # \ { } % ~ _ ^ & $ ""
```

These are described in OOXML in the following form:

```
<w:p w:rsidR="00AB630B" w:rsidRDefault=
"00AB630B" w:rsidP="00AB630B">
<w:r>
<w:t xml:space="preserve">&lt;&gt;
```

```
</w:t>
</w:r>
<w:proofErr w:type="spellStart"/>
<w:r>
<w:t>as'q</w:t>
</w:r>
<w:proofErr w:type="spellEnd"/>
<w:r>
<w:t xml:space="preserve"> ...# \ { }
</w:t>
</w:r>
<w:proofErr w:type="gramStart"/>
<w:r>
<w:t>% ~</w:t>
</w:r>
<w:proofErr w:type="gramEnd"/>
<w:r>
<w:t xml:space="preserve"> _ ^ & amp;
$ ""</w:t>
</w:r>
</w:p>
```

The resulting \TeX code is:

```
$$$ as'q ... \# \$\backslash$ \{ \} \% \~
\_ \^ \& \$ "\",
```

5.7 Math formulae

Math formulae (Word Equations) are described in a hierarchical XML structure inside the OOXML document. It can be easily walked by a recursive algorithm to create \TeX output. The name of the XML node that hosts mathematical formulae is `m:oMath`. While standard parts of the document are described by nodes that have the `w` XML namespace, Word Equations have the `m` namespace.

There is no space to show an OOXML fragment that holds a quite complex formula, but here is the formula and the resulting \TeX code that was generated by docx2tex:

$$B(v) \cong \frac{1}{\sqrt{2\pi}} \int_v^\infty e^{-\frac{t^2}{2}} dt$$

```
$B(v)\cong \frac{1}{\sqrt{2\pi}} \int_{\int v}^{\infty} e^{-\frac{t^2}{2}} dt$
```

6 A use case

Let us suppose the following scenario: Two authors decide to write a scientific article about their research topic and submit it to a conference or journal. First they split the proposed article into sections and assign each section to one of the authors. They start to work independently using Word 2007. After both of the authors finish they merge the resulting text into a single document. After that step the first author reads the whole document and makes changes using the Track Changes function of Word.

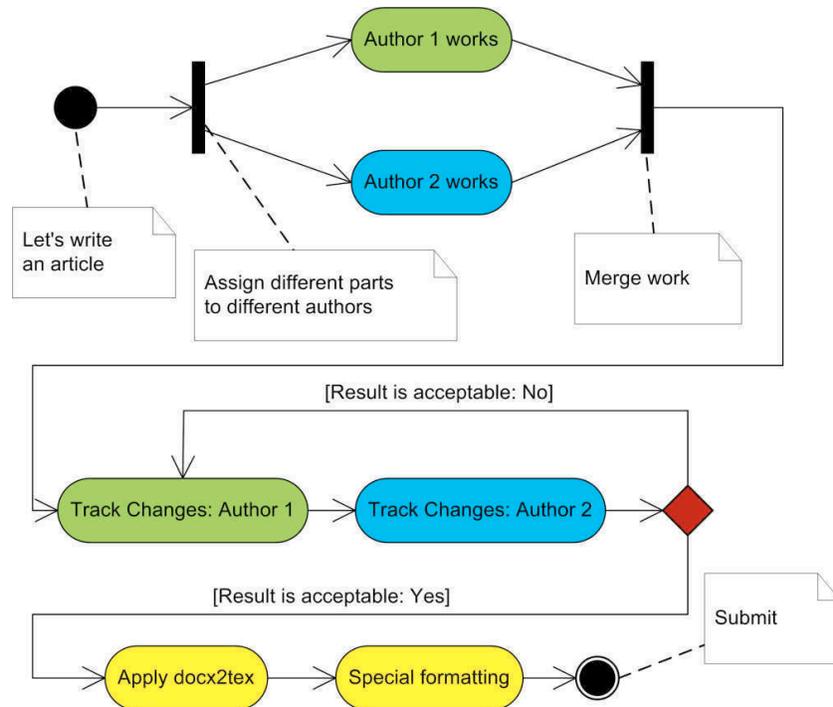


Figure 2: Workflow

The second author accepts or rejects the changes of the first author and also makes his own changes using the Track Changes function of Word. When the authors agree that the quality of the article is acceptable, they convert it to T_EX using docx2tex and apply special formatting required for the particular conference or journal. Now the article can be submitted.

This workflow is illustrated in Figure 2.

As can be seen we exploited the strengths of both the WYSIWYG Word 2007 system to support effective team work and the typesetting T_EX system to produce the best quality printout. The conversion between the file formats they use was performed using docx2tex. Note that currently docx2tex is able to do rough conversion and cannot apply special commands and styles.

Readers not familiar with the Track Changes function should consider Figure 3.

7 Conclusion and further work

In this article we introduced a tool called docx2tex that is dedicated to producing T_EX documents from Word 2007 OOXML documents. The main advantage of this solution over classical methods is that we process the bare XML content of OOXML packages instead of processing binary files or exploiting the capabilities of the COM API of Word, thus mak-

ing our solution more robust and usable.

We presented the main features of docx2tex, related primarily to text processing and formatting, structuring the document, handling images, tables and references. There are some important features that we consider worth implementing in the future:

1. More Equations support
2. Embedded vector graphical drawings
3. Configuration settings
4. Optional font and coloring support
5. Documentation
6. Automated installer

Multicolumn document handling and templating may also worth considering.

We published the application as free and open source software under the terms of the BSD [13] license, so anybody can use it royalty free and can add new features to the current feature set.

The source code and the binary of the application can be downloaded from <http://codeplex.com/docx2tex/>. If the reader would like to participate in the development of docx2tex, please contact the authors.

References

- [1] Wikipedia article on What You See Is What You Get. <http://en.wikipedia.org/wiki/WYSIWYG>

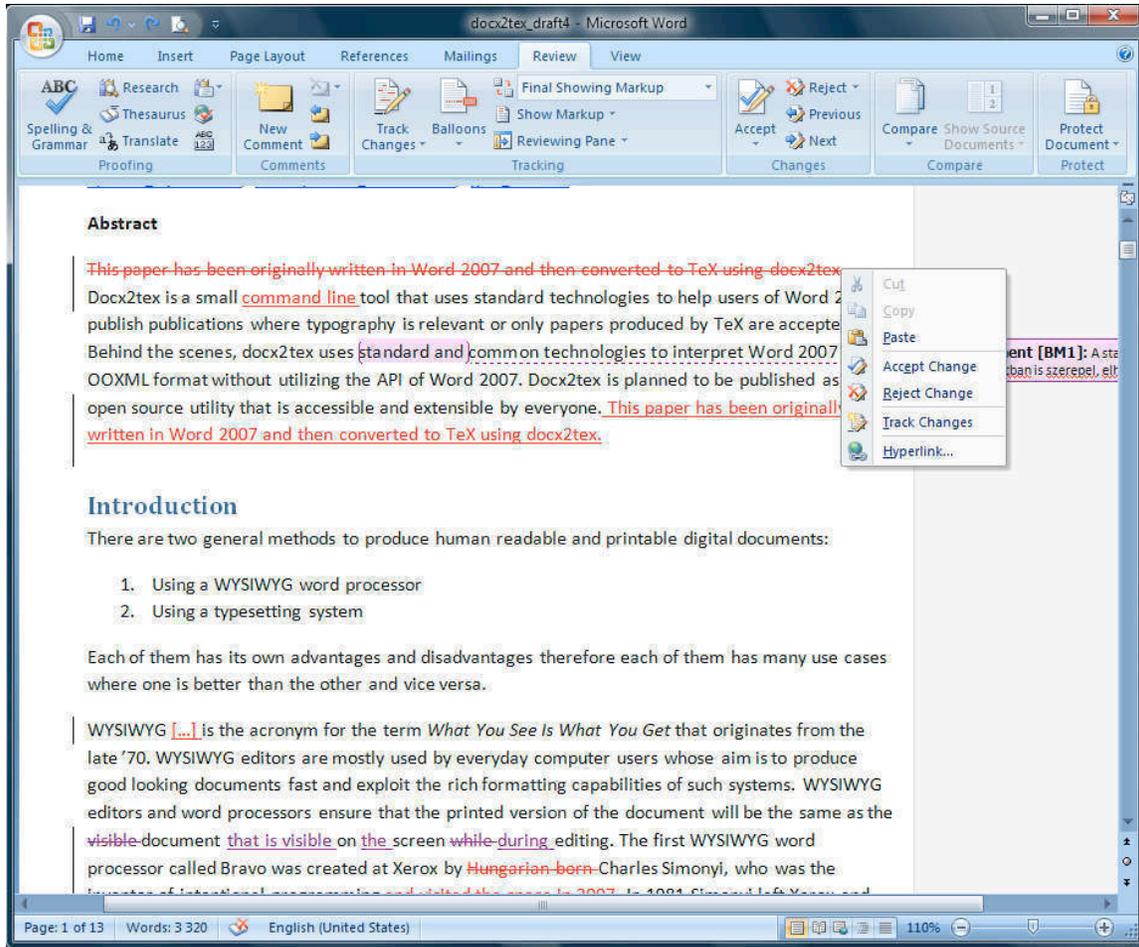


Figure 3: Track Changes function of Word

- | | |
|--|---|
| <p>[2] Wikipedia article on Microsoft Word. http://en.wikipedia.org/wiki/Microsoft_Office_Word</p> <p>[3] Product page of Microsoft Word. http://office.microsoft.com/en-us/word/FX100487981033.aspx</p> <p>[4] Wikipedia article on TeX. http://en.wikipedia.org/wiki/TeX</p> <p>[5] Product page of <i>Word2Tex</i>. http://www.chikrii.com/</p> <p>[6] Product page of <i>Word-to-LaTeX</i>. http://kebrt.webz.cz/programs/word-to-latex/</p> <p>[7] Product page of OpenOffice.org. http://www.openoffice.org/</p> <p>[8] Project page of <i>rtf2latex2e</i>. http://sourceforge.net/projects/rtf2latex2e/</p> | <p>[9] ECMA 376 OOXML Standard. http://www.ecma-international.org/publications/standards/Ecma-376.htm</p> <p>[10] ECMA 335 .NET CLI Standard. http://www.ecma-international.org/publications/standards/Ecma-335.htm</p> <p>[11] Link to ISO/IEC 23271:2006 Standard. http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html</p> <p>[12] Project page of ImageMagick. http://www.imagemagick.org/</p> <p>[13] Article on BSD license. http://en.wikipedia.org/wiki/BSD_license</p> |
|--|---|

Languages for bibliography styles

Jean-Michel Hufflen

LIFC (EA CNRS 4269)

University of Franche-Comté

16, route de Gray

25030 Besançon Cedex, France

`hufflen (at) lifc dot univ-fcomte dot fr`

`http://lifc.univ-fcomte.fr/~hufflen`

Abstract

BIB \TeX is the most commonly used bibliography processor in conjunction with L \TeX . To put bibliography styles into action, it uses a stack-based language written with postfix notation. Recently, other approaches have been proposed: some use a script programming language for designing bibliography styles, e.g., Perl in Bibulus; some are based on converters to XML texts and use XSLT for bibliography styles; a recent proposal — the `biblatex` package — consists of using L \TeX commands to control the layout of generated references, and more. We propose a comparative study of these approaches and show which programming styles are encouraged, from a point of view related to methodology. Finally, we explain how this study has influenced the design of MIBIB \TeX , our multilingual reimplement of BIB \TeX .

Keywords Bibliographies, bibliography styles, BIB \TeX , software quality, `bst`, BIB \TeX ++, `cl-bibtex`, MIBIB \TeX , packages `natbib`, `jurabib`, and `latexbib`, `Tib`, XML, XSLT, `nbst`, Perl, DSSSL.

1 Introduction

Three decades ago,¹ some programming languages were designed to be universal, that is, to serve all purposes. All of these languages — e.g., PL/1 [25], Ada [2] — have failed to be accepted as filling this role. Nowadays, only the C programming language [30] is still used for a very wide range of applications. Present-day programming languages are very diverse and put different paradigms into action: procedural programming, object-oriented programming, functional programming, process-oriented programming, logic programming, . . . In addition, most of these present languages are specialised, that is, designed for particular purposes, even if they are not formally limited to only one class of applications: two good examples are Java [28] and C# [39], originally designed for client-server applications. But, if you are building a standalone application using the object-oriented paradigm and are especially interested in the efficiency of the resulting program, it is well-known that a better choice is C++ [47], even if code generated by Java and C# compilers have greatly improved since their first versions.

The purpose of this article is neither a comparison of all programming languages — which would be

impossible — nor an absolute comparison of several programming languages — which would not be of interest — but a comparative study of languages used to develop *bibliography styles*. BIB \TeX [42] is the bibliography processor most commonly used in conjunction with the L \TeX typesetting engine [40], so most of the bibliography styles used for L \TeX texts are written using `bst`, the stack-based language of BIB \TeX [40, § 13.6]. But other proposals exist, based on other programming paradigms, and this article aims to study the advantages and drawbacks of these approaches. We will not discuss the typographical conventions ruling the typesetting of bibliographies — readers interested in this topic can consult manuals like [5, § 10], [6, §§ 15.54–15.76], [16, pp. 53–54] — but are interested only in the *development* of bibliography styles — from scratch or derived from other existing styles — and the expressive power of languages used to do that.

In Section 2, we recall the main quality factors of software, and show which factors are interesting from a point of view related to bibliography styles. Then BIB \TeX is studied in Section 3, including some modern use of this program. Other approaches are based on XML,² as shown in Section 4. This experience

¹ That is, at the time of \TeX 's first version. . . Let us recall that we are celebrating \TeX 's 30th birthday.

² EXtensible Markup Language. Readers interested in an introductory book to this formalism can refer to [44].

of dealing with several ways to develop bibliography styles has influenced the design of MIBIB \TeX — for ‘MultiLingual BIB \TeX ’, our multilingual reimplement of BIB \TeX [18]: we explain that in Section 5. Reading this article requires only a basic knowledge of BIB \TeX and a little experience about bibliography styles; we think that our examples should be understandable,³ even if readers do not know thoroughly the languages used throughout this article.

2 Criteria

Of course, this section does not aim to replace a textbook about software quality, we just make precise the terminology we use. Then we explain how these notions are applied to bibliography styles.

2.1 General point of view

The main reference for the terminology used in software quality is the beginning of [38], as recalled in most works within this topic. [38, Ch. 1] clearly distinguishes *external* quality factors, that may be detected by users of a product, and *internal* factors, that are only perceptible to computer professionals. Here are the main external quality factors:

correctness the ability of software products to exactly perform their tasks, as defined by the requirements and specification;

robustness the ability of software systems to work even in abnormal conditions;

extendability the ease with which products may be adapted to changes of specifications;

reusability the ability of products may be combined with others;

others being *efficiency*, *portability*, *verifiability*, *integrity*, *ease of use*, etc. Internal quality factors include *modularity*, *legibility*, *maintainability*, etc. The factors related to modularity are studied more precisely in [38, Ch. 2], they include:

modular decomposability the ability for a problem to be decomposed into subproblems;

modular composability the ability for modules to be combined freely with each other;

modular understandability each module can be separately understood by a human reader;

modular continuity a small change in a problem specification results in a change of just a module or few modules.⁴

³ Complete texts may be downloaded from MIBIB \TeX ’s home page: <http://lifc.univ-fcomte.fr/~hufflen/texts/mlbibtex/hc-styles/>.

⁴ This terminology is related to mathematical analysis: a function is continuous if a small change in the argument will yield a small change in the result.

2.2 Tasks of a bibliography processor

Given *citation keys* — stored in an `.aux` file when a source text is processed by L \TeX [40, Fig. 12.1] — a bibliography processor searches bibliography database files for resources associated with these keys, performs a sort operation on bibliographical items,⁵ and arranges them according to a bibliography style, the result being a source file for a ‘References’ section, suitable for a word processor. So does BIB \TeX .

Roughly speaking, a bibliography has to do two kinds of tasks:

- some are related to ‘pure’ programming, e.g., sorting bibliographical items, while
- others are related to put markup, in order for the word processor to be able to typeset the bibliography of a printed work.

The extendability of such a tool concerns these two kinds of tasks. On the one hand, we should be able to add a new relation order for sorting bibliographical items, since these lexicographical orders are language-dependent [24]. On the other hand, we should be able to build a new bibliography style, according to a publisher’s specification. This style may be developed from scratch if we do not find a suitable existing style. Or we can get it by introducing some changes to another style, i.e., *reusing* some parts of the previous style. In addition, finding the parts that have to be changed is related to the notion of modular understandability. Of course, building a new bibliography style is not an end-user’s task, but it should be possible by people other than the bibliography processor’s developers.

Another notion is related to extending a bibliography processor: improving it so that it is usable with more word processors. If we consider the formats built on T \TeX [34], L \TeX is still widely used, but more and more people are interested in alternatives, such as ConT \TeX t [13]. Likewise, some new typeset engines, such as X \TeX [32] or LuaT \TeX [14], should be taken into account. In addition, it should be possible to put the contents of a bibliography database file on a Web page, that is, to express the information about these items using the HTML language.⁶ A last example is given by RTF:⁷ at first glance, deriving bibliographies using the internal markup language of Microsoft Word may seem strange, but

⁵ ... unless the bibliography style is *unsorted*, that is, the order of items is the order of first citations. In practice, most bibliography styles are ‘sorted’.

⁶ HyperText Markup Language. [41] is a good introduction to this language.

⁷ Rich Text Format. A good introductory book to this markup language is [4].

```

@STRING{srd = {Stephen Reeder Donaldson}}
@BOOK{donaldson1993,
  AUTHOR = srd,
  TITLE = {The Gap into Power: A Dark and
    Hungry God Arises},
  PUBLISHER = {HarperCollins},
  SERIES = {The Gap},
  NUMBER = 3,
  YEAR = 1993}
@BOOK{donaldson1993a,
  EDITOR = srd,
  TITLE = {Strange Dreams},
  PUBLISHER = {Bantam-Spectra},
  YEAR = 1993}
@BOOK{murphy-mullaney2007,
  AUTHOR = {Warren Murphy and James
    Mullaney},
  TITLE = {Choke Hold},
  PUBLISHER = {Tor},
  ADDRESS = {New-York},
  SERIES = {The New Destroyer},
  NUMBER = 2,
  NOTE = {The original series has been
    created by Richard Sapir and
    Warren Murphy},
  YEAR = 2007,
  MONTH = nov}

```

Figure 1: Bibliographical entries in the .bib format.

such a strategy may cause Word end-users to discover progressively the tools related to \TeX .

3 \BIBTeX

3.1 Basic use

How to use \BIBTeX in conjunction with \LaTeX is explained in [40, § 12.1.3], and the .bib format, used within bibliography database files, is detailed in [40, § 13.2]; an example is given in Figure 1. As mentioned above, bibliography styles are written in a stack-based language using postfix notation. As an example, Figure 2 gives two functions used within the plain style of \BIBTeX .

\BIBTeX is indisputably correct⁸ and robust: as far as we have used it, the bibliographies it derives have satisfactory layout, at least for bibliographies of English-language documents. In addition, it has never crashed during our usage of it, even when dealing with syntactically incorrect .bib files.

⁸ When the word ‘correct’ is used in software engineering, it is related to the existence of a formal specification — i.e., a mathematical description — of the behaviour, and the program should have been proved correct w.r.t. this specification. Here we adopt a more basic and intuitive sense: the program’s results are what is expected by end-users.

```

FUNCTION {format.title}
{ title empty$
  { "" }
  { title "t" change.case$ }
  if$
}
FUNCTION {new.sentence.checkb}
{ empty$
  swap$ empty$
  and
  'skip$
  'new.sentence
  if$
}

```

Figure 2: Two functions from \BIBTeX ’s plain style.

Extending \BIBTeX , however, may be very tedious, especially for functionalities related to *programming*. For example, the only way to control the `SORT` command consists of using the entry variable `sort.key$` [40, Table 13.7]. Some workarounds may allow the definition of sort procedures according to lexicographic orders for natural languages other than English, but with great difficulty. Developing bibliography styles for word processors other than \LaTeX has been done, but only for formatters built on \TeX , e.g., `ConTeXt` [17]. In other cases, this task may be difficult since some features related to \TeX are hard-wired in some built-in functions of \BIBTeX , e.g., the use of ‘~’ for an unbreakable space character is in the specification of the `format.name$` function [23]. As an example, there is a converter from .bib format to HTML: `BIBTeX2HTML` [9]. It uses \BIBTeX , but most of this translator is not written using \BIBTeX ’s language, but in Objective CAML,⁹ a strongly typed functional programming language including object-oriented features [37]. Using such a tool — as well as the bibliography styles developed for `ConTeXt`’s texts [17] — is possible only if end-users do not put \LaTeX commands inside the values associated with \BIBTeX ’s fields.

We think that the continuity of the bibliography styles written using the `bst` language is average. Introducing some changes concerning the layout of fragments is easy, e.g., short-circuiting case changes for a title, as shown in [40, § 13.6.3], as well as changing the style of a string by using a command like ‘`\emph{...}`’¹. That is due to the fact that inserting additional strings before or after the contents of a field is easy if this information is at the stack’s top and has not been popped yet by means of the `writes$`

⁹ Categorical Abstract Machine Language.

```

\bibitem[{\Murphy\jbbtasep Mullaney\jbdy {2007}}]{%
  {}%
  {}{}{book}{2007}{}{}{}{}%
  {New-York\bpubaddr {} Tor\bibbdsep {} 2007}}%
  {{Choke Hold}}%
  {}{}{2}{}{}{}{}{}%
  ]{murphy-mullaney2007}
\jbbibargs {\bibnf {\Murphy} {\Warren} {W.} {} {} \Bibbtasep \bibnf {Mullaney}
  {James} {J.} {} {} } {\Warren MurphyJames Mullaney} {aus} {\bibtfont {Choke
  Hold}\bibatsep\ \apyformat {New-York\bpubaddr {} Tor\bibbdsep {} \novname\
  2007} \numberandseries {2}{The New Destroyer Series} \jbnote {1} {The
  original series has been created by Richard Sapir and Warren Murphy} }
  {\bibhowcited} \jbendnote {The original series has been created by Richard
  Sapir and Warren Murphy} \jbdointem {{\Murphy}{Warren}{W.}{}{}};
  {Mullaney}{James}{J.}{}{}} {} {} \bibAnnoteFile {murphy-mullaney2007}

```

Figure 3: $\text{BIB}\text{T}\text{E}\text{X}$'s output as used by the *jurabib* package.

function. For the same reason, adding a closing punctuation sign is easy; a shorthand example to do that is the `add.period$` function. Often handling a new field is easy, too [40, § 13.6.3]. On the other hand, changing the order of appearance of fields may be tedious.

In addition, it is well-known that there is no modularity within the *bst* language: each style is a monolithic file. If you develop a new style from an existing one, you just copy the *.bst* file onto a new file, and apply your changes. Of course, doing such a task requires good ‘modular understanding’ of the functions belonging to the ‘old’ style. Sometimes, that is easy — cf. the `format.title` function given in Figure 2 — while other times, understanding the role of a function is possible only if you know the stack’s state — cf. the `new.sentence.checkb` function in the same figure.¹⁰

3.2 Task delegation

Originally, all the predefined bibliography styles provided by $\text{BIB}\text{T}\text{E}\text{X}$'s generated ‘pure’ $\text{L}\text{A}\text{T}\text{E}\text{X}$ texts, in the sense that only basic $\text{L}\text{A}\text{T}\text{E}\text{X}$ commands were used: the `\bibitem` command, the `thebibliography` environment [40, § 12.1.2], and some additional commands for word capitalisation or emphasis. No additional package was required when derived bibliographies were processed by $\text{L}\text{A}\text{T}\text{E}\text{X}$.

This situation has changed when the author-date system was implemented by the *natbib* package and the bibliography styles associated with it [40, § 12.3.2]. Progressively, other bibliography styles have been released, working as follows: $\text{BIB}\text{T}\text{E}\text{X}$'s output is marked up with $\text{L}\text{A}\text{T}\text{E}\text{X}$ commands defined

¹⁰ This function is used when the decision of beginning a new sentence within a reference depends on the presence of two fields within an entry.

in an additional package. Citation and formatting functions can be customised by redefining these commands. In other words, we can say that $\text{BIB}\text{T}\text{E}\text{X}$ *delegates* the layout of bibliographies to these commands.

3.2.1 Interface packages

Figure 3 gives an example of using the *jurabib* bibliography style. The $\text{L}\text{A}\text{T}\text{E}\text{X}$ commands provided by the *jurabib* package can be redefined like any $\text{L}\text{A}\text{T}\text{E}\text{X}$ command, although the best method is to use the `\jurabibsetup` command, as shown in [40, § 12.5.1]. A similar approach is used within the *amsxport* bibliography style [8] and the bibliography styles usable with $\text{Con}\text{T}\text{E}\text{X}\text{T}$ [17].

This *modus operandi* is taken to extremes by the *biblatex* package [36]. In such cases, $\text{BIB}\text{T}\text{E}\text{X}$ is used only to search bibliography database files and sort references. The advantage: end-users can customise the layout of bibliographies without any knowledge of the *bst* language. But $\text{BIB}\text{T}\text{E}\text{X}$ still remains used to sort references, and this task is not easily customisable, as mentioned above.

3.2.2 *Tib*

In fact, this notion of task delegation already existed in *Tib* [1], a bibliography processor initially designed for use with Plain TEX , although it can also be used with $\text{L}\text{A}\text{T}\text{E}\text{X}$. An example of a bibliography style file used by *Tib* is given in Figure 4: it consists of some *Tib* commands — e.g., ‘f’ for ‘citations as footnotes’ — followed by some definitions of TEX commands for typesetting citation references and bibliographies’ items. That is, *Tib* delegates a bibliography’s layout to these commands. Let us recall that the bibliography database files searched by this processor do not

```

#
# standard footnote format (latex)
#
# if titles are desired in loc. cit. references, see note in stdftl.ttx
#
#         include word-definition file (journals and publishers)
I TMACLIB amsabb.ttz
f         footnotes
L         use ibid and loc cit
CO        empty citation string
O         for multiple citations use ordering of reference file

%The lines below are copied verbatim into the output document as TeX commands.
%First the file Macros.ttx is \input with Macros and default settings.
%The control string \TMACLIB is just a path.
%The \footnote macro is from LaTeX
%
\input \TMACLIB stdftl.ttx %macros for formatting reference list
\Refstda\Citesuper %set general formats for reference list and citations
\def\LCitemark{\footnotemark}\def\RCitemark{ }
\def\Citecomma{$^,$\footnotemark}
\def\LAitemark{\addtocounter{footnote}{1}\arabic{footnote}}
\def\RAitemark{ }
\def\LCitemark#1\RCitemark{\def\Ztest{ } \def\Zstr{#1}}

```

Figure 4: The footl.tib file.

```

%A |srd|
%T The Gap into Power: A Dark and Hungry God
  Arises
%P HarperCollins
%S The Gap
%N 3
%D 1993

%A |srd|
%T Strange Dreams
%I Bantam-Spectra
%D 1993

%A Warren Murphy
%A James Mullaney
...
%O November 2007. The original series...
The 'srd' abbreviation should be defined by means of
the following Tib command:
      D srd Stephen Reeder Donaldson

```

Figure 5: Entries using the Refer format.

use the .bib format, but rather the Refer format,¹¹ an example being given in Figure 5.

3.3 Extending bst

The following works allow bibliography style writ-

¹¹ The pybibliographer program can be used as a converter from the .bib format to the Refer format: see [40, § 13.4.5] for more details.

ers to compile bst styles, and annotate or extend the result. As far as we know, they are not widely used. If we consider a style already written in bst and to be adapted, this approach allows more ambitious changes. However, they do not propose a new methodology for designing such styles, so taking maximum advantage of the target languages is difficult for style designers.

3.3.1 BIBTEX++

BIBTEX++ [31] allows a bst style file to be compiled into Java classes [28]. As an example, the `new.sentence.checkb` function (cf. Fig. 2) is compiled into the Java function `new_sentence_checkb` given in Figure 6. BIBTEX++ can also run native bibliography styles developed in Java, from scratch or derived from the compilation of ‘old’ styles. Other functionalities, such as the production of references for programs other than L^AT_EX, can be implemented by means of *plug-ins*. There are six steps in BIBTEX++’s process: for example, parsing a .bst file is the fourth one. After each step, there is a *hook*, as a callback that allows this process to be customised.

3.3.2 cl-bibtex

cl-bibtex [35] is based on ANSI¹² Common Lisp [11]. It includes an interpreter for the bst language, and

¹² American National Standards Institute.

```
public void new_sentence_checkb(String s0,
                               String s1) {
    int i0, i1 ; i1 = BuiltIn.empty(s1) ;
    i0 = i1 ; i1 = BuiltIn.empty(s0) ;
    i0 = and(new Cell(i0),i1) ;
    if (i0 <= 0) new_sentence() ;
}
```

Figure 6: A bst function compiled into Java.

```
(define-bst-primitive "if$"
  ((pred (boolean)) (then (symbol body))
   (else (symbol body)))
  ()
  :interpreted
  (bst-execute-stack-literal
   (if pred then else)))
```

Figure 7: Implementation of `if$` in `cl-bibtex`.

can also compile a `BIBTEX` style file into a Common Lisp program, as a starting point for customising such a style, by refining the corresponding Common Lisp program. As a short example, we show in Figure 7 how the `if$` function of `BIBTEX` is implemented.

4 Using XML-like formats

Over the past several years, XML has become a central formalism for data interchange, so some projects are based on an XML-like language representing bibliographical items.

4.1 Converters

Several converters from the `.bib` format into an XML-like format have been developed: the `bib2xml` program [43], and the converter used as part of the `BIBTEXXML` project [12]. `MIBIBTEX` uses such a converter, too, and the result of the conversion of the second bibliographical entry of Figure 1 is given in Figure 8; the conventions used throughout such XML texts are a revision of the specification given in [10, § B.4.4].

The main difficulty of these tools is related to the `LATEX` commands put inside the values associated with `BIBTEX` fields. The `bib2xml` converter expands the commands for accents and diacritical signs into the corresponding single letters belonging to the Unicode encoding [48], but just drops out the ‘\’ characters that open the other commands. `MIBIBTEX`’s converter processes more commands — e.g., `\emph`, `\textbf` — but of course, the way of dealing with user-defined commands should be defined by end-users [21].

```
<book id="donaldson1993a">
  <editor>
    <name>
      <personname>
        <first>Stephen Reeder</first>
        <last>Donaldson</last>
      </personname>
    </name>
  </editor>
  <title>Strange Dreams</title>
  <publisher>Bantam Spectra</publisher>
  <year>1993</year>
</book>
```

Figure 8: XML-like format used in `MIBIBTEX`.

4.2 XSLT

XSLT¹³ is the language used for the transformation of XML texts. Building a ‘References’ section is a particular case of transformation. This point is true for `LATEX` source files as well as verbatim texts or HTML pages. Figure 9 shows how multiple authors or editors connected by an empty `and` tag can be processed, the result being a source text for `LATEX`. More ambitious examples of using XSLT for typesetting texts are given in [46].

We have personally written many XSLT programs serving very diverse purposes. This language allows good modularity and reusability of fragments of existing programs. It allows users to write robust programs, too. As for developing bibliography styles, it offers good continuity, except for multilingual extensions. It was difficult to add information for a natural language without directly modifying an existing style. More precisely, that was difficult with the first version (1.0) [49], but has been improved in XSLT 2.0 where using *modes* has been refined [50, § 6.5]. Likewise, the expressive power of the `xs1:sort` element has been improved in this new version [50, § 13].

Extending XSLT functionalities often consists of calling external functions written using a more ‘classical’ programming language such as C or Java. That is possible, but not in a portable way, because it depends on the programming languages accepted by each XSLT processor. In practice, this point mainly concerns new lexicographical order relations within bibliography styles.

4.3 nbst

`nbst`¹⁴ is the language used within `MIBIBTEX` for specifying bibliography styles. As explained in [18],

¹³ eXtensible Stylesheet Language Transformations.

¹⁴ New Bibliography STyles.

```

<xsl:template match="author">
  <xsl:apply-templates/><xsl:text>. </xsl:text>
</xsl:template>
<xsl:template match="editor">
  <xsl:apply-templates/>
  <xsl:text>, </xsl:text>
  <xsl:choose>
    <xsl:when test="count(*) > 1">
      <xsl:text>\bbled</xsl:text>
    </xsl:when>
    <xsl:otherwise>\bbleds</xsl:otherwise>
  </xsl:choose>
  <xsl:text>. </xsl:text>
</xsl:template>
<xsl:template match="name | personname">
  <xsl:apply-templates/>
</xsl:template>
<xsl:template match="first">
  <xsl:value-of select="concat(.,' ')" />
</xsl:template>
<xsl:template match="last">
  <xsl:value-of select="." />
</xsl:template>
<xsl:template match="and">
  <xsl:choose>
    <xsl:when
      test="following-sibling::and or
            following-sibling::and-others">
      <xsl:text>, </xsl:text>
    </xsl:when>
    <xsl:otherwise>
      <xsl:text> \bbland\ </xsl:text>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

```

Figure 9: Dealing with authors or editors in XSLT.

this language is close to XSLT, and introduces a kind of inheritance for natural languages' specification. First, we look for a template whose `language` attribute matches the current language, and second a more general template, without the `language` attribute.

MIBIBTEX is written in Scheme [29], and XML texts are represented using the SXML¹⁵ format [33]. Roughly speaking, this format uses prefixed notation, surrounded by parentheses — as in any Lisp dialect — for tags surrounding contents. As an example, the result of parsing the bibliographical entries of Figure 1 is sketched in Figure 10. Dealing directly with Scheme functions is needed when new language-dependent lexicographical order relations are to be

¹⁵ Scheme implementation of XML.

```

(*TOP*
(*PI* xml "version=\"1.0\"
         encoding=\"ISO-8859-1\"")
(mlbiblio
...
(book
 (@ (id "murphy-mullaney2007"))
  (author
   (name (personname (first "Warren")
                     (last "Murphy")))
   (and
    (name (personname (first "James")
                     (last "Mullaney"))))
   (title "Choke Hold") (publisher "Tor")
   (year "2007") (month (nov)) (number "2")
   (series "The New Destroyer")
   (address "New-York")
   (note "The original series..."))))

```

Figure 10: Using the SXML format.

added [24]. `nbst` texts can call functions directly written in Scheme, as well.

4.4 Perl

Perl¹⁶ [51] can be used for bibliography styles, as is done by Bibulus [52], this program being based on the `bib2xml` converter [43]. The resulting bibliography styles are compact, modular, and easily extensible. The modularity of Bibulus styles can be illustrated by the `\bibulus` command that can be used in place of the `\bibliographystyle` command:

```

\bibulus{citationstyle=numerical,
         surname=comes-first,
         givennames=initials,
         blockpunctuation=.}

```

Multilingual features are processed by means of substitutions, which can easily be incorrect: for example, a month name precedes the year in English, but follows the year in Hungarian. So a rough substitution of an English month name is insufficient.¹⁷ Last but not least, Bibulus is not very easy to use, it is presently accessible only to developers.

4.5 DSSSL

DSSSL¹⁸ [27] was initially designed as the stylesheet language for SGML¹⁹ texts. Since XML is a subset of SGML, stylesheets written using DSSSL can be applied to XML texts. DSSSL is rarely used now,

¹⁶ Practical Extraction Report Language.

¹⁷ The same criticism holds for the `babelbib` package [15].

¹⁸ Document Style Semantics Specification Language.

¹⁹ Standard Generalised Markup Language. Now it is only of historical interest. Readers interested in this metalanguage can refer to [3].

```

<!DOCTYPE style-sheet PUBLIC "-//James Clark//DTD DSSSL Style Sheet//EN">
<style-sheet>
  <style-specification id="hcs">
    <style-specification-body>
      (root (let ((margin-size 1in)) ; DSSSL uses quantities, analogous to TEX's dimensions.
        (make simple-page-sequence
          ; ; An identifier ending with the ':' characters is a key, bound to the following value.
          page-width: 210mm page-height: 297mm left-margin: margin-size
          right-margin: margin-size top-margin: margin-size bottom-margin: margin-size
          header-margin: margin-size footer-margin: 12mm center-footer: (page-number-sosofo)
          (process-children))))
      (element book
        (make-reference (lambda (current-children) ; Function to be applied as soon as the general
          (make sequence ; framework for a reference has just been built: cf. the
            (author-xor-editor current-children) ; definition of our make-reference
            (process-matching-children "title") ; function below.
            (process-seriesinfo current-children)
            (apply sosofo-append
              (map process-matching-children
                '("publisher" "address" "month" "year" "note"))))))))
      (element author (process-author-or-editor)) ; The same for editor elements.
      (element name (process-children-trim)) ; The same for number elements.
      (element personname (processing-matching-children "first" "von" "last" "junior"))
      (element first (ending-with space-literal))
      (element last (process-children-trim))
      (element and (if (node-list-empty? (select-elements (follow (current-node)) "and"))
        (literal " and " )
        comma-space-literal))
      (element (book title) (make sequence font-posture: 'italic (process-and-closing-period)))
      (element year (process-and-closing-period)) ; The same for series and note elements.
      (element month (make sequence (process-children) space-literal))
      (element jan (literal "January")) ... ; Other month elements skipped.
      (element publisher (ending-with comma-space-literal)) ; The same for address elements.
      ...
      ;; Definitions for particular literals and strings:
      (define comma-space-literal (literal ", "))
      (define period-string ".")
      (define space-literal (literal " "))
      ;; General framework for references' layout:
      (define make-reference
        (let ((biblioentry-indent 20pt))
          (lambda (process-f)
            (make paragraph
              first-line-start-indent: (- biblioentry-indent) font-size: 12pt quadding: 'justify
              space-before: 10pt start-indent: biblioentry-indent
              (literal "[" (attribute-string "id") "]" ") (process-f (children (current-node))))))
      ;; Some utility functions:
      (define (process-author-or-editor)
        (process-matching-children "name" "and"))
      (define (ending-with literal-0)
        (make sequence (process-children-trim) literal-0))
      ...
    </style-specification-body>
  </style-specification>
</style-sheet>

```

Figure 11: Example of a DSSSL stylesheet.

```

(define (process-and-closing-period)
  (let ((the-string (string-trim-right (data (current-node))))) ; Get the contents and leave trailing
    ; space characters.
    (literal (if (check-for-closing-sign? the-string) ; Checking if the-string ends with '.', '?', or '!'.
      the-string
      (string-append the-string period-string))))))

(define (author-xor-editor node-list)
  (let ((author-node-list (select-elements node-list "author"))
        (editor-node-list (select-elements node-list "editor")))
    (make sequence
      (cond ((node-list-empty? author-node-list)
             (if (node-list-empty? editor-node-list)
                 (error "Neither author, nor editor!")
                 (make sequence (process-node-list editor-node-list) (literal ", editor."))))
            ((node-list-empty? editor-node-list)
             (make sequence
              (process-node-list author-node-list)
              (if (check-for-closing-sign? (string-trim-right (data author-node-list)))
                  (empty-sosof)
                  (literal period-string))))
            (else (error "Both author and editor!"))))
    (literal " "))))

```

Figure 12: Some auxiliary functions implemented in DSSSL.

```

(define (b-if$ sxml-mlbiblio-tree current-entry-plus)
  ;; sxml-biblio-tree is the complete tree of all the entries to be processed, current-entry-plus the annotated
  ;; tree of the current entry.
  (let* ((i2 ((b-bst-stack-pv 'pop))) ; "Else" part.
         (i1 ((b-bst-stack-pv 'pop))) ; "Then" part.
         (i0 ((b-bst-stack-pv 'pop))) ; Condition.
        (if (integer? i0)
            (b-process-sequence (if (positive? i0) i1 i2) sxml-mlbiblio-tree current-entry-plus)
            (begin
              (msg-manager 'bst-type-conflict) 'if$ i0
              #t))))

```

Figure 13: Implementing `if$` within MIBIB \TeX 's compatibility mode.

but the example we show illustrates how a functional programming language can implement a bibliography style. More examples can be found in [10, § 7.5].

Figure 11 gives some excerpts of a stylesheet that displays the items of a bibliography by labelling them with their own keys. The core expression language of DSSSL is a side-effect free subset of Scheme. As shown in Figure 11, processing elements uses pattern-matching:

```

(element name E)
(element (name0 name) E0)

```

the *E* expression specifies how to process the *name* element, unless this element is a child of the *name*₀ element, in which case the *E*₀ expression applies. The choice of the accurate expression is launched by functions such as `process-matching-children`, `process-children`, and `process-node-list`.

Expressions like *E* or *E*₀ consist of assembling *literals* by means of the `make` form, using types predefined in DSSSL: `paragraph`, `sequence`, ... The generic type of such results is called *sosof*²⁰ w.r.t. DSSSL's terminology.

Figure 12 illustrates this style of programming by showing some specific details: how to implement BIB \TeX 's `add.period$` function, and the switch between `author` and `editor` elements for a book. This stylesheet can be run by the `jade`²¹ program; as shown in [10, § 7.5.2], the \TeX -like typeset engine able to process such results is Jade \TeX .

Fragments of DSSSL stylesheets can be organised into libraries, so this language is modular. Most of the implementations of it are robust, efficient, but

²⁰ Specification Of a Sequence Of Flow Objects.

²¹ James Clark's Awesome DSSSL Engine.

they are neither extensible, nor easy to use, because we have to make precise a predefined *backend*. For example, if jade is used to process the complete stylesheet given in Figures 11 & 12, the possible backends are `tex` (resp. `rtf`), in which case the result is to be processed by Jade \TeX (resp. Microsoft Word or OpenOffice). Deriving texts directly processable by \LaTeX or Con \TeX t is impossible.

5 The application to MIBIB \TeX

When we designed MIBIB \TeX 's present version, we had had much experience in programming DSSSL and XSLT stylesheets. We thought that a language close to DSSSL would provide more expressive power for developing, but would be accessible only for programmers. A language close to XSLT is better from this point of view, provided that an extension mechanism is given for operations related to pure programming, e.g., the definition of new relation orders [24]. In addition, performing some operations may be more difficult than in `bst`, e.g., the `add.period$` functionality.

The only solution is to provide an initial library *legible* from a point of view related to methodology [19]. A compatibility mode is needed in order to ease the transition between old and new bibliography styles [20] — Figure 13 shows how the `if$` function is implemented within this mode, in comparison with the implementation of `cl-bibtex`, given in Figure 7. This progressively led us to the architecture described in [22].

We can be objective about MIBIB \TeX only with difficulty. However, several points seem to us to confirm our choices. First, XSLT has succeeded as a language able to deal with XML texts, much more so than DSSSL with SGML texts. Second, the need for a classical programming language: using the whole expressive power of Scheme — and not a subset as in DSSSL — allowed us to program efficiently, by using advanced features of Scheme. Third, our experience with Con \TeX t [21] seems to confirm the extendability of our tool.

6 Conclusion

Table 1 summarises our experience with the languages we have described above. Of course, this synthesis is not as objective as benchmarks would have been. It is just a study of the effort we have made for developing bibliography styles, and a professional view of the results we have found.

To end, let us make a last remark about what is done in MIBIB \TeX : the separation of functionalities related to programming, written in Scheme, and specifications of layout, given in an XSLT-like

	BIB \TeX	XSLT	DSSSL	Bibulus
Correctness	✓			
Robustness	✓	✓	✓	
Extendability	✗	✓	✗	✓/✗
Reusability	✓/✗			
Modularity	poor	✓	✓	✓
Continuity	average	✓ ^a	✓	✓
Efficiency	✓		✓	
Ease of use	✓	average	✗	✗

^a ... except for multilingual features, in XSLT 1.0.

Table 1: Languages for bibliography styles: synthesis.

language. Analogous combinations exist, the most widely used are a logic programming language, like Prolog,²² called within a C (or similar) program. This *modus operandi* allows programmers to use a very specialised language only when it is suitable. There is an analogous example within \TeX 's world: Lua \TeX . Functionalities related to typesetting are performed by commands built into \TeX , whereas other functions are implemented by means of the Lua language [26]. So \TeX is used as the wonderful typesetting engine that it is, and functionalities difficult to implement with \TeX 's language²³ are delegated to a more traditional programming language.

BIB \TeX is still a powerful bibliography processor, but the main way to extend it easily concerns the layout of the bibliographies. That was sufficient some years ago, but not now with the use of Unicode, new processors like X \TeX , and new languages like HTML.

7 Acknowledgements

I have been able to write this article because I have had much occasion to become familiar with the languages and applications mentioned above. For example, some years ago, a colleague asked me to help him with a DSSSL program because I knew Scheme: that was my first contact with this language and SGML, before XML came out. Another time, a friend who used Plain \TeX asked me a question about `Tib`, although I had merely heard of the name of this program, and I discovered it in this way. So I was thinking about all these people when I was writing this article, and I am grateful to them. Many thanks to Karl Berry: as usual, he is a conscientious proofreader and ‘figure-positioner’.

²² A good introductory book to this language is [7].

²³ Some examples can be found in [45].

References

- [1] James C. ALEXANDER: *T1b: a T_EX Bibliographic Preprocessor*. Version 2.2, see `CTAN:biblio/tib/tibdoc.tex`. 1989.
- [2] ANSI: *The Programming Language Ada[®] Reference Manual*. Technical Report ANSI/MIL-STD-1815A-1983, American National Standard Institute, Inc. LNCS No. 155, Springer-Verlag. 1983.
- [3] Neil BRADLEY: *The Concise SGML Companion*. Addison-Wesley. 1997.
- [4] Sean M. BURKE: *RTF Pocket Guide*. O'Reilly. July 2003.
- [5] Judith BUTCHER: *Copy-Editing. The Cambridge Handbook for Editors, Authors, Publishers*. 3rd edition. Cambridge University Press. 1992.
- [6] *The Chicago Manual of Style*. The University of Chicago Press. The 14th edition of a manual of style revised and expanded. 1993.
- [7] William F. CLOCK SIN and Christopher S. MELLISH: *Programming in Prolog*. 5th edition. Springer-Verlag. 2003.
- [8] Michael DOWNES: “The `amsrefs` L^AT_EX Package and the `amsxport` BIB_TE_X Style”. *TUGboat*, Vol. 21, no. 3, pp. 201–209. September 2000.
- [9] Jean-Christophe FILLIÂTRE and Claude MARCHÉ: *The BIB_TE_X2HTML Home Page*. June 2006. <http://www.lri.fr/~filliatr/bibtex2html/>.
- [10] Michel GOOSSENS and Sebastian RAHTZ, with Eitan M. GURARI, Ross MOORE, and Robert S. SUTOR: *The L^AT_EX Web Companion*. Addison-Wesley Longmann, Inc., Reading, Massachusetts. May 1999.
- [11] Paul GRAHAM: *ANSI Common Lisp*. Series in Artificial Intelligence. Prentice Hall, Englewood Cliffs, New Jersey. 1996.
- [12] Vidar Bronken GUNDERSEN and Zeger W. HENDRIKSE: *BIB_TE_X as XML Markup*. January 2007. <http://bibtexml.sourceforge.net>.
- [13] Hans HAGEN: *ConT_EXt, the Manual*. November 2001. <http://www.pragma-ade.com/general/manuals/cont-enp.pdf>.
- [14] Hans HAGEN: “Lua_TE_X: Howling to the Moon”. *Biuletyn Polskiej Grupy Użytkowników Systemu T_EX*, Vol. 23, pp. 63–68. April 2006.
- [15] Harald HARDERS: „Mehrsprachige Literaturverzeichnisse: Anwendung und Erweiterung des Pakets `babelbib`“. *Die T_EXnische Komödie*, Bd. 4/2003, S. 39–63. November 2003.
- [16] *Hart's Rules for Composers and Readers at the University Press*. Oxford University Press. 39th edition. 1999.
- [17] Taco HOEKWATER: “The Bibliographic Module for ConT_EXt”. In: *EuroT_EX 2001*, pp. 61–73. Kerkrade (the Netherlands). September 2001.
- [18] Jean-Michel HUFFLEN: “MIBIB_TE_X's Version 1.3”. *TUGboat*, Vol. 24, no. 2, pp. 249–262. July 2003.
- [19] Jean-Michel HUFFLEN: “Bibliography Styles Easier with MIBIB_TE_X”. In: *Proc. EuroT_EX 2005*, pp. 179–192. Pont-à Mousson, France. March 2005.
- [20] Jean-Michel HUFFLEN: “BIB_TE_X, MIBIB_TE_X and Bibliography Styles”. *Biuletyn GUST*, Vol. 23, pp. 76–80. In *BachoT_EX 2006 conference*. April 2006.
- [21] Jean-Michel HUFFLEN: “MIBIB_TE_X Meets ConT_EXt”. *TUGboat*, Vol. 27, no. 1, pp. 76–82. EuroT_EX 2006 proceedings, Debrecen, Hungary. July 2006.
- [22] Jean-Michel HUFFLEN: “MIBIB_TE_X Architecture”. *ArsT_EXnica*, Vol. 2, pp. 54–59. In GUIT 2006 meeting. October 2006.
- [23] Jean-Michel HUFFLEN: “Names in BIB_TE_X and MIBIB_TE_X”. *TUGboat*, Vol. 27, no. 2, pp. 243–253. TUG 2006 proceedings, Marrakesh, Morocco. November 2006.
- [24] Jean-Michel HUFFLEN: “Managing Order Relations in MIBIB_TE_X”. *TUGboat*, Vol. 29, no. 1, pp. 101–108. EuroBachoT_EX 2007 proceedings. 2007.
- [25] IBM SYSTEM 360: *PL/1 Reference Manual*. March 1968.
- [26] Roberto IERUSALIMSKY: *Programming in Lua*. 2nd edition. Lua.org. March 2006.
- [27] International Standard ISO/IEC 10179:1996(E): *DSSSL*. 1996.
- [28] *Java Technology*. March 2008. <http://java.sun.com>.
- [29] Richard KELSEY, William D. CLINGER, Jonathan A. REES, Harold ABELSON, Norman I. ADAMS IV, David H. BARTLEY, Gary BROOKS, R. Kent DYBVIG, Daniel P. FRIEDMAN, Robert HALSTEAD, Chris HANSON, Christopher T. HAYNES, Eugene Edmund KOHLBECKER, JR, Donald OXLEY, Kent M. PITMAN, Guillermo J. ROZAS, Guy Lewis STEELE, JR, Gerald Jay

- SUSSMAN and Mitchell WAND: “Revised⁵ Report on the Algorithmic Language Scheme”. *HOSC*, Vol. 11, no. 1, pp. 7–105. August 1998.
- [30] Brian W. KERNIGHAN and Dennis M. RITCHIE: *The C Programming Language*. 2nd edition. Prentice Hall. 1988.
- [31] Ronan KERYELL: “BIB \TeX ++: Towards Higher-Order BIB \TeX ing”. In: *Euro \TeX 2003*, p. 143. ENSTB. June 2003.
- [32] Jonathan KEW: “X \TeX in \TeX Live and beyond”. *TUGboat*, Vol. 29, no. 1, pp. 146–150. EuroBach \TeX 2007 proceedings. 2007.
- [33] Oleg E. KISELYOV: *XML and Scheme*. September 2005. <http://okmij.org/ftp/Scheme/xml.html>.
- [34] Donald Ervin KNUTH: *Computers & Typesetting. Vol. A: The \TeX book*. Addison-Wesley Publishing Company, Reading, Massachusetts. 1984.
- [35] Matthias KÖPPE: *A BIB \TeX System in Common Lisp*. January 2003. <http://www.nongnu.org/cl-bibtex>.
- [36] Philipp LEHMAN: *The biblatex Package. Programmable Bibliographies and Citations*. Version 0.7 (beta). December 2007. <http://www.ctan.org/tex-archive/macros/latex/expt1/biblatex/doc/biblatex.pdf>.
- [37] Xavier LEROY, Damien DOLIGEZ, Jacques GARRIGUE, Didier RÉMY and Jérôme VOILLON: *The Objective Caml System. Release 0.9. Documentation and User’s Manual*. 2004. <http://caml.inria.fr/pub/docs/manual-ocaml/index.html>.
- [38] Bertrand MEYER: *Object-Oriented Software Construction*. Series in Computer Science. Prentice Hall International. 1988.
- [39] MICROSOFT CORPORATION: *Microsoft C# Specifications*. Microsoft Press. 2001.
- [40] Frank MITTELBACH and Michel GOOSSENS, with Johannes BRAAMS, David CARLISLE, Chris A. ROWLEY, Christine DETIG and Joachim SCHROD: *The L \TeX Companion*. 2nd edition. Addison-Wesley Publishing Company, Reading, Massachusetts. August 2004.
- [41] Chuck MUSCIANO and Bill KENNEDY: *HTML & XHTML: The Definitive Guide*. 5th edition. O’Reilly & Associates, Inc. August 2002.
- [42] Oren PATASHNIK: *BIB \TeX ing*. February 1988. Part of the BIB \TeX distribution.
- [43] Chris PUTNAM: *Bibliography Conversion Utilities*. February 2005. <http://www.scripps.edu/~cdputnam/software/bibutils/bibutils.html>.
- [44] Erik T. RAY: *Learning XML*. O’Reilly & Associates, Inc. January 2001.
- [45] Denis B. ROEGEL : « Anatomie d’une macro ». *Cahiers GUTenberg*, Vol. 31, p. 19–27. Décembre 1998.
- [46] Bob STAYTON: *DocBook—XSL. The Complete Guide*. 3rd edition. Sagehill Enterprises. February 2005.
- [47] Bjarne STROUSTRUP: *The C++ Programming Language*. 2nd edition. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts. 1991.
- [48] THE UNICODE CONSORTIUM: *The Unicode Standard Version 5.0*. Addison-Wesley. November 2006.
- [49] W3C: *XSL Transformations (XSLT). Version 1.0*. W3C Recommendation. Edited by James Clark. November 1999. <http://www.w3.org/TR/1999/REC-xslt-19991116>.
- [50] W3C: *XSL Transformations (XSLT). Version 2.0*. W3C Recommendation. Edited by Michael H. Kay. January 2007. <http://www.w3.org/TR/2007/WD-xslt20-20070123>.
- [51] Larry WALL, Tom CHRISTIANSEN and Jon ORWANT: *Programming Perl*. 3rd edition. O’Reilly & Associates, Inc. July 2000.
- [52] Thomas WIDMAN: “Bibulus—a Perl XML Replacement for BIB \TeX ”. In: *Euro \TeX 2003*, pp. 137–141. ENSTB. June 2003.

Vistas for T_EX: liberate the typography! (Part I)

Chris Rowley

Semantic Maths for All! Project

Department of Mathematics and Statistics

Faculty of Mathematics, Computing and Technology

Open University, London, UK

c.a.rowley (at) open dot ac dot uk

Abstract

This is a polemic in favour of liberating the core typesetting structures and algorithms around which T_EX is built from the monolithic superstructure of the mini-dinosaur of a program called `tex` and its more or less modernised and approachable derivatives such as `xetex` and `luatex`.

Although the high-level aims of the programme of activity advocated here have a lot in common with those of the very exciting and active LuaT_EX project, the route I propose seems to me to be very different. The major ambition of the latter project is to embed (something similar to) the whole of the current T_EX system within a vastly more complex monolith [*sic*] of an application which will presumably be well-adapted to the formatting needs of oriental languages. To this monolith are now being added many well-oriented intrusions [*sic*] into but a single instance of that ancient bedrock of T_EX!

Of course, `luatex` promises to provide a spectacularly sophisticated and highly hackable system that will eventually enable a great evolutionary radiation of species within the phylum of automated document processing; hence the importance and fascination, for me at least, of the developmental path of the LuaT_EX project.

Pursuing the paleontological metaphor well beyond its point of total and painful collapse, my plan can be thought of as providing many tools that can be easily dispersed in such a way that T_EX's clever genes can influence (for the good) far more aspects of the evolution of automated typesetting: all this abundance being more speedily and robustly achieved due to not being held back by the decision to build all future systems on a perfectly preserved and complete digestive system from a fossilised ancestral T_EXosaur.

I am here also making a plea to the Grand Technical Wizards of TUG to widen support from their development fund's treasure chest to encompass projects that are designed to spread T_EX's influence and presence throughout the fertile modern world of document processing via its algorithms alone, without the dead weight of its monolithic, programmatic paradigm and the many somewhat dated aspects of its detailed software design.

Adding topicality and an even longer time-base to the metaphors, please can we have plentiful levels of international funding to support an actual Big Bang to get the elementary particles of T_EX spread throughout the typesetting universe, rather than funding only an engineering wonder (for interesting but small-scale experiments to find new T_EX-like particles) such as the LHC: *LuaT_EX's Hard-problems Cruncher*, a 'shining star in the East' which I fear may spin-off some big black-holes to trap even the most energetic of us mortal, western programmers.

1 Introduction

This could easily have been the shortest genuine paper in this, or any, T_EX Users Group proceedings. All it needs to say is: please support Free, Open and Reusable Algorithms from T_EX (yes folks, the FORAT Campaign starts here!).

However, I will attempt, in this and subsequent papers, to expand on some examples of what I mean by liberating T_EX's formatting algorithms in the form, for example, of C++ libraries, embeddable JavaScript or similar reusable artefacts.

In Section 3 I shall also provide more or less

deep discussions, through examples, of the most important and most difficult part of doing this in a practical and useful way: the provision of good external interfaces. But we shall begin in Section 2, with some introductory remarks (not all strictly pertinent) concerning current T_EX's use and misuse of its formatting subsystems. The paper concludes with a note on non-T_EX math formatters, being an introduction to further study in this area.

The whole is permeated by a sincere plea for greatly increased support, of all types and by the many TUGs and all individuals in the T_EX com-

munity, for this outward-looking work so crucial to ensuring a very long life for both the great gifts that arrived within \TeX , and also for the inventive and experimental spirit of its pioneers. May this paper be the first of many that promote the creation of Really Useful Software from \TeX 's Inner Engineering (the RUSTIE project).

2 The structure of \TeX

The sum of the ‘document manipulation’ parts of \TeX can be well described as a text processing application whose originality and utility is provided by many specialised formatting engines. Many of these engines are barely recognisable as independent software artefacts due to the programming techniques wisely chosen by Knuth to implement the system [6]. (His methodology can these days be well described as the EO method: Extreme Optimisation, of both space and time!)

Each of these deeply embedded formatting engines will, when it has been disentangled from the sticky mess of connections and optimisations that holds it deep within \TeX 's embrace, reveal itself as a thing of beauty. Less appealing, each such newly disentangled creature, whilst happy to be breathing free, will also be a very vulnerable beast as it will have no communication interfaces through which to nurture it. Thus its liberators will need to carefully craft protective interfaces in order to ensure the fledgling formatter's viability in the real and exciting, but non-programmatic, world of 21st century documents.

These hidden gems include formatters for words, LR-boxes (with natural and many other width specifications), split-able [*sic*] LR-boxes, leaders of various types, paragraphs, split-able [*sic*] paragraphs, justified lines, formulas, superscripts, fractions, radicals with bar, etc.

One aspect of these specialised formatters clearly shows \TeX 's ancient pedigree, from a time when data flow had kept to a minimum. This is their lack of flexibility, in the following sense. With the exception of using the concept of ‘unset glue’, \TeX 's formatters will always combine to produce, from a given input, a unique, fully specified boxful of formatted output (often with a claim of its ‘optimal’ quality).

As Frank Mittelbach and I have copiously argued over the years, for a sophisticated document formatter a more useful product would be a reasonably sized collection of possible formattings that are all ‘good enough’. To this collection could possibly be added some ranking of their absolute quality but even more useful would be a few descriptors or quantisations of how good each is, together with other information that may be of use to other co-operating

formatters. Such output could then be used by other ‘higher-level’ formatters to choose the formatting most suited to their higher purposes.

I shall not pursue such valuable enhancements here as I am only asking for the Moon, not Mars, ... this year; the benefits of this approach to the practical optimisation of formatted documents have been long known and much discussed since Frank Mittelbach and I [18] introduced them.

3 Two (of many) examples

The two of \TeX 's many embedded formatters about which I have chosen to say a little more here are not necessarily either the most complex or the easiest to specify, but they are both central to \TeX 's *raison d'être*. I have somewhat presumptuously chosen to put these into the form of outline draft specifications. I hope that they will start a process that leads smoothly and quickly to full specifications and to the production of partial but usable ‘proof-of-concept’ library implementations. For the maths formatter some more detailed work has been done (Section 3.3) that will soon form the contents of a funding bid for the project.

3.1 The paragraph formatter

This is just one possible route towards the exposure of a \TeX -like paragraph mechanism and it is treated very briefly here. I hope it inspires others to help expand this to a full and carefully explained specification, to be published in this series.

Note that this outline description assumes the existence of methods for formatting ‘LR-boxes’: these are ‘single lines’ of text with well-defined spacing of ‘words’. In turn that formatter will require a ‘word formatter’, etc. I plan to provide a fuller explanation of these ideas in a future paper.

The inputs to this formatter would be as follows.

- The material to be formatted (see below).
- Parameter settings. Although it may not be essential, it would be very useful to have the ability to input values for all the parameters that are used by \TeX 's algorithm for forming and formatting a paragraph. Of course, many of them will have sensible default values that could be fixed or, in context, inherited.

The material to be formatted would consist of the following.

- Pure text (e.g. Unicode strings).
- Some ‘formatting information’ such as:
 - font selection hints
 - line-breaking hints
 - word-division information

- even direct formatting instructions (!) all with a precisely defined syntax.
- Pre-formatted material in the form of ‘in-line boxes’.

The basic liberated version of this formatter would return two types of material (this is slightly more than current \TeX ’s internal mechanism can be bothered to do, despite the information being available). Applications using it are free to ignore item 2.

1. A formatting of the paragraph: the formatted ‘lines’ of the paragraph plus other information about their layout in a well-defined format.
2. The number of lines, information about the break-points used (e.g. their location in the input string, whether a word was broken), an evaluation of how ‘good’ each break-point is and of the quality of the formatting of each line.

Note that here I have not explicitly discussed any of \TeX ’s sophisticated escape mechanisms that implement a small range of specialised extra activities, such as ‘adjust material’, marks or ‘whatsit nodes’. These are good examples of Knuth’s cleverly ad hoc use of small and tightly integrated extensions to a basic algorithm in order to emulate the effect of fully modelling these varied aspects of the document formatting process as separate modules. Such features of \TeX thus express a limited collection of ideas from a wider class that is important to automated document processing. Being ad hoc, Knuth’s efficient implementations typically use inappropriate models and hence have severe deficiencies. It is therefore probably not sensible to reproduce such escape mechanisms when providing our exposures of The Good Things of \TeX^{TM} .

A more advanced version of this liberated formatter would allow the replacement of \TeX ’s algorithm for finding break-points but this would entail provision for whole new parametrisations of the process and thus could decrease its immediate usability. A more practical way of providing this formatter may be to split it up into smaller modules that undertake distinct parts of the task, leaving the **master paragraph formatter** with only the two tasks of controlling the whole process and of handling all external interfaces (which would need to be customisable).

3.2 The mathematics formatter

This formatter has turned out to be the central feature in this final form of my diatribe; thus I shall warn you that it gets tediously detailed from here on. Although many of you would like to follow the example of Sebastian Rahtz [3] and banish all mathematics

from the \TeX world (if not all worlds), we must nevertheless face the reality that **\TeX Does Math!!**—both within the \TeX world but also, far more and growing rapidly, outside it, including much that will never see the guts of a \TeX processor. Hence if, over the next 10 years or so, real world ‘ \TeX for maths’ is not to lose all contact with ‘ \TeX the processor’, then \TeX ’s maths formatter must get out there and strut its stuff wherever maths is being stuffed into digital form.

For the liberated form of this important formatter it is more difficult to specify in suitably general terms the nature of the material to be formatted; the obvious specification is ‘Presentation MathML containing Unicode strings’ [16] but it is not clear to me at this stage whether this will always be sufficiently rich in information about the mathematical structure of the notation to be typeset. However, a lot of work has been done, and is continuing, on a wide range of uses of mathematical notation in computers, together with their associated description and formatting requirements. Exposing this formatter will greatly help many aspects of this research and development effort, in particular the task of determining what needs to be encoded in the input to ensure high quality maths output.

The output will be material that can be used by an ‘LR-box’ or ‘paragraph’ formatter.

The current understanding of the parametrisation needed for this task is also still somewhat empirical. It is known that standard \TeX ’s algorithm is severely under-parametrised for the tasks in which it claims supremacy but, in contrast, many applications will require only a far simpler parametrisation.

It is therefore desirable that the implementation of this algorithm should build in sensible default rules for determining plausible values of all \TeX ’s ‘maths parameters’ from data as meagre as just the nominal text font size. However, this liberation must at least remove all of current \TeX ’s explicit overloading in the ‘standard parameter set’ in order to be more generally usable. It may also be sensible to remove some aspects of the parametrisation from the current dependence on the choice of fonts.

The following subsection contains further details of this project; it is an extract from a funding proposal currently being pursued for this task. Throughout, the phrase ‘Standard \LaTeX ’ refers to a well known and defined (mainly ad hoc at present) subset of \LaTeX ’s math mode, but using the full range of Unicode maths characters, for encoding mathematical structures and glyphs. This is not a good phrase for this beast, thus it has also been dubbed ‘Lo \TeX ’ [17] (as in the common multiple *Lcm*).

3.3 Liberating T_EX: maths formatting

A fact and the obvious question: Many current applications, like the above specification, require a maths formatter: why not make it T_EX's?

Until quite recently the standard T_EX formatter for Computer Modern was the only such application that was both widely available and offered even reasonable typeset quality. However, bits of T_EX were, back then, totally incompatible with other applications that used different font resource technologies or series. Thus there are now many applications that, with varying degrees of success, attempt to emulate the quality of T_EX without its restrictions on input and output. There are now also some serious rivals to T_EX's typeset quality for mathematics but they also share two, at least, of T_EX's problems: being too far embedded in widely used and sophisticated document processing systems; and being closely linked to particular font series.

Whilst it may not be feasible to get the highest quality formatting using a given maths font series without careful choice of the set of layout parameters (and their values) specific to that series [7, 8], it is certainly possible to give the world T_EX's current algorithm with some liberation of the parametrisation. For example, it is straightforward to remove from its layout parameter set-up the many explicit (and 30-year old) overloadings and relationships that were necessary to Knuth's giving us this wonderful gift, so expertly tuned to a particular font series (Monotype Modern), itself about 100 years old![15]

More importantly, this quality can now be made available in a portable form so that it could be easily linked into any application, such as browsers or office suites, that need to render structured representations of mathematics, particularly MathML 3.0 and 'Standard L^AT_EX' or L^OT_EX. Post-liberation, this valuable treasure can be further enhanced by making it more configurable so as to allow extensions of its capabilities in the following two areas: the diversity of 1.5-dimensional (or maybe fully 2-dimensional) text-based layouts that it can construct; the range of fonts, glyphs, colour resources and other printing marks that can be used in these constructs (and maybe built-in interactivity).

For many modern applications it will be essential to provide (at least in a derivative version) optimisation for fast parallel rendering of a large collection of (typically small) maths fragments, possibly with different 'quality control requirements'.

A typical such application (from the trendy worlds of Web 2.0 and/or the Semantic Web) of the near-medium future will be fast, highly interac-

tive, agent-supported browsing of large collections of pages that contain a lot of connectivity and mathematical intelligence embedded in semantically rich encodings of mathematical notation [19].

3.3.1 Inputs

- Maths material encoded in a fully specified language that describes, at least, the presentational structure (or, visual semantics) of the material. This currently would typically be either a precisely defined (and large) subset of P-MML (Presentation MathML 3.0 [16]) or a syntactically precise subset of the currently used range of L^AT_EX-related math-mode syntax (L^OT_EX).¹

In order to support a more semantically oriented and user-friendly form of L^AT_EX input, it may be wise to provide a preprocessor that accepts precisely restricted uses of `\newcommand`.

- A modern 'maths font' resource (similar to OpenType with the necessary 'math tables' as supplied with Microsoft's Cambria Math font).

3.3.2 Outputs

This is not so easy to standardise. The high-level specification is that it will consist of 'glyphs plus rules' (plus, possibly, other simple graphical components and colour features) that are absolutely positioned in a local coordinate system relative to a reference or 'base point' that in turn can be used to position and orient the output on a page. It will also contain the necessary pointers to glyph rendering and paint resources, etc.

Since there is no widely accepted standard language for precisely such output, it will be necessary to use a simple fixed internal representation akin to T_EX's h/vlist or DVI languages. However, unlike the current T_EX paradigm, modules to convert this internal language to a range of commonly needed languages will also be included. Examples of outputs are thus some type of PDF and SVG fragments and other application-specific formats such as RTF or the internal renderable format of a web browser.

4 Other mathematics formatters

Time has mitigated against extending this section beyond these very brief comments on some important existing non-T_EX maths formatters. So there is another paper or two waiting in the wings.

For a long time there have been non-T_EX maths formatters in general use, such as *techexplorer* [20]. There are now a large number of these; here is a partial list of those that are definitely 'fit-to-purpose',

¹ There is also a far wider need for L^OT_EX, together with standard translations to/from P-MML.

be that quick, simple and clear rendering of simple maths in browsers, or high quality use of well-designed fonts for more classical paper presentation:

jmath, Mathplayer for Internet Explorer,
Gecko-Math (used in Firefox et al.),
Microsoft's Rich Edit (in Office 2007),
MathEX (incorporates techexplorer), SWiM.

How much the design of each of these systems uses, or is influenced by, Knuth's algorithm and layout rules ([5], Appendix G) has not I think been much studied; maybe the individual authors of the systems once knew the answers?

Only Microsoft's system [23, 2, 22] (as demonstrated in my talk at San Diego) claims typographical quality that is better than that of \TeX ; it is also the only one that is not intimately connected with MathML [16] although the two are reasonably friendly. The creators of this RichEdit system state that it uses ' \TeX 's mathematical typography principles' but they go on to remark that this task was nevertheless 'considerably harder than any of us imagined it would be', taking 15 years of elapsed time to complete (fortunately, throughout this time the then boss of the whole company took a keen personal interest in the whole project). They conclude that 'mathematical typography is very intricate and varied'.

Although the associated product does not seem to be widely used, the GtkMathView project [11] has worked on and documented a lot of interesting ideas and artefacts in this area.

References

- [1] Hermann Zapf. About micro-typography and the *hz*-program. *Electronic Publishing* 6(3), pages 283–288, Wiley, 1993.
<http://cajun.cs.nott.ac.uk/compsci/epo/papers/epoddaui.html>
- [2] About Rich Edit Controls.
[http://msdn.microsoft.com/en-us/library/bb787873\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/bb787873(VS.85).aspx)
- [3] Sebastian P. Q. Rahtz. Banish Maths! Daily personal communications, 1990–2005.
- [4] D. E. Knuth and M. F. Plass. Breaking paragraphs into lines. *Software – Practice and Experience*, 11(11), pages 1119–1184, 1981.
- [5] D. E. Knuth. *\TeX : The Program*. Addison-Wesley, 1986, 1993.
- [6] D. E. Knuth. *The \TeX book*. Addison-Wesley, 1986, 1993.
- [7] Richard Southall. Designing a new typeface with METAFONT, in *\TeX for Scientific Documentation*, Springer 1986, pages 161–179.
<http://www.springerlink.com/content/57432v516731367n/>
- [8] Peter Karow and Herman Zapf. Digital typography. Private communication, 2003.
- [9] John Plaice, Yannis Haralambous and Chris Rowley. An extensible approach to high-quality multilingual typesetting. In *RIDE-MLIM 2003*, IEEE Computer Society Press, 2003.
- [10] Extensible Markup Language (XML).
<http://www.w3c.org/XML>
- [11] GtkMathView Home Page.
<http://helm.cs.unibo.it/mml-widget>
- [12] X \TeX . <http://scripts.sil.org/xetex>
- [13] L \TeX : A document preparation system.
<http://www.latex-project.org>
- [14] Lua \TeX . <http://www.luatex.org>
- [15] Monotype Modern: Description. <http://www.paratype.com/fstore/default.aspx?fcode=871&search=Monotype+Modern>
- [16] Mathematical Markup Language (MathML) Version 3.0, W3C Working Draft.
<http://www.w3.org/TR/MathML3>
- [17] Chris Rowley and Stephen Watt. The need for a 'Standard L \TeX '. Private communication, July 2007.
- [18] Frank Mittelbach and Chris Rowley. The pursuit of quality: How can automated typesetting achieve the highest standards of craft typography?
In *Electronic Publishing*, pages 261–273, Cambridge University Press, 1992.
- [19] Christoph Lange and Michael Kohlhase. SWiM: A Semantic Wiki for Mathematical Knowledge Management.
Poster at <http://kwarc.info/projects/swim/pubs/poster-semwiki06.pdf>
- [20] Don DeLand. From \TeX to XML: The legacy of techexplorer and the future of math on the Web. Abstract in *TUGboat* 28:3, page 369, TUG, *Proceedings of the 2007 Annual Meeting*.
- [21] Unicode. <http://www.unicode.org>
- [22] Murray Sargent. Using RichEdit 6.0 for Math.
<http://blogs.msdn.com/murrays/archive/2007/10/28/using-richedit-6-0-for-math.aspx>
- [23] Ross Mills and John Hudson, Editors. *Mathematical Typesetting: Typesetting Solutions from Microsoft*. Glossy brochure distributed at TypeCon 2007, Seattle, August 1–5, 2007.

Why didn't METAFONT catch on?

Dave Crossland

University of Reading, UK

dave (at) lab6 dot com

<http://www.understandinglimited.com>

Abstract

METAFONT is an algebraic programming language for describing the shapes of letters, designed and implemented by Knuth as part of the original \TeX typesetting system. It was one of the earliest digital type design systems, and is completely capable of dealing with the letters of any writing system, has always been freely available, and is remarkably powerful. Yet it never caught on with type designers. Why?

“There are three kinds of people. Those that can count, and those that can't.”

There is type in typography, but there is also type in psychology: Personality type.

There are many ways of thinking about personality type [1] and the famous *Myers-Briggs* typology places importance on four attitudes. First, there is our preference for competition or cooperation, or whether we tend to make decisions logically or emotionally. Second, our use of language reveals the way we think, with some people preferring more abstract language and others preferring more concrete language. Third is our attitude to time keeping, which may be exploratory or scheduling, and fourth is our orientation to socialising, where after a party we may feel drained or energised.

Put together, these four preferences between two options yield 16 personality types. The book *Please Understand Me 2* [2] puts them into a cohesive system that groups the 16 types into four temperaments, fleshed out by labels and personified by Greek gods: Epimethean ‘Guardians,’ Dionysian ‘Artisans,’ Apollonian ‘Idealists,’ and Promethean ~~\TeX~~ ies: ‘Rationalists.’

Such broad theories for how people differ probably can't be taken too far, as ultimately people are all pretty much alike; *“what one man can do, another man can do.”* But there is a common sentiment that some of us are more abstract in our language and more logical in our thinking than others.

Software is pretty abstract and logical, and people who become immersed in the world of software tend to be of a Promethean temperament. The arguments for software freedom especially have that kind of draw. \TeX takes an abstract and logical approach to digital typography, from concept to usage, and METAFONT is no exception. But graphic design

is not abstract and logical, for the most part; it is visual, concrete, more emotional than logical.

Throughout the long history of desktop publishing, people have generally not found \TeX typesetting intuitive, preferring desktop publishing applications with graphical user interfaces. Even for those who go deep enough into graphic design to arrive at type design, METAFONT is almost entirely ignored — despite being freely available, completely capable, and remarkably powerful. I believe this issue of personality type is a primary reason why METAFONT has not caught on.

Let's consider type design divorced from the engineering of font software for a moment.

Design happens at various scales at the same time. In type design, the lowest visible level is that of the letter, where you are dealing mainly with the black shapes of the letter. It is obvious what those are, but there are also the ‘white’ shapes. If you are not sure what that means, imagine an image of a letter, and invert it so that the black becomes white and the white becomes black. Now, look around you to find a letter printed large on something like a poster or book cover. Looking at the letter, shift your awareness to the ‘negative space’ in and around the letter, and bring these white shapes into perceptual focus. It is hard to describe them, but they are there, and designing them is as important as designing the black shapes.

The next level up is that of words. Here there are not only the white shapes inside and around the letters, but those *between* the letters. At this level we can also see patterns in the black shapes *across* letters; things that look similar, yet are not exactly the same.

Consider the lowercase n and h. These contain several similar shapes, but looking closely, you will

see that there are slight differences. Balancing these similarities and differences is a core part of the type design process. There are strong patterns in some sets of letters, weaker similarities in other sets, and some letters that are less typical, yet still look like they belong with the rest. The letter *s* is perhaps the most different, and Knuth wrote an interesting essay about the peculiarities of that letter [3].

Finally, there is the level of paragraphs. When a paragraph is set with a typeface, a different impression of the letters emerges. This must be taken into account at the other levels. Seen in this way, a type design is a collection of individual glyph shapes that fits together cohesively at all levels.

We can now see clearly the subtle distinction between a font and a typeface. The same typeface can be implemented in a variety of typesetting technologies — metal, software, even potato — with the end result appearing the same. A font is a typeface implemented in software. The term ‘software’ spans programs and data, and fonts are a peculiar kind of software because they are both programs *and* data, while normally the two have some separation. Examples of programs within fonts are TrueType hints and OpenType layout features; these instruct the computer to display the type in various ways. The data in a font is the glyph point data and metrics table data.

There are generally two approaches to implementing typefaces in software. The ‘outline’ approach involves drawing each letter by interactively placing points along its outline. This attempts to be a direct facsimile of drawing letters on paper. Interpolation between sets of outlines means this approach can handle the creation of large typeface families.

The ‘stroke’ approach is where each letter is constructed by specifying points along the path of a pen’s stroke, and the attributes of the pen’s nib at those points. Archetypal pieces can be designed and used like Lego blocks to construct whole glyphs, with refinements made for the individual requirements of each letter. With parametrisation to make the shared values of shapes easily adjustable, such as widening stems or modifying serifs, this approach can handle a large typeface family in a cohesive and powerful way.

Today the outline approach is dominant because it gives instant visual feedback and exacting control; it is direct and visceral. This means designing type at the level of individual letter shapes is intuitive and a typeface emerges quickly.

It is especially suited to implementing existing type designs where all the aspects have already

been thought out; the \TeX community provides a clear example of this in the *AMS Euler* project [4], where a team of Stanford students attempted to digitise a new type design for mathematics that Zapf had drawn on paper; the developers tried both approaches and felt tracing outlines was most appropriate. FontForge [5] is a vigorously developed free software font editor application for working in this way today.

While not suitable for implementing existing type designs, METAFONT’s abstract and logical nature makes it powerful for dealing with type at the level of words. While initially slow, it speeds up later stages of the design process, especially when covering very large character sets. I think it is ideally suited to developing new type designs where the designer is not sure of the precise look that they are trying to capture and want to experiment with a variety of sweeping changes to their design.

\TeX works [6] attempts to make \TeX typesetting more visual and interactive. While still abstract and logical compared to desktop publishing applications like Scribus [7], its user interface design and the Sync \TeX technology [8] tightly interconnect the code and the document, making \TeX more visual, interactive, concrete and emotional.

Today METAFONT source code is written, various programs are run to generate graphics, then another program is used to view them. These programs may be METAFONT, or METAFONT and then `mftrace` [9], or METAPOST [10] with MetaType1 [11]. All involve a whole long process that is similar to writing \TeX documents in the traditional manner. But with \TeX works, the \TeX source code is rendered into a document in near real-time, so there is a very quick Boyd cycle [12] between adjusting the typesetting code and seeing the document rendered.

Perhaps if there was a graphical user interface to visualise METAFONT code in near real-time, type designers who feel writing code is unintuitive could be more confident about doing so. The simple GUI shown by Sherif & Fahmy in their Arabic design work is an example of this [13]. It might even be feasible to have two-way interaction between code and rendering, as Inkscape [14] achieves for SVG. Perhaps then, METAFONT might catch on.

Dave Crossland is an international public speaker on software freedom and fonts, runs a small business doing type and information design and systems administration, and is a committee member of UK-TUG. He is currently studying at the University of Reading’s Department of Typography on the MA Typeface Design programme.

References

- [1] http://en.wikipedia.org/wiki/Category:Personality_typologies
- [2] Keirse, D. *Please Understand Me II: Temperament, Character, Intelligence*. 1998: Prometheus Nemesis.
- [3] Knuth, D. *Digital typography*. 1999: CSLI.
- [4] Knuth, D. & Zapf, H. *AMS Euler: A New Typeface for Mathematics*. 1989: Scholarly Publishing.
- [5] <http://fontforge.sourceforge.net>
- [6] <http://tug.org/texworks/>
- [7] <http://www.scribus.net>
- [8] <http://itexmac.sourceforge.net/SyncTeX.html>
- [9] <http://lilypond.org/mftrace/>
- [10] <http://www.tug.org/metapost.html>
- [11] <http://www.ctan.org/tex-archive/fonts/utilities/metatype1/>
- [12] Osinga, F. *Science, Strategy and War: The Strategic Theory of John Boyd*. 2006: Routledge.
- [13] Sherif, A. and Fahmy, H. Meta-designing parameterized Arabic fonts for AlQalam. In this volume, 435–443.
- [14] <http://www.inkscape.org>
- [15] <http://metafont.tutorial.free.fr>

Copyright © 2008 Dave Crossland. Verbatim copying and redistribution of this entire article is permitted, provided this notice is preserved.

Creating cuneiform fonts with MetaType1 and FontForge

Karel Píška

Institute of Physics of the ASCR, v. v. i.

CZ-182 21 Prague, Czech Republic

piska (at) fzu dot cz

Abstract

A cuneiform font collection covering Akkadian, Ugaritic and Old Persian glyph subsets (about 600 signs) has been produced in two steps. With MetaType1 we generate intermediate Type 1 fonts, and then construct OpenType fonts using FontForge. We describe cuneiform design and the process of font development.

1 Introduction

I am interested in scripts, alphabets, writing systems, and in fonts, their computer representation. Ten years ago I decided to create Type 1 fonts for cuneiform, and last year, to extend them to Unicode OpenType versions. In my older Type 1 version (1998/9) [4] the raw text was written ‘by hand’ and then directly compiled into Type 1 with `t1asm` from `t1utils` [10]. The glyph set consisted of several separate Type 1 components to cover Akkadian (according to Labat) in a ‘Neo-Assyrian’ form (three files), Ugaritic, and Old Persian.

Three books served as principal sources: two Akkadian syllabaries edited by R. Labat [1] and F. Thureau-Dangin [2], and the encyclopedia *The World’s Writing Systems* [3]. No scanning of pictures or clay tablets was performed. The fonts are based on a starting point — my simple design of *wedges* in three variant forms (see also Fig. 1, below):



Our aim is to use free and open source software to produce “open source fonts”. Thus, to create the fonts only non-proprietary tools have been employed: MetaType1; FontForge; `t1utils`, `gawk`; other standard Unix utilities such as `bash`, `sed`, `sort`, ... In the following sections we will explain the process of creating fonts and illustrate it with numerous examples.

2 Producing Type 1 with MetaType1

The MetaType1 package [6], developed by the authors of Latin Modern, \TeX Gyre and other font collections (B. Jackowski, J. Nowacki, P. Strzelczyk):

- runs METAPOST (any available version) to produce `eps` files with outlines for all glyphs;
- collects all the data into one Type 1 file.

The information about the font and its glyphs is described in the METAPOST source files; addi-

tional macros are defined in MetaType1 extensions or may be appended by the user. For illustrations see the examples below. An explanation of some technical details and techniques how to work with MetaType1 can be found in the tutorial written by Klaus Höppner [7], which also includes a simple complete example and Makefile.

2.1 Font description in MetaType1

As usual with METAFONT or METAPOST the compilation is invoked by a main control file — `naakc.mp`:

```
input fontbase;
use_emergency_turningnumber;
input naak.mpe;
maybeinput "naakc.mpd";
maybeinput "naakc.mph";
maybeinput "naug.mph";
maybeinput "naop.mph";
beginfont
maybeinput "naak.mpg";
maybeinput "naug.mpg";
maybeinput "naop.mpg";
endfont
```

Global font parameters may be defined in a font header file — `naakc.mph`:

```
% FONT INFORMATION
pf_info_familyname "NeoAssyrianClassicType1";
pf_info_weight "Medium";
pf_info_fontname "NeoAssyrianClassicType1";
pf_info_version "002.001";
pf_info_author "Karel Piska at fzu.cz 2008";
pf_info_italicangle 0;
pf_info_underline -100, 50;
pf_info_fixedpitch false;
pf_info_adl 750, 250, 0;
italic_shift:=0;
```

Internal glyph names and metric data can be assigned as follows:

```
% INTRODUCE CHARS
standard_introduce("ash.akk");
standard_introduce("hal.akk");
standard_introduce("mug.akk");
```

```

standard_introduce("zadim.akk");
standard_introduce("ba.akk");
standard_introduce("zu.akk");
.....
% METRICS
wd._ash.akk=240; ht._ash.akk=160; dp._ash.akk=0;
wd._hal.akk=340; ht._hal.akk=160; dp._hal.akk=0;
wd._mug.akk=380; ht._mug.akk=220; dp._mug.akk=0;
wd._zadim.akk=460; ht._zadim.akk=220;
    dp._zadim.akk=0;
wd._ba.akk=420; ht._ba.akk=240; dp._ba.akk=0;
wd._zu.akk=500; ht._zu.akk=240; dp._zu.akk=0;
.....

```

(We do not define the encoding in Type 1.)

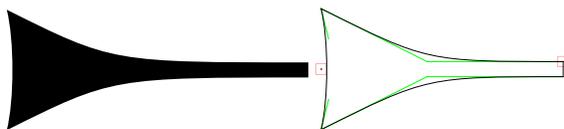
2.2 Glyph contours in MetaType1

Simple (atomic) elements — single wedges are defined by macros:

```

def wh(expr l,x,y)=
  r:=5; w:=40; b:=5; c:=20; d:=70;
  z[nw] 0=(x+l,y+r);
  z[nw] 0a=(x+d,y+r); z[nw] 1b=(x+d,y+r);
  z[nw] 1=(x,y+w);
  z[nw] 1a=(x+b,y+c); z[nw] 2b=(x+b,y-c);
  z[nw] 2=(x,y-w);
  z[nw] 2a=(x+d,y-r); z[nw] 3b=(x+d,y-r);
  z[nw] 3=(x+l,y-r);
  p[nw]=compose_path.z[nw](3);
  Fill p[nw];
  nw:=nw+1;
enddef;

```



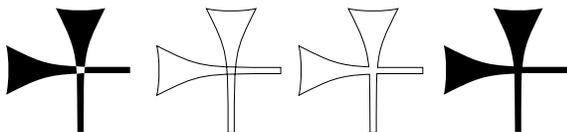
Another definition of a wedge — a single path:

```

def pwh(expr l,x,y)=
  (x+l,y+r)..controls(x+d,y+r)..(x,y+w)
  ..controls(x+b,y+c)and(x+b,y-c)..(x,y-w)
  ..controls(x+d,y-r)..(x+l,y-r)--cycle;
enddef;

```

In compound elements, the rendering of intersecting areas may depend on printer/viewer. Therefore, removing overlap in Type 1 (and probably also in OpenType) is required. We use the macro `find_outlines`:



```

def whv(expr x,y)=
  save pa,pb,pc; path pa,pb,pc;
  r:=5; w:=40; b:=5; c:=20; d:=70;
  pa:=pwh(200,x,y); pb=pvw(200,x+80,y+100);

```

```

find_outlines(pa,pb)(pc);
p[nw]:=pc1;
Fill p[nw];
nw:=nw+1;
enddef;

```

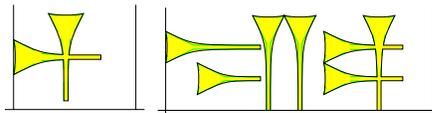
Complete glyphs — that is, cuneiform signs — are demonstrated in the following examples. Horizontal/vertical wedges, composites or their groups are also defined by macros; the arguments, for example, denote their lengths and coordinates.

```

beginlyph(_mash.akk);
  save p; path p[]; nw:=0;
  whv(0,120);
  standard_exact_hsbw("mash.akk");
endglyph;

beginlyph(_sag.akk);
  save p; path p[]; nw:=0;
  wh(240,0,160);
  wh(160,80,80);
  wv(240,220,240);
  wv(240,300,240);
  whhv(400,120);
  standard_exact_hsbw("sag.akk");
endglyph;

```



2.3 Generating Type 1

An intermediate Type 1 is generated from scratch:

```

FN=$1 # font file name
MT1=$2 # MetaType1 direction
mpost '\generating:=0;' input $FN.mp
gawk -f $(MT1)/mp2pf.awk \
  -v CD=$(MT1)/pfcommon.dat -v NAME=$FN
gawk -f $(MT1)/packsubr.awk -v LEV=5 \
  -v OUP=$FN.pn $FN.p
t1asm -b $FN.pn $FN.pfb

```

with some simplifications (intentional for cuneiform): glyph design is simple; no kerning pairs are needed; the characters occupy independent boxes; no hyphenation; and no internal encoding in the intermediate Type 1 is defined. Theoretically, we could use a common Type 1 with several external encoding vectors, but in practice, joining all the glyphs into one OpenType font is a better and simpler solution.

3 FontForge and producing OpenType

Scripts in the FontForge scripting language read Type 1, build data for OpenType (especially, define the encoding) and then generate OTF and TTF files. Here is a table showing the Unicode areas with which we are concerned:

Definition of encoding (Unicode)

Plane 0
 U+0020–U+007F ASCII block
 Plane 1
 Cuneiform ranges “Standard” Unicode
 U+10380–U+1039F Ugaritic
 U+103A0–U+103D7 Old Persian
 U+12000–U+123FF Cuneiform signs
 U+12500–U+1277F Neo-Assyrian glyph container
 (temporary “Private Area”)

To begin, we introduce a new Unicode font:

```
#!/usr/bin/fontforge
# 1.sfd, 2.names, 3.pfb, 4.otf, 5.ttf
New();Reencode("UnicodeFull")
SetFontNames($2,$2,$2)
#SetFontOrder(3); # cubic
#SetFontOrder(2); # quadratic
ScaleToEm(250)
Save($1)
...
```

Then we copy glyphs from Type 1 to OpenType: we open and read a Type 1 font and access glyphs by name (in Type 1) and copy them to appropriate locations addressed by Unicode numbers:

```
Open($3);Select("ash.akk");Copy();Close();\
Open($1);Select("u12501");Paste();
Save($1);Close();
Open($3);Select("hal.akk");Copy();Close();\
Open($1);Select("u12502");Paste();
Save($1);Close();
Open($3);Select("mug.akk");Copy();Close();\
Open($1);Select("u12503");Paste();
Save($1);Close();
...
```

A FontForge user command eliminates the repetition:

```
#!/usr/bin/fontforge # copy.pe
# SF source font, SG source glyph
# DF destination font, DG dest. glyph
Open($1);Select($2);Copy();Close();
Open($3);Select($4);Paste();
Save($3);
```

with references for the Neo-Assyrian block:

```
# $SF is source font
# $SFD temporary font (internal)
./copy.pe $SF "ash.akk" $SFD u12501
./copy.pe $SF "hal.akk" $SFD u12502
./copy.pe $SF "mug.akk" $SFD u12503
...
```

The Neo-Assyrian glyphs are allocated in the container; existing glyphs are linked from the Cuneiform range by references:

```
Select("u12743");CopyReference();
Select("u12000");Paste();
Select("u1264E");CopyReference();
Select("u12009");Paste();
```

```
Select("u12580");CopyReference();
Select("u1200A");Paste();
...
```

This operation may also be executed using a FontForge routine:

```
#!/usr/bin/fontforge # addref.pe
# 1. font, 2. glyph point in container
# 3. reference point
# addref.pe $fontname.sfd u12743 u12000
Open($1);Select($2);CopyReference();
Select($3);Paste();Save($1);
```

and then:

```
addref.pe $FN u12743 u12000
addref.pe $FN u1264E u12009
addref.pe $FN u12580 u1200A
...
```

Generating OpenType itself completes step 2: we can generate both OTF and TTF.

```
# $4 is OTF, $5 is TTF
Open($1);Generate($4); # with options
Open($1);Generate($5); # with options
```

Unfortunately, the glyph repertoire does not correspond to Unicode because, first, more than 300 glyphs do not have Unicode code points, and, on the other hand, my fonts cover only about 20% of the Unicode Sumerian-Akkadian cuneiform range (cuneiform signs and numeric signs).

In the final OpenType fonts, PostScript glyph names are omitted, the Akkadian glyph container (OTF/a) contains all Neo-Assyrian glyphs (according to Labat), partly defined by references in Unicode cuneiform block (OTF/c). Here is a table showing some of the correspondences:

PostScript	OTF/a	OTF/c
ash.akk	12501	u12038
hal.akk	12502	u1212C
mug.akk	12503	u1222E
zadim.akk	12504	
ba.akk	12505	u12040
zu.akk	12506	u1236A
su.akk	12507	u122E2
shum.akk	12508	
bal.akk	12509	u12044
adII.akk	1250A	
bulII.akk	1250B	
tar.akk	1250C	u122FB
an.akk	1250D	u1202D
ka.akk	1250F	u12157

3.1 Hinting

The OpenType output was “satisfactory” as autohinted with FontForge (Fig. 1); no hinting instructions are included in the TrueType fonts.

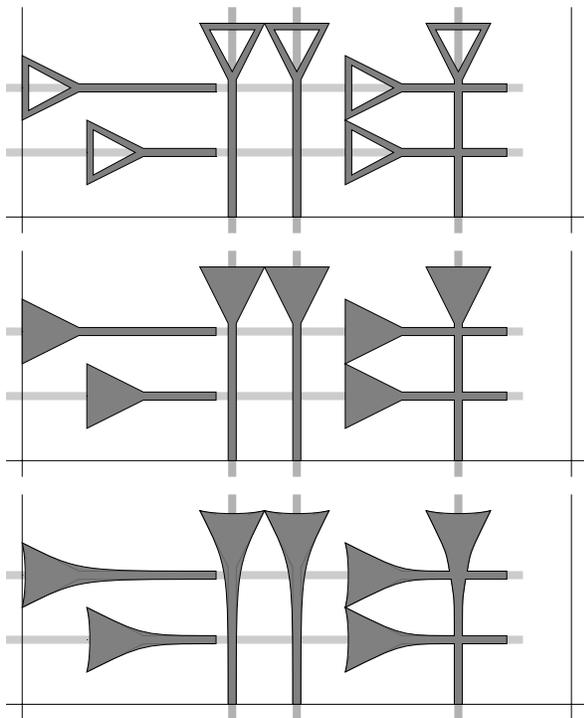


Figure 1: Wedge design in three variants with hinting.

4 Support for (pdf)LaTeX and XeTeX

Old and simple LaTeX macros for Type 1 fonts and (pdf)LaTeX were modified for XeLaTeX to bind symbolic glyph names using their Unicode numbers.

```
\def\NAfontC#1{%
    xakkadian.sty
    \font\NAC="[nacunc.ttf]" at #1pt
}
\def\NAfont{\NAfontC{10}} % default font
%
\def\AKK{%
\NAfont%
\def\ash{{\NAC\char"12501}}%
\let\dil\ash\let\tilIIII\ash\let\ttil\ash%
\let\rum\ash\let\ruIII\ash\let\ina\ash%
\let\asIII\ash\let\azIII\ash%
\def\hal{{\NAC\char"12502}}\let\buluh\hal%
\def\mug{{\NAC\char"12503}}%
\let\muk\mug\let\muq\mug\let\puk\mug%
\def\zadim{{\NAC\char"12504}}%
\def\ba{{\NAC\char"12505}}%
\let\paII\ba%
\def\zu{{\NAC\char"12506}}%
\let\ssuII\zu%
...

```

The two following examples show font usage.

1. Glyph index, numbers correspond to Labat [1]:

- a a: 579/1 (579-NAc67)
- a' aI: 397 (397-NAb141)
- á aII: 334 (334-NAb78)
- à aIII: 383 (383-NAb127) see pi
- a₄ aIIII: 579/2 (582-NAc70) see àm
- a₇ aVII: 589 (589-NAc77) see ha
- aa aa: 579/6 (587-NAc75)
- ab ab: 128 (128-NAa128)
- áb abII: 420 (420-NAb164) see lid
- ablal ablal: 525 (525-NAc13)
- ad ad: 145 (145-NAa145)
- ád adII: 10 (10-NAa10)

2. Sample text in Akkadian (with transliteration):

```

\ash\kur\nu\giIIII \a \gaq\qa\ri [ \la \ta\ari \ ]
a.na kur.nu.gi4 a qaq.qa.ri [ la ta.a.ri ]


\DETD\innana\dumu\miII \DETD\sin \uII\zu\un\shaII [ \ish\kun\ ]
dinnana.dumu.midsin ú.zu.un.šá [ iš.kun ]


\ish\kun\ma \dumu\miII \DETD\sin \uII\zu\un[\sha]
iš.kun.ma dumu.midsin ú.zu.un.[ša]


\ash\na \eII \e\tte\shu\bat \DETD\ir\kal\la
a.na é e.še.su.batdir.kal.la


\ash\na \eII \sha \e\ri\bu\shuII \la \a\ssu\ui
a.na é ša e.ri.bu.šú la a.šu.ú


\ash\na \har\ra\ni \sha \a\lak\ta\shaII \la \ta\ta\arat
a.na har.ra.ni ša a.lak.ta.šá la ta.a.a.rat


\ash\na \eII \sha \e\ri\bu\shuII \zu\um\mu\ui \nu\ui\ra
a.na é ša e.ri.bu.šú zu.um.mu.ú nu.ú.ra


\ash\shar \sahar\haII \bu\bu\us\su\nu \a\kal\shu\nu \tti\itt\ttu
a.šar sahar.há bu.bu.us.su.nu a.kal.šu.nu ti.it.tu

Line 1: Akkadian text using the cuneiform font
Line 2: The corresponding source input in the LaTeX command {\AKK source }
Line 3: Transliteration (dots and spaces added manually)

```

5 Conclusion

Both METAFONT and MetaType1 (=METAPOST) are programmable. But METAFONT produces only bitmaps, while in MetaType1, we must not define areas to fill or unfill with bitmap matrices which would depend on the device (resolution, blacker and other parameters). Rather, we are restricted to outlines:

- glyphs must be defined by closed curves, i.e. sequences of splines;
- we produce the Type 1 format directly;
- the MetaType1 commands Fill/Unfill denote the output of curves in the PostScript Type 1

representation with proper path direction and correct order of spline segments;

- final filling/unfilling is delegated to the PostScript/PDF rasterization systems.

Between PFB (Type 1) and OTF (PS/CFF flavored OpenType) we can find only formal differences in internal representation and organization; the mathematical outline curves and hints are identical. On the other hand, OTF and TTF (TrueType flavored OpenType) may differ in approximation of curve segments, since the underlying representations use cubic and quadratic polynomials, respectively. For our simple cuneiform design of wedges, though, a common approximation is workable. \LaTeX can read all font formats: \TeX fonts, OTF, TTF, etc.; OpenOffice 2.3 (on my computer) can work only with TTF. (I can say nothing about MS Word because I do not have this product.)

MetaType1 and FontForge give the advantage of programmability with open source data. In FontForge, the interactive approach in glyph design is dominant; theoretically we could define glyph outlines in the FontForge scripting language but it would be very difficult and inefficient. METAFONT/METAPOST (MetaType1) are more flexible and modular: they allow for solving mathematical equations, common processing and maintenance of related fonts, automatic calculation of parameters, and systematic modifications.

A typical task for MetaType1 is to combine a small number of components into many composite glyphs uniformly. This is common for “special kinds of fonts”: just as Latin Modern and \TeX Gyre can combine letters + accents, the cuneiform fonts can combine wedges; operations to produce composite glyphs can be defined and applied in a simple way, and generation and maintenance can be repeated for numerous fonts.

The older non-Unicode versions of cuneiform fonts have been already referenced in the subsection “External links / Fonts” in http://en.wikipedia.org/wiki/Cuneiform_script (a web search for “cuneiform” should find it also). They have been already used by scholars; e.g. for syllabaries and computer transliteration of sample texts for students.

Now I plan to finish and publish the new “Unicode” version, by extending the glyph repertoire to other glyphs and other shapes, corresponding to other languages and their historical period. Preliminary experimental OpenType fonts are available on my web site [11].

My final wish is that the MetaType1 package would be extended to “MetaOpenType” to produce OpenType font formats directly.

6 Acknowledgements

I want to thank the authors of MetaType1, FontForge (G. Williams) and other developers and maintainers of free and open source software.

References

- [1] René Labat. *Manuel d'épigraphie akkadienne*. Troisième édition. Imprimerie nationale, Paris, 1959.
- [2] F. Thureau-Dangin. *Le syllabaire accadien*. Librairie Orientaliste Paul Geuthner, Paris, 1926.
- [3] *The World's Writing Systems*. P. T. Daniels and W. Bright, eds. Oxford University Press, New York–Oxford, 1996.
- [4] Karel Piška. Fonts for Neo-Assyrian Cuneiform. *Proceedings of the EuroTEX Conference (Paperless TEX)*, Heidelberg, Germany, September 20–24, 1999, Günter Partosch and Gerhard Wilhelms, eds. Gießen, Augsburg, 1999, ISSN 1438-9959, 142–154. <http://www-hep.fzu.cz/~piska/cuneiform.html>
- [5] Cuneiform script (Wikipedia). http://en.wikipedia.org/wiki/Cuneiform_script
- [6] Bogusław Jackowski, Janusz M. Nowacki, Piotr Strzelczyk. Programming PostScript Type 1 fonts using MetaType1: Auditing, enhancing, creating. *Proceedings of EuroTEX 2003*, Brest, France, 24–27 June 2003. *TUGboat* 24:3, pp. 575–581, <http://tug.org/TUGboat/Articles/tb24-3/jackowski.pdf>; CTAN:fonts/utilities/metatype1; <ftp://bop.eps.gda.pl/pub/metatype1>.
- [7] Klaus Höppner. Creation of a PostScript Type 1 logo font with MetaType1. *Proceedings of XVII European TEX Conference, 2007*. *TUGboat* 29:1, pp. 34–38, <http://tug.org/TUGboat/Articles/tb29-1/tb91hoepner.pdf>.
- [8] George Williams. Font creation with FontForge. *EuroTEX 2003 Proceedings*, *TUGboat* 24:3, pp. 531–544, <http://tug.org/TUGboat/Articles/tb24-3/williams.pdf>; <http://fontforge.sourceforge.net>.
- [9] Free Software Foundation. GNU awk, <http://www.gnu.org/software/gawk>.
- [10] Eddie Kohler. *t1utils* (Type 1 tools), <http://freshmeat.net/projects/t1utils>.
- [11] <http://www-hep.fzu.cz/~piska/cuneiform/opentype.html>

Do we need a ‘Cork’ math font encoding?

Ulrik Vieth

Vaihinger Straße 69

70567 Stuttgart

Germany

ulrik dot vieth (at) arcor dot de

Abstract

The city of Cork has become widely known in the \TeX community, ever since it gave name to an encoding developed at the European \TeX conference of 1990. The ‘Cork’ encoding, as it became known, was the first example of an 8-bit text font encoding that appeared after the release of \TeX 3.0, and was later followed by a number of other encodings based on similar design principles.

As of today, the ‘Cork’ encoding represents only one out of several possible choices of 8-bit subsets from a much larger repertoire of glyphs provided in fonts such as Latin Modern or \TeX Gyre. Moreover, recent developments of new \TeX engines are making it possible to take advantage of OpenType font technology directly, largely eliminating the need for 8-bit font encodings altogether.

During the entire time since 1990 math fonts have always been lagging behind the developments in text fonts. While the need for new math font encodings was recognized early on and while several encoding proposals have been discussed, none of them ever reached production quality or became widely used.

In this paper, we review the situation of math fonts as of 2008, especially in view of recent developments of Unicode and OpenType math fonts such as the STIX fonts or Cambria Math. In particular, we try to answer the question whether a ‘Cork’ math font encoding is still needed or whether Unicode and OpenType might eliminate the need for \TeX -specific math font encodings.

1 History and development of text fonts

1.1 The ‘Cork’ encoding

When the 5th European \TeX conference was held in Cork in the summer of 1990, the \TeX community was undergoing a major transition phase. \TeX 3.0 had just been released that year, making it possible to switch from 7-bit to 8-bit font encodings and to support hyphenation for multiple languages.

Since the ability to properly typeset and hyphenate accented languages strongly depended on overcoming the previous limitations, European \TeX users wanted to take advantage of the new features and started to work on new font encodings [1, 2, 3]. As a result, they came up with an encoding that became widely known as the ‘Cork’ encoding, named after the site of the conference [4].

The informal encoding name ‘Cork’ stayed in use for many years, even after \LaTeX 2 ϵ and NFSS2 introduced a system of formal encoding names in 1993–94, assigning OTn for 7-bit old text encodings, Tn for 8-bit standard text encodings, and Ln for local or non-standard encodings [5]. The ‘Cork’ encoding

was the first example of a standard 8-bit text font and thus became the T1 encoding.

While the ‘Cork’ encoding was certainly an important achievement, it also introduced some novel features that may have seemed like a good idea at that time but would be seen as shortcomings or problems from today’s point of view, after nearly two decades of experience with font encodings.

In retrospect, the ‘Cork’ encoding represents a typical example of the \TeX -specific way of doing things of the early 1990s without much regard for standards or technologies outside the \TeX world.

Instead of following established standards, such as using ISO Latin 1 or 2 or some extended versions for Western and Eastern European languages, the ‘Cork’ encoding tried to support as many languages as possible in a single font encoding, filling the 8-bit font table to the limit with accented characters at the expense of symbols. Since there was no more room left in the font table, typesetting symbols at first had to be taken from the old 7-bit fonts, until a supplementary text symbol TS1 encoding [6] was introduced in 1995 to fill the gap.

When it came to implementing the T1 and TS1 encodings for PostScript fonts, it turned out that the encodings were designed without taking into account the range of glyphs commonly available in standard PostScript fonts.

Both font encodings could only be partially implemented with glyphs from the real font, while the remaining slots either had to be faked with virtual fonts or remain unavailable. At the same time, none of the encodings provided access to the full set of available glyphs from the real font.

1.2 Alternatives to the ‘Cork’ encoding

As an alternative to using the T1 and TS1 encodings for PostScript fonts, the TeXnANSI or LY1 encoding was proposed [7], which was designed to provide access to the full range of commonly available symbols (similar to the TeXBase1 encoding), but also matched the layout of the OT1 encoding in the lower half, so that it could be used as drop-in replacement without any need for virtual fonts.

In addition to that, a number of non-standard encodings have come into use as local alternatives to the ‘Cork’ encoding, such as the Polish QX, the Czech CS, and the Lithuanian L7X encoding, each of them trying to provide better solutions for the needs of specific languages.

In summary, the ‘Cork’ encoding as the first example of an 8-bit text encoding (T1) was not only followed by additional encodings based on the same design principles for other languages (Tn), but also supplemented by a text symbol encoding (TS1) and complemented by a variety of local or non-standard encodings (LY1, QX, CS, etc.).

As became clear over time, the original goal of the ‘Cork’ encoding of providing a single standard encoding for as many languages as possible couldn’t possibly be achieved within the limits of 8-bit fonts, simply because there are far too many languages and symbols to consider, even when limiting the scope to Latin and possibly Cyrillic or Greek.

2 Recent developments of text fonts

2.1 Unicode support in new TeX fonts

It was only in recent years that the development of the Latin Modern [8, 9, 10] and TeX Gyre fonts [11, 12] has provided a consistent implementation for all the many choices of encodings.

As of today, the ‘Cork’ encoding represents only one out of several possible 8-bit subsets taken from a much larger repertoire of glyphs. The full set of glyphs, however, can be accessed only when moving

beyond the limits of 8-bit fonts towards Unicode and OpenType font technology.

2.2 Unicode support in new TeX engines

As we are approaching the TUG 2008 conference at Cork, the TeX community is again undergoing a major transition phase. While TeX itself remains frozen and stable, a number of important developments have been going on in recent years.

Starting with the development of PDFTeX since the late 1990s the use of PDF output and scalable PostScript or TrueType fonts has largely replaced the use of DVI output and bitmap PK fonts.

Followed by the ongoing development of XeTeX and LuaTeX in recent years the use of Unicode and OpenType font technology is also starting to replace the use of 8-bit font encodings as well as traditional PostScript or TrueType font formats.

Putting everything together, the development of new fonts and new TeX engines in recent years has enabled the TeX community to catch up with developments of font technology in the publishing industry and to prepare for the future.

The only thing still missing (besides finishing the ongoing development work) is the development of support for Unicode math in the new TeX engines and the development of OpenType math fonts for Latin Modern and TeX Gyre.

3 History and development of math fonts

When TeX was first developed in 1977–78, the 7-bit font encodings for text fonts and math fonts were developed simultaneously, since both of them were needed for typesetting mathematical textbooks like *The Art of Computer Programming*.

When TeX 3.0 made it possible to switch from 7-bit to 8-bit font encodings, it was the text fonts driving these new developments while the math fonts remained largely unchanged.

As a result, the development of math fonts has been lagging behind the corresponding text fonts for nearly two decades now, ever since the development of the ‘Cork’ encoding started in 1990.

In principle, a general need for new math fonts was recognized early on: When the first implementations of ‘Cork’ encoded text fonts became available, it was soon discovered that the new 8-bit text fonts couldn’t fully replace the old 7-bit text fonts without resolving the inter-dependencies between text and math fonts. In practice, however, nothing much happened since there was no pressing need.

3.1 The ‘Aston’ proposal

The first bit of progress was made in the summer of 1993, when the L^AT_EX3 Project and some T_EX users group sponsored a research student to work on math font encodings for a few months.

As a result, a proposal for the general layout of new 8-bit math font encodings was developed and presented at TUG 1993 at Aston University [13]. Unlike the ‘Cork’ encoding, which became widely known, this ‘Aston’ proposal was known only to some insiders and went largely unnoticed.

After only a few months of activity in 1993 the project mailing list went silent and nothing further happened for several years, even after a detailed report was published as a L^AT_EX3 Project Report [14].

3.2 The ‘newmath’ prototype

The next bit of progress was made in 1997–98, when the ideas of the ‘Aston’ proposal were taken up again and work on an implementation was started.

This time, instead of just discussing ideas or preparing research documents, the project focussed on developing a prototype implementation of new math fonts for several font families using a mixture of METAFONT and fontinst work [15].

When the results of the project were presented at the EuroT_EX 1998 conference [16], the project was making good progress, although the results were still very preliminary and far from ready for production.

Unfortunately, the project then came to a halt soon after the conference when other activities came to the forefront and changed the scope and direction of the project [17, 18].

Before the conference, the goal of the project had been to develop a set of 8-bit math font encodings for use with traditional T_EX engines (within the constraints of 16 families of 256 glyphs) and also to provide some example implementations by means of reencoding and enhancing existing font sets.

After the conference, that goal was set aside and put on hold for an indefinite time by the efforts to bring math into Unicode.

4 Recent developments of math fonts

4.1 Unicode math and the STIX fonts

While the efforts to bring math into Unicode were certainly very important, they also brought along a lot of baggage in the form of a very large number of additional symbols, making it much more work to provide a reasonably complete implementation and nearly impossible to encode all those symbols within the constraints of traditional T_EX engines.

In the end, the Unicode math efforts continued over several years until the symbols were accepted [19, 20] and several more years until an implementation of a Unicode math font was commissioned [21] by a consortium of scientific and technical publishers, known as the STIX Project.

When the first beta-test release of the so-called STIX fonts [22] finally became available in late 2007, nearly a decade had passed without making progress on math font encodings for T_EX.

While the STIX fonts provide all the building blocks of Unicode math symbols, they are still lacking T_EX support and may yet have to be repackaged in a different way to turn them into a usable font for use with T_EX or other systems.

Despite the progress on providing the Unicode math symbols, the question of how to encode all the many Unicode math symbols in a set of 8-bit font encodings for use with traditional T_EX engines still remains unresolved. Most likely, only a subset of the most commonly used symbols could be made available in a set of 8-bit fonts, whereas the full range of symbols would be available only when moving to Unicode and OpenType font technology.

4.2 OpenType math in MS Office 2007

While the T_EX community and the consortium of scientific publishers were patiently awaiting the release of the STIX fonts before reconsidering the topic of math font encodings, outside developments have continued to move on. In particular, Microsoft has moved ahead and has implemented its own support for Unicode math in Office 2007.

They did so by adding support for math typesetting in OpenType font technology [23, 24] and by commissioning the design of the Cambria Math font as an implementation of an OpenType math font [25, 26, 27]. In addition, they have also adopted an input language called ‘linear math’ [28], which is strongly based on T_EX concepts.

While OpenType math is officially still considered experimental and not yet part of the OpenType specification [29], it is already a *de facto* standard, not only because it has been deployed to millions of installations of Office 2007, but also because it has already been adopted by other projects, such as the FontForge font editor [30] and independent font designs such as Asana Math [31].

In addition, the next release of the STIX fonts scheduled for the summer of 2008 is also expected to include support for OpenType math.

4.3 OpenType math in new T_EX engines

At the time of writing, current development versions of X_YT_EX have added some (limited) support for OpenType math, so it is already possible to use fonts such as Cambria Math in X_YT_EX [32], and this OpenType math support will soon become available to the T_EX community at large with the upcoming release of T_EX Live 2008.

Most likely, LuaT_EX will also be adding support for OpenType math eventually, so OpenType math is likely to become a *de facto* standard in the T_EX world as well, much as we have adopted other outside developments in the past.

4.4 OpenType math for new T_EX fonts?

Given these developments, the question posed in the title of this paper about the need for new math font encodings may soon become a non-issue.

If we decide to adopt Unicode and OpenType math font technology in new T_EX engines and new fonts, the real question is no longer how to design the layout of encoding tables but rather how to deal with the technology of OpenType math fonts, as we will discuss in the following sections.

5 Future developments in math fonts

5.1 Some background on OpenType math

The OpenType font format was developed jointly by Microsoft and Adobe, based on concepts adopted from the earlier TrueType and PostScript formats. The overall structure of OpenType fonts shares the extensible table structure of TrueType fonts, adding support for different flavors of glyph descriptions in either PostScript CFF or TrueType format.¹

One of the most interesting points about OpenType is the support for ‘advanced’ typographic features, supporting a considerable amount of intelligence in the font, enabling complex manipulations of glyph positioning or glyph substitutions. At the user level, many of these ‘advanced’ typographic features can be controlled selectively by the activation of so-called OpenType feature tags.

Despite its name, the OpenType font format is not really open and remains a vendor-controlled specification, much like the previous TrueType and PostScript font formats developed by these vendors. The official OpenType specification is published on a Microsoft web site at [29], but that version may not necessarily reflect the latest developments.

¹ An extensive documentation of the OpenType format and its features as well as many other important font formats can be found in [33].

In the case of OpenType math, Microsoft has used its powers as one of the vendors controlling the specification to implement an extension of the OpenType format and declare it as ‘experimental’ until they see fit to release it. Fortunately, Microsoft was smart enough to borrow from the best examples of math typesetting technology when they designed OpenType math, so they chose T_EX as a model for many of the concepts of OpenType math.

5.2 The details of OpenType math

The OpenType MATH table One of the most distinctive features of an OpenType math font is the presence of a MATH table. This table contains a number of global font metric parameters, much like the `\fontdimen` parameters of math fonts in T_EX described in Appendix G of *The T_EXbook*.

In a traditional T_EX setup these parameters are essential for typesetting math, controlling various aspects such as the spacing of elements such as big operators, fractions, and indices [34, 35].

In an OpenType font the parameters of the MATH table have a similar role for typesetting math. From what is known, Microsoft apparently consulted with Don Knuth about the design of this table, so the result is not only similar to T_EX, but even goes beyond T_EX by adding new parameters for cases where hard-wired defaults are applied in T_EX.

In the X_YT_EX implementation the parameters of the OpenType MATH table are mapped internally to T_EX’s `\fontdimen` parameters. In most cases this mapping is quite obvious and straight-forward, but unfortunately there are also a few exceptions where some parameters in T_EX do not have a direct correspondence in OpenType. It is not clear, however, whether these omissions are just an oversight or a deliberate design decision in case a parameter was deemed irrelevant or unnecessary.

Support for OpenType math in X_YT_EX still remains somewhat limited for precisely this reason; until the mapping problems are resolved, X_YT_EX has to rely on workarounds to extract the necessary parameters from the OpenType MATH table.

At the time of writing, the extra parameters introduced by OpenType generalizing the concepts of T_EX have been silently ignored. It is conceivable, however, that future extensions of new T_EX engines might eventually start to use these parameters in the math typesetting algorithms as well.

In the end, whatever technology is used to typeset OpenType math, it remains the responsibility of the font designer to set up the values of all the many parameters affecting the quality of math typesetting. Unfortunately, for a non-technical designer such a

task feels like a burden, which is better left to a technical person as a font implementor.

For best results, it is essential to develop a good understanding of the significance of the parameters and how they affect the quality of math typesetting. In [35] we have presented a method for setting up the values of metric parameters of math fonts in \TeX . For OpenType math fonts, we would obviously have to reconsider this procedure.

Font metrics of math fonts Besides storing the global font metric parameters, the OpenType MATH table is also used to store additional glyph-specific information such as italic corrections or kern pairs, as well as information related to the placement of math accents, superscripts and subscripts.

In a traditional \TeX setup the font metrics of math fonts have rather peculiar properties, because much of the glyph-specific information is encoded or hidden by overloading existing fields in the TFM metrics in an unusual or non-intuitive way [36].

For example, the width in the TFM metrics is not the real width of the glyph. Instead, it is used to indicate the position where to attach the subscript. Similarly, the italic correction is used to indicate the offset between subscript and superscript.

As another example, fake kern pairs involving a skewchar are used to indicate how much the visual center of the glyphs is skewed in order to determine the position where to attach a math accent.

In OpenType math fonts all such peculiarities will become obsolete, as the MATH table provides data structures to store all the glyph-specific metric information in a much better way. In the case of indices, OpenType math has extended the concepts of \TeX by defining ‘cut-ins’ at the corners on both sides of a glyph and not just to the right.

Unfortunately, while the conceptual clarity of OpenType math may be very welcome in principle, it may cause an additional burden on font designers developing OpenType math fonts based on traditional \TeX fonts (such as the Latin Modern fonts) and trying to maintain metric compatibility.

In such cases it may be necessary to examine the metrics of each glyph and to translate the original metrics into appropriate OpenType metrics.

Font encoding and organization The encoding of OpenType fonts is essentially defined by Unicode code points. Most likely, a typical OpenType math font will include only a subset of Unicode limited to the relevant ranges of math symbols and alphabets, while the corresponding text font may contain a bigger range of scripts.

In a traditional \TeX setup the math setup consists of a series of 8-bit fonts organized into families. Typically, each font will contain one set of alphabets in a particular style and a selection of symbols filling the remaining slots.

In a Unicode setup the math setup will consist of only one big OpenType font, containing all the math symbols and operators in the relevant Unicode slots, as well as all the many styles of math alphabets assigned to slots starting at U+1D400.

As a result, there will be several important conceptual implications to consider in the design and implementation of OpenType math fonts, such as how to handle font switches of math alphabets, how to include the various sizes of big operators, delimiters, or radicals, or how to include the optical sizes of superscripts and subscripts.

Handling of math alphabets In a traditional \TeX setup the letters of the Latin and Greek alphabets are subject to font switches between the various math families, usually containing a different style in each family (roman, italic, script, etc.).

In a Unicode setup each style of math alphabets has a different range of slots assigned to it, since each style is assumed to convey a different meaning.

When dealing with direct Unicode input, this might not be a problem, but when dealing with traditional \TeX input, quite a lot of setup may be needed at the macro level to ensure that input such as a or a or \mathbf{a} will be translated to the appropriate Unicode slots.

An additional complication arises because the Unicode code points assigned to the math alphabets are non-contiguous for historical reasons [37]. While most of the alphabetic letters are taken from one big block starting at U+1D400, a few letters which were part of Unicode already before the introduction of Unicode math have to be taken from another block starting at U+2100.

An example implementation of a \LaTeX macro package for $X_{\text{Y}}\TeX$ to support OpenType math is already available [32], and it shows how much setup is needed just to handle math alphabets. Fortunately, such a setup will be needed only once and will be applicable for all Unicode math fonts, quite unlike the case of traditional \TeX fonts where each set of math fonts requires its own macro package.

Handling of size variants Ever since the days of DVI files and PK fonts, \TeX users have been accustomed to thinking of font encodings in terms of numeric slots in an encoding table, usually assuming a 1:1 mapping between code points and glyphs.

However, there have always been exceptions to this rule, most notably in the case of a math extension font, where special TFM features were used to set up a linked list from one code point to a series of next-larger glyph variants representing different sizes of operators, delimiters, radicals, or accents, optionally followed by an extensible version.

In a traditional \TeX font each glyph variant has a slot by itself in the font encoding, even if it was addressed only indirectly.

In an OpenType font, however, the font encoding is determined by Unicode code points, so the additional glyph variants representing different sizes cannot be addressed directly by Unicode code points and have to remain unencoded, potentially mapped to the Unicode private use area, if needed.

While the conceptual ideas of vertical and horizontal variants and constructions in the OpenType MATH table are very similar to the concepts of charlists and extensible recipes in \TeX font metrics, it is interesting to note that OpenType has generalized these concepts a little bit.

While \TeX supports extensible recipes only in a vertical context of big delimiters, OpenType also supports horizontal extensible constructions, so it would be possible to define an extensible overbrace or underbrace in the font, rather than at the macro level using straight line segments for the extensible parts. In addition, the same concept could also be applied to arbitrarily long arrows.

Optical sizes for scripts In a traditional \TeX setup math fonts are organized into families, each of them consisting of three fonts loaded at different design sizes representing text style and first and second level script style.

If a math font provides optical design sizes, such as in the case of traditional METAFONT fonts, these fonts are typically loaded at sizes of 10 pt, 7 pt, 5 pt, each of them having different proportions adjusted for improved readability at smaller sizes.

If a math font doesn’t provide optical sizes, such as in the case of typical PostScript fonts, scaled-down versions of the 10 pt design size will have to make do, but in such cases it may be necessary to use bigger sizes of first and second level scripts, such as 10 pt, 7.6 pt, 6 pt, since the font may otherwise become too unreadable at such small sizes.

In OpenType math the concept of optical sizes from \TeX and METAFONT has been adopted as well, but it is implemented in a different way, typical for OpenType fonts. Instead of loading multiple fonts at different sizes, OpenType math fonts incorporate the multiple design variants in the same font and

activate them by a standard OpenType substitution mechanism using a feature tag `ssty=0` and `ssty=1`, not much different from the standard substitutions for small caps or oldstyle figures in text fonts.

It is important to note that the optical design variants intended for use in first and second level scripts, using proportions adjusted for smaller sizes, are nevertheless provided at the basic design size and subsequently scaled down using a scaling factor defined in the OpenType MATH table.

If an OpenType math font lacks optical design variants for script sizes and does not support the `ssty` feature tag, a scaled-down version of the basic design size will be used automatically. The same will also apply to non-alphabetic symbols.

Use of OpenType feature tags Besides using OpenType feature tags for specific purposes in math fonts, most professional OpenType text fonts also use feature tags for other purposes, such as for selecting small caps or switching between oldstyle and lining figures. Some OpenType fonts may provide a rich set of features, such as a number of stylistic variants, initial and final forms, or optical sizes.

Ultimately, it remains to be seen how the use of OpenType feature tags will influence the organization of OpenType fonts for \TeX , such as Latin Modern or \TeX Gyre, not just concerning new math fonts, but also existing text fonts.

So far, the Latin Modern fonts have very closely followed the model of the Computer Modern fonts, providing separate fonts for each design size and each font shape or variant.

While it might well be possible to eliminate some variants by making extensive use of OpenType feature tags, such as by embedding small caps into the roman fonts, implementing such a step would imply an important conceptual change and might cause unforeseen problems.

Incorporating multiple design sizes into a single font might have similar implications, but the effects might be less critical if they are limited to the well-controlled environment of math typesetting.

In the \TeX Gyre fonts the situation is somewhat simpler, because these fonts are currently limited to the basic roman and italic fonts and do not have small caps variants or optical sizes.

Incorporating a potential addition of small caps in \TeX Gyre fonts by means of OpenType feature tags might well be possible without causing any incompatible changes. Similarly, incorporating some expanded design variants with adjusted proportions for use in script sizes would also be conceivable when designing \TeX Gyre math fonts.

5.3 The impact of OpenType math

As we have seen in the previous sections, OpenType math fonts provide a way of embedding all the relevant font-specific and glyph-specific information needed for high-quality math typesetting.

In many aspects, the concepts of OpenType math are very similar to \TeX or go beyond \TeX . However, the implementation of these concepts in OpenType fonts will be different in most cases.

Given the adoption of OpenType math as a *de facto* standard and its likelihood of becoming an official standard eventually, OpenType math seems to be the best choice for future developments of new math fonts for use with new \TeX engines.

While $X_{\text{f}}\TeX$ has already started to support OpenType math and Lua \TeX is very likely to follow, adopting OpenType for the design of math fonts for Latin Modern or \TeX Gyre will take more time and will require developing a deeper understanding of the concepts and data structures.

Most importantly, however, it will also require rethinking many traditional assumptions about the way fonts are organized.

Thus, while the topic of font encodings of math fonts may ultimately become a non-issue, the topic of font technology will certainly remain important.

5.4 The challenges of OpenType math

Developing a math font has never been an easy job, so attempting to develop a full-featured OpenType math font for Latin Modern or \TeX Gyre certainly presents a major challenge to font designers or font implementors for a number reasons.

First, such a math font will be really large, even in comparison with text fonts, which already cover a large range of Unicode.² It will have to extend across multiple 16-bit planes to account for the slots of the math alphabets starting at U+1D400, and it will also require a considerable number of unencoded glyphs to account for the size variants of extensible glyphs and the optical variants of math alphabets.

Besides the size of the font, such a project will also present many technical challenges in dealing with the technology of OpenType math fonts.

While setting up the font-specific parameters of the OpenType MATH table is comparable to setting up the `\fontdimen` parameters of \TeX 's math fonts, setting up the glyph-specific information will require detailed attention to each glyph as well as extensive

testing and fine-tuning to achieve optimal placement of math accents and indices.

Finally, there will be the question of assembling the many diverse elements that have to be integrated in a comprehensive OpenType math font. So far, the various styles of math alphabets and the various optical sizes of these alphabets have been designed as individual fonts, but in OpenType all of them have to be combined in a single font. Moreover, the optical sizes will have to be set up as substitutions triggered by OpenType feature tags.

6 Summary and conclusions

In this paper we have reviewed the work on math font encodings since 1990 and the current situation of math fonts as of 2008, especially in view of recent developments in Unicode and OpenType font technology. In particular, we have looked in detail at the features of OpenType math in comparison to the well-known features of \TeX 's math fonts.

While OpenType math font technology looks very promising and seems to be the best choice for future developments of math fonts, it also presents many challenges that will have to be met.

While support for OpenType math in new \TeX engines has already started to appear, the development of math fonts for Latin Modern or \TeX Gyre using this font technology will not be easy and will take considerable time.

In the past, the \TeX conference in Cork in 1990 was the starting point for major developments in text fonts, which have ultimately led to the adoption of Unicode and OpenType font technology.

Hopefully, the \TeX conference at Cork in 2008 might become the starting point for major developments of math fonts in a similar way, except that this time there will be no more need for a new encoding that could be named after the site of the conference.

Acknowledgements

The author wishes to acknowledge feedback, suggestions, and corrections from some of the developers of projects discussed in this review.

A preprint of this paper has been circulated on the Unicode math mailing list hosted at Google Groups [38] and future discussions on the topics of this paper are invited to be directed here.

References

- [1] Yannis Haralambous: \TeX and Latin alphabet languages. *TUGboat*, 10(3):342–345, 1989.
<http://www.tug.org/TUGboat/Articles/tb10-3/tb25hara-latin.pdf>

² In the example of the Cambria Math font, the math font is reported to have more than 2900 glyphs compared to nearly 1000 glyphs in the Cambria text font.

- [2] Nelson Beebe: Character set encoding. *TUGboat*, 11(2):171–175, 1990.
<http://www.tug.org/TUGboat/Articles/tb11-2/tb28beebe.pdf>
- [3] Janusz S. Bień: On standards for CM font extensions. *TUGboat*, 11(2):175–183, 1990.
<http://www.tug.org/TUGboat/Articles/tb11-2/tb28bien.pdf>
- [4] Michael Ferguson: Report on multilingual activities. *TUGboat*, 11(4):514–516, 1990.
<http://www.tug.org/TUGboat/Articles/tb11-4/tb30ferguson.pdf>
- [5] Frank Mittelbach, Robin Fairbairns, Werner Lemberg: \LaTeX font encodings, 2006.
<http://www.ctan.org/tex-archive/macros/latex/doc/encguide.pdf>
- [6] Jörg Knappen: The release 1.2 of the Cork encoded DC fonts and text companion fonts. *TUGboat*, 16(4):381–387, 1995. Reprint from the Proceedings of the 9th European \TeX Conference 1995, Arnhem, The Netherlands.
<http://www.tug.org/TUGboat/Articles/tb16-4/tb49knap.pdf>
- [7] Berthold K. P. Horn: The European Modern fonts. *TUGboat*, 19(1):62–63, 1998
<http://www.tug.org/TUGboat/Articles/tb19-1/tb58horn.pdf>
- [8] Bogusław Jackowski, Janusz M. Nowacki: Latin Modern: Enhancing Computer Modern with accents, accents, accents. *TUGboat*, 24(1):64–74, 2003. Proceedings of the TUG 2003 Conference, Hawaii, USA.
<http://www.tug.org/TUGboat/Articles/tb24-1/jackowski.pdf>
- [9] Bogusław Jackowski, Janusz M. Nowacki: Latin Modern: How less means more. *TUGboat*, 27(0):171–178, 2006 Proceedings of the 15th European \TeX Conference 2005, Pont-à-Mousson, France.
<http://www.tug.org/TUGboat/Articles/tb27-0/jackowski.pdf>
- [10] Will Robertson: An exploration of the Latin Modern fonts. *TUGboat*, 28(2):177–180, 2007.
<http://www.tug.org/TUGboat/Articles/tb28-2/tb89robertson.pdf>
- [11] Hans Hagen, Jerzy B. Ludwiczowski, Volker RW Schaa: The new font project: \TeX Gyre. *TUGboat*, 27(2):250–253, 2006. Proceedings of the TUG 2006 Conference, Marrakesh, Morocco.
<http://www.tug.org/TUGboat/Articles/tb27-2/tb87hagen-gyre.pdf>
- [12] Jerzy B. Ludwiczowski, Bogusław Jackowski, Janusz M. Nowacki: Five years after: Report on international \TeX font projects. *TUGboat*, 29(1):25–26, 2008. Proceedings of the 17th European \TeX Conference 2007, Bachotek, Poland.
<http://www.tug.org/TUGboat/Articles/tb29-1/tb91ludwiczowski-fonts.pdf>
- [13] Alan Jeffrey: Math font encodings: A workshop summary. *TUGboat*, 14(3):293–295, 1993. Proceedings of the TUG 1993 Conference, Aston University, Birmingham, UK.
<http://www.tug.org/TUGboat/Articles/tb14-3/tb40mathenc.pdf>
- [14] Justin Ziegler: Technical report on math font encodings. \LaTeX 3 Project Report, 1993.
<http://www.ctan.org/tex-archive/info/1tx3pub/processed/13d007.pdf>
- [15] Math Font Group (MFG) web site, archives, papers, and mailing list.
<http://www.tug.org/twg/mfg/>
<http://www.tug.org/twg/mfg/archive/>
<http://www.tug.org/twg/mfg/papers/>
<http://www.tug.org/mailman/listinfo/math-font-discuss>
- [16] Matthias Clasen, Ulrik Vieth: Towards a new Math Font Encoding for (\LaTeX) \TeX . *Cahiers GUTenberg*, 28–29:94–121, 1998. Proceedings of the 10th European \TeX Conference 1998, St. Malo, France.
<http://www.gutenberg.eu.org/pub/GUTenberg/publicationsPDF/28-29-clasen.pdf>
- [17] Ulrik Vieth *et al.*: Summary of math font-related activities at Euro \TeX 1998. *MAPS*, 20:243–246, 1998.
<http://www.ntg.nl/maps/20/36.pdf>
- [18] Ulrik Vieth: What is the status of new math font encodings? Posting to mailing list, 2007.
<http://www.tug.org/pipermail/math-font-discuss/2007-May/000068.html>
- [19] Barbara Beeton, Asmus Freytag, Murray Sargent III: Unicode Support for Mathematics. Unicode Technical Report UTR#25. 2001.
<http://www.unicode.org/reports/tr25/>
- [20] Barbara Beeton: Unicode and math, a combination whose time has come — Finally! *TUGboat*, 21(3):174–185, 2000. Proceedings of the TUG 2000 Conference, Oxford, UK.
<http://www.tug.org/TUGboat/Articles/tb21-3/tb68beet.pdf>
- [21] Barbara Beeton: The STIX Project — From Unicode to fonts. *TUGboat*, 28(3):299–304, 2007. Proceedings of the TUG 2007 Conference,

- San Diego, California, USA.
<http://www.tug.org/TUGboat/Articles/tb28-3/tb90beet.pdf>
- [22] STIX Fonts Project: Web Site and Frequently Asked Questions.
<http://www.stixfonts.org/>
<http://www.stixfonts.org/STIXfaq.html>
- [23] Murray Sargent III: Math in Office Blog.
<http://blogs.msdn.com/murrays/default.aspx>
- [24] Murray Sargent III: High-quality editing and display of mathematical text in Office 2007.
<http://blogs.msdn.com/murrays/archive/2006/09/13/752206.aspx>
- [25] Tiro Typeworks: Cambria Math Specimen.
http://www.tiro.nu/Articles/Cambria/Cambria_Math_Basic_Spec_V1.pdf
- [26] John Hudson, Ross Mills: Mathematical Typesetting: Mathematical and scientific typesetting solutions from Microsoft. Promotional Booklet, Microsoft, 2006.
<http://www.tiro.com/projects/>
- [27] Daniel Rhatigan: Three typefaces for mathematics. The development of Times 4-line Mathematics, AMS Euler, and Cambria Math. Dissertation for the MA in typeface design, University of Reading, 2007.
http://www.typeculture.com/academic_resource/articles_essays/pdfs/tc_article_47.pdf
- [28] Murray Sargent III: Unicode Nearly Plain Text Encodings of Mathematics. Unicode Technical Note UTN#28, 2006.
<http://www.unicode.org/notes/tn28/>
- [29] Microsoft Typography: OpenType specification version 1.5.
<http://www.microsoft.com/typography/otspec/>
- [30] George Williams: FontForge. Math typesetting information.
<http://fontforge.sourceforge.net/math.html>
- [31] Apostolos Syropoulos: Asana Math.
www.ctan.org/tex-archive/fonts/Asana-Math/
- [32] Will Robertson: Experimental Unicode math typesetting: The unicode-math package.
<http://scripts.sil.org/svn-public/xetex/TRUNK/texmf/source/xelatex/unicode-math/unicode-math.pdf>
- [33] Yannis Haralambous: Fonts and Encodings. O'Reilly Media, 2007. ISBN 0-596-10242-9
<http://oreilly.com/catalog/9780596102425/>
- [34] Bogusław Jackowski: Appendix G Illuminated. *TUGboat*, 27(1):83–90, 2006. Proceedings of the 16th European T_EX Conference 2006, Debrecen, Hungary.
<http://www.tug.org/TUGboat/Articles/tb27-1/tb86jackowski.pdf>
- [35] Ulrik Vieth: Understanding the aesthetics of math typesetting. *Biuletyn GUST*, 5–12, 2008. Proceedings of the 16th BachoT_EX Conference 2008, Bachotek, Poland.
<http://www.gust.org.pl/projects/e-foundry/math-support/vieth2008.pdf>
- [36] Ulrik Vieth: Math Typesetting in T_EX: The Good, the Bad, the Ugly. *MAPS*, 26:207–216, 2001. Proceedings of the 12th European T_EX Conference 2001, Kerkrade, Netherlands.
<http://www.ntg.nl/maps/26/27.pdf>
- [37] Unicode Consortium: Code Charts for Symbols and Punctuation.
<http://www.unicode.org/charts/symbols.html>
- [38] Google Groups: Unicode math for T_EX.
<http://groups.google.com/group/unimath>

Meta-designing parameterized Arabic fonts for AlQalam

Ameer M. Sherif, Hossam A. H. Fahmy

Electronics and Communications Department

Faculty of Engineering, Cairo University, Egypt

ameer dot sherif (at) gmail dot com, hfahmy (at) arith dot stanford dot edu

<http://arith.stanford.edu/~hfahmy>

Abstract

In this paper we discuss how parameterized Arabic letters are meta-designed using METAFONT and then used to form words. Parameterized Arabic fonts enable greater flexibility in joining glyphs together and rendering words with imperceptible junctions and smoother letter extensions. This work aims to produce written Arabic with quality close to that of calligraphers. Words produced using our parameterized font are compared to other widely used fonts in a subjective test and results are presented.

1 Introduction

The Arabic script is used for a multitude of languages and is the second most widely used script in the world. However, due to the inherent complexity [3, 6] of producing high quality fonts and typesetting engines, the support for Arabic digital typography has been very weak.

OpenType is currently the de facto standard font technology. It has many features to support a wide variety of scripts, yet has its limitations for Arabic [7]. The most significant limitations are probably the following two.

1. The concept of letter boxes connecting together via other boxes of elongation strokes is not suitable for highest quality Arabic typesetting. When connecting glyphs to one another, the junctions rarely fit perfectly because adjacent letter glyphs usually have different stroke directions at the starting and ending points.
2. The use of pre-stored glyphs for different ligatures is limiting. The number of possible ligatures is far greater than what can be made available.

In order to achieve an output quality close to that of Arabic calligraphers, we modeled [7] the pen nib and its movement to draw curves using METAFONT. In this paper, we use the pen stroke macros that we have defined to meta-design the primitive glyphs needed for a good quality Arabic font. So far, we are working with the Naskh writing style and we provide a fully dynamic and flexible design leading to smooth junctions between letters. We also developed a simple algorithm to perform kerning in the case of letters that do not connect to what

follows them. According to a survey we conducted, our design surpasses the widely used fonts.

Our work is not yet finished. In the future, we need to provide for the automatic placement of dots and diacritic marks and complete the rest of the required shapes.

2 Strokes in Arabic glyphs

The Arabic alphabet, although consisting of 28 different letters, depends on only 17 different skeletons. The dots added above or below some of these skeletons are the means of differentiating one letter from another. For example the letters *ḡīm* (ج) and *ḥā'* (ح) have the same shape as the letter *ḥā'* (ح), but *ḡīm* has a dot below, and *ḥā'* has a dot above. When we discuss a primitive we mention its use in the group of letters having the same skeleton, not individual letters, and this further simplifies our designs.

2.1 The Arabic measurement unit

Over a thousand years ago, Ibn-Muqlah, one of the early theorists of Arabic calligraphy, was probably the first to make the choice of the *nuqtā* (Arabic for dot) as a measurement unit for letter forms [3]. He chose it in order to have some fixed measurements between different letter forms. For example, in the Naskh writing style, the height of *'alif* is 4 *nuqtās*, and the width of an isolated *nūn* is 3 *nuqtās*. The *nuqtā* or dot we refer to is that made by the pen used to write the letter, i.e., it is not a constant like the *pt*. The horizontal width of the *nuqtā* in Naskh (where the pen is held at an inclination of 70 degrees to the horizontal) is approximately equal to the diagonal of a dot drawn by the pen as shown in Fig. 1. Since the dot is a square then the *nuqtā* width is slightly less

than the pen width multiplied by the square root of 2. In our work we take it as $1.4 \times \text{pen width}$ and in our METAFONT programs it is simply abbreviated as *n*.

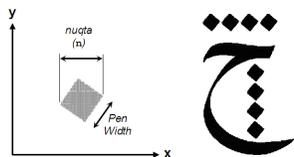


Figure 1: The *nuqta* as a measurement unit.

2.2 Stroke point selection

We define a calligrapher’s pen stroke as a continuous movement of the pen. The location where the calligrapher pauses defines the end of a stroke and the start of a new one. Thus, a circular path may be considered as only one stroke because the start and end points are defined by the movement of the hand and not by the appearance.

The first step in the process of meta-designing any primitive or letter is to select the points through which the pen strokes pass. This is not an easy choice. When designing outline fonts, the solution is usually to scan a handwritten letterform, digitize its outline, then make the necessary modifications. We did not adopt this approach because Arabic letters do not have fixed forms but rather depend on the calligrapher’s style. Since we are *meta*-designing, we are more concerned with how the letter is drawn and not just a single resulting shape. Hence, instead of capturing the fine details of a specific instance of the letter by one calligrapher, we wanted to capture the general features of the letter. To help us accomplish this, we based our design on the works of multiple calligraphers.

The letter *’alif* is shown in Fig. 2 with three different possibilities of point selection. The leftmost glyph requires the explicit specification of the tangential angles at points 1 and 2. In the middle glyph, just connecting the points 1–3–4–2 with a Bézier curve can produce the same curve without explicitly specifying any angles: $\mathbf{z1}.. \mathbf{z3}.. \mathbf{z4}.. \mathbf{z2}$.

Theoretically, we can specify the path using an infinite number of points, but the fewer the points, the better the design and the easier to parameterize it. Adding more points that also lie on the same path can be done as in the rightmost glyph, but point 5 is redundant because the stroke is symmetric, and can be produced without explicitly specifying any angles or tension.

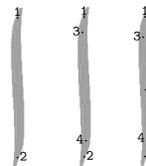


Figure 2: Selecting points to define the path of the letter *’alif*.

This *’alif* example shows that the minimum number of points to choose for any stroke is two, and their locations are at the endpoints of the stroke. These are the easiest points to select. Intermediate points are then chosen when curve parameters such as starting and ending directions and tensions are not enough to define the curve as needed for capturing important letter features. Hence more points are usually needed in stroke segments with sharp bends or in asymmetric strokes.

2.3 Stroke point dependencies

In our design, we model the direction of the stroke as it is drawn by the calligrapher, i.e., the stroke of the letter *’alif* is drawn from top to bottom. The points in our designs are numbered in order according to the pen direction. So for the letter *’alif*, the stroke begins at point 1 and ends at point 2.

However, a calligrapher chooses the starting point of the *’alif* stroke depending on the location of the base line. This means that point 1 is chosen relative to point 2, so we define 1 based on 2. Since METAFONT is a declarative language, not an imperative one, the two statements: $\mathbf{z1} = \mathbf{z2} + 3$; and $\mathbf{z2} = \mathbf{z1} - 3$; evaluate exactly the same. Yet we try to make the dependencies propagate in the natural logical order, which then makes editing the METAFONT glyph code an easier job; hence, the first expression is the better choice.

3 Meta-designing Arabic letters

Several characteristics of the letter shapes discovered during our design process were not mentioned explicitly in most calligraphy books. Calligraphers do not measure their strokes with precise rulers and their descriptions are only approximate. Detailed features of the letters are embedded implicitly in their curves as they learned them by practice. However, in our design, we represent the stroke mathematically and require accurate descriptions. The following sections show a couple of examples.

We start by studying the letter shapes and noting the fine variations that might exist between ‘similar’ shapes. Then we select the stroke points, decide

on the pen direction at each point based on the letter shape and stroke thickness, and finally draw the strokes using our `qstroke` macro [7].

3.1 The concept of primitives

Meta-design enables us to break the forms of glyphs into smaller parts, which are referred to as primitives. These primitives may be whole letters or just parts of letters that exist exactly as they are or with small modifications in other letters. We can save design time and increase meta-ness by reusing primitives.

In Knuth's work on his Computer Modern (CM) fonts [4], primitives were not explicitly defined as black boxes and then reused. The use of primitives requires more parameterization than what CM deals with. There, the use of primitives was worthy only in small and limited flexibility shapes such as serifs and arcs, for which Knuth wrote subroutines. The difference between his work and ours is that he parameterized letters to get a large variety of fonts, while we parameterize primitives to make the letters more flexible and better connected, not to produce different fonts. Indeed, as mentioned earlier, our current focus is the Naskh writing style only.

Our classification of Arabic primitives consists of three categories:

- **Type-1 primitives** are used in many letters without any modifications.
- **Type-2 primitives** are dynamic and change shape slightly in different letters.
- **Type-3 primitives** are also dynamic but much more flexible.

3.2 Type-1 primitives

This category includes diacritics and pen strokes common in many letters. The *nuqtā*, *kāf's shāra*, and the *hamza* are Type-1 primitives. Fig. 3 shows the *shāra* of the letter *kāf* and the *hamza*. These glyphs are drawn using a pen with half the width of the regular pen. Other diacritics like the short vowels *fatha* and *kasra* are dynamic, and do change length and inclination angle.



Figure 3: The *shāra* of *kāf* and the *hamza*.

The 'tail' primitive is another example of a Type-1 primitive. It is used as the ending tail in letters like *wāw*, *rā'*, and *zāy* in both their isolated and ending forms. Fig. 4 shows the tail designed using METAFONT on the left and its use in *wāw* and *rā'* on the right.



Figure 4: The tail primitive.

Even in cases with kerning where the tail may collide with deep letters that follow it, many calligraphers raise the letter as a whole without modifying the tail's shape. Fig. 5 shows an example of kerning applied to letters with tails. We follow the same approach in our design and the tail requires no flexibility parameters.



Figure 5: Four consecutive tails in a word as written in the Qur'an [1], [26:148]. Notice the identical tails despite the different vertical positioning.

Listing 1 shows the code for the tail. The natural direction of drawing is from point 3 to 4 to 5. But in fact, the stroke is only between 3 and 4; the last part of the tail is called a *shāzzya* and calligraphers usually outline it using the tip of the pen nib then fill it in. We use the METAFONT `filldraw` macro for that purpose. We use `filldraw` instead of `fill` in order to give thickness to the *shāzzya* edge at point 5. Also note the coordinate points dependencies: `z4` depends on `z3` and `z5` depends on `z4` not `z3`. This makes modification of the glyph much easier by separating the definition of the stroke segment and that of the *shāzzya*, i.e., if we modify the stroke, the *shāzzya* is not affected, unlike if `z5` was a function of `z3`.

```

z4 = z3 + (-1.7n, -2n);
z5 = z4 + (-2n, .36n);
path raa_body;
raa_body = z3{dir -95} .. tension 1.3 ..
z4{dir -160};
qstroke(raa_body, 85, 100, 0, 0);
path shathya;
shathya = (x4, top y4){dir -160} ..
{dir 160}z5{dir -38} .. tension 1.3 ..
(rt bot z4){dir 15} -- cycle;
pickup pencircle scaled 1.2;
filldraw shathya;

```

Listing 1: METAFONT code for the tail primitive.

The code first defines the points in relation to each other using the *nuqtā* (`n`) as the unit of measure-

ment. Then a path variable is created which holds the path definition of the stroked part of the tail. The `qstroke` macro [7] is used to draw the stroke. The second path defines the *shāẓya* outline.

3.3 Type-2 primitives

This class of primitives has few parameters that enable only slight variations in the primitive’s shape to facilitate its use in different letters. We will discuss two primitives of this type: the ‘*wāw* head’ and the ‘*alif* stem’.

3.3.1 The *wāw* head primitive

This primitive is a circular glyph used in the starting and isolated forms of the letters *wāw*, *fā’*, and *qāf*. It consists of two parts: the head and the neck. In most calligraphy books, the head is described as being exactly the same in all three letters. However small differences exist between the heads due to the connections with different letter skeletons. Fig. 6 shows the letters *fā’* and *qāf* as drawn in three books. Note how the circular head does in fact look slightly different in both letters, yet none of these books mention that there are variations in the head.

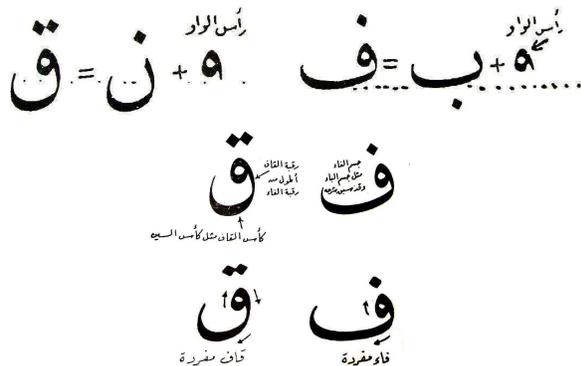


Figure 6: The letters *fā’* and *qāf* as drawn by three calligraphers, from top to bottom: Afify [2], Mahmoud [5], and Zayed [8].

Fig. 7 shows that the *wāw* head primitive consists of 2 strokes, one between points 1–2, the other between points 2–3–4. We approximate the differences between the *wāw*, *fā’*, and *qāf* by altering the 3–4 segment. Thus the same primitive may be used for the three letters in their isolated form. This same primitive is used in their ending forms by moving point 1 down and to the right, to connect to a preceding letter or *kashīda*.

3.3.2 The ‘*alif* stem primitive

The stem of the ‘*alif*’ (Fig. 2) is used in many letters: *lām* (all forms), *kāf* (isolated and final forms), *mām*

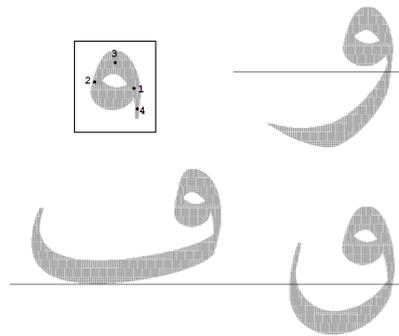


Figure 7: The *wāw* head primitive.

(final form), and *tā’* (all forms). Most calligraphers describe the straight stroke in the *lām*, *kāf*, and *tā’* as being identical to the ‘*alif*’. Fig. 8 shows a calligrapher’s description [5] stating that the form of the vertical stroke in the different letters is exactly the same as the ‘*alif*’. This is a crude approximation because there are differences in the thickness, curvature, inclination, and height (in case of the *tā’*) between the isolated ‘*alif*’ and the modified form used in other letters.

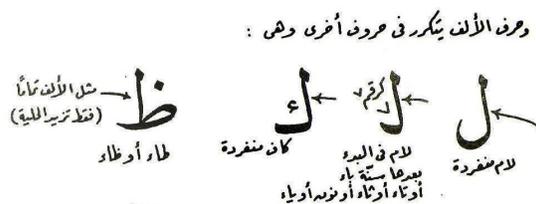


Figure 8: Approximate directions in calligraphy books.

Fig. 9 shows our design: on the far right the isolated ‘*alif*’ and to its left the modified ‘*alif*’ that is used in *lām* and *kāf*. The modified ‘*alif*’ is thinner with less curvature near the middle, or in other words more tension, together with more overall inclination. Listing 2 shows that they both have the same height and the thickness of the stem is achieved by increasing the pen nib angle.

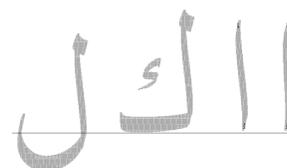


Figure 9: The ‘*alif*’ primitive.

```

% Description for isolated 'alif
curve := -100; incline := 70; height := 4.5n;
z2 = z1 + (0, -height);
path saaq;
saaq = z1{dir curve} .. tension 1.4 ..
      z2{dir curve};
qstroke(saaq, incline, incline, 0, 0);

% Description for 'alif used in lam and kaaf
curve := -95; incline := 79; height := 4.5n;
z2 = z1 + (0.3n, -height);
path saaq;
saaq = z1{dir curve} .. tension 1.4 ..
      z2{dir curve};
qstroke(saaq, incline, incline - 3, 0, 0);

```

Listing 2: METAFONT code for the 'alif primitive.

3.4 Type-3 primitives

Type-3 primitives are glyphs that have a wider dynamic range, and greater flexibility. In this section, we discuss the skeleton of the letter *nūn*, called the *kasa* (Arabic for cup) and the *kashīda*. Calligraphers often use the great flexibility of these primitives to justify lines.

3.4.1 Kasa primitive

The body of the letter *nūn* is used in the isolated and ending forms of *sīn*, *ṣīn*, *ṣād*, *ḍād*, *lām*, *qāf*, and *yā'*. Fig. 10 shows the *kasa* in five letters. The *kasa* has two forms, short and extended. The short form is almost 3 *nuqtās* in width in the case of *nūn*, one *nuqtā* longer in *yā'*, and slightly shorter in *lām*. This difference between the *kasa* of the *lām* and the *nūn* is not well documented in calligraphy books, where most calligraphers mention that both are the same and only few state that in the *lām* it is slightly smaller.

An important property of the *kasa* is that it can be extended to much larger widths. In its extended form, it can range from 9–13 *nuqtās*. Fig. 11 shows the short form together with three instances of the longer form generated from the same METAFONT code. Note that its width can take any value between 9 and 13, not just integer values, depending on line justification requirements. Also note how the starting *senn* (vertical stroke to the right) of the letter is shorter in extended forms.

3.4.2 Kashida primitive

Another very important primitive for justification, the *kashīda* is used in almost all connected letters. As an illustrative example, Fig. 12 shows the letter *ḥā'* in its initial form with two different *kashīda* lengths,

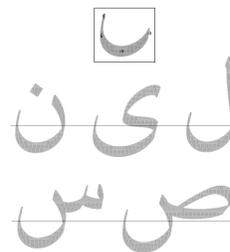


Figure 10: The *kasa* primitive.

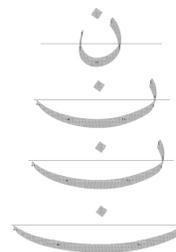


Figure 11: The letter *nūn* shown with *kasa* widths of 3, 9, 10 and 13 *nuqtās*.

differing by 3 *nuqtās*. The parameter *tatwil* controls this length by varying the distance between points 3 and 4, both the horizontal and vertical components, as shown in this line of code:

```
z3 = z4 + (1.74n, 0.116n) + (0.5tatwil, 0.025tatwil)*n;
```

As *tatwil* increases, point 3 moves further from point 4 both to the right and upward. This vertical change helps maintain the curvature in the *kashīda*. If no vertical adjustment is made, longer *kashīdas* look like separate straight lines with a sharp corner at their intersection with the surrounding letters. Calligraphers, on the other hand, draw curved lines rather than straight ones producing aesthetically better shapes. For these reasons, in our definition of the stroke, the tangential direction at point 3 is left free depending on the distance between 3 and 4. We will see in the next section how *kashīdas* are adjusted to join letters together smoothly.

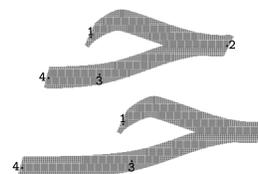


Figure 12: The initial form of the letter *ḥā'* with two different *kashīda* lengths.

4 Forming words

The combination of primitives to form larger entities is the final step towards producing Arabic script that is as cursively connected and flexible as calligraphers' writings. The parameterization of the glyphs allows us to piece them together perfectly as if they were drawn with just one continuous stroke.

4.1 Joining glyphs with *kashīdas*

The most widely used glyph to connect other letters is the *kashīda*. In this section we will explain the mechanism we use in order to make the junction between letters as smooth as possible. In current font standards, such as OpenType and TrueType, *kashīdas* are made into fixed glyphs with pre-defined lengths, and are substituted when needed between letters to give the feeling of extending the letter. But because that design for the *kashīda* is static, as are the rest of the surrounding letters, they rarely join well. It is evident that the word produced is made of different segments joined by merely placing them close to each other.

In our work, the *kashīda* is dynamic and can take continuous values, not just predefined or discrete values. We believe that when a *kashīda* is extended between any two letters, it does not belong to just one of them; instead, it is a connection between them both. This belief is the result of experimenting with different joining methods.

Let us take the problem of joining the two letters *ḥā'* and *dāl* as an example to illustrate the *kashīda* joining mechanism we have developed. The solution we propose is to pass the *tatwīl* parameter to the macros producing the two glyphs, and the *kashīda* length is distributed between both glyphs. This enables us to fix the ends of the glyphs to be joined at one angle, which is along the x-axis in the Naskh style, since any *kashīda* in that style must at one point move in this direction before going up again. To accommodate long *kashīdas*, each glyph ending point is moved further from its letter and slightly downwards. Long *kashīdas* need more vertical space in order to curve smoothly, sometimes pushing the letters of a word upwards.

Other than affecting the ending points, the parameter also affects the curve definition on both sides by varying the tensions, while keeping the direction of the curves at the intersection along the negative x-axis (since the stroke is going from right to left). The resulting word at many different *kashīda* lengths is shown in Fig. 13.



Figure 13: Placing a *kashīda* between the letters *ḥā'* and *dāl* with different lengths: 2, 3, 5 and 7 nuqtas.



Figure 14: The word *Mohammed* as an example of vertical placement (Thuluth writing style).

4.2 Vertical placement of glyphs

In written Arabic, the existence of some letter combinations may force the starting letter of a word to be shifted upwards in order to accommodate for the ending letters to lay on the baseline of the writing. A very simple example of that property is the name *Mohammed* when written with ligatures, where the initial letter *mīm* is written well above the baseline, as shown in Fig. 14.

It is hence obvious that the starting letter's vertical position is dependent on the word as a whole. It might then be thought that it is easier to draw the words starting from the left at the baseline and then move upwards while proceeding to the right. But this has two problems: one is that the horizontal positioning of the last letter depends on the position of the first letter on the right and on the length of the word, and the second is that a left to right drawing would be against the natural direction of writing and may result in an unnatural appearance.

The solution is then to walk through the word till its end and analyze each letter to know where to position the beginning letter vertically, and then start the actual writing at the right from that point going left. This process is what a calligrapher actually does before starting to write a word. So, for a combination of letters, we benefit from the declarative nature of METAFONT. The following rules are applied:

- The horizontal positioning starts from the right,
- the vertical positioning starts from the left at the baseline, and
- writing starts from the right.

To illustrate this better, see Fig. 15. The ligature containing the letters *sīm*, *ǧīm*, and *wāw* is traced from left-to-right as shown, going through points 1–2–3, the starting points of each glyph, until

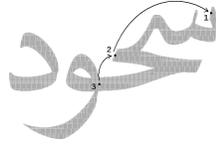


Figure 15: Tracing a word from left-to-right to know the starting vertical position.

أوزان - أوزان
زرع رزاز
أن رن

Figure 16: Kerning: letters with ‘tails’ need special attention. The first line shows our output to the right and the current font technologies to the left. The last two lines show our output.

the vertical position of point 1 is known. The next step is to start writing the word starting from point 1. Isolated letters like *dāl* are not taken into consideration, because they do not affect the preceding letters vertically.

4.3 Dealing with kerning

To imitate what calligraphers do as in Fig. 5, we had to invent a special kerning algorithm. Fig. 16 shows our output. We raise the level of any ‘deep letter’ following a ‘tail letter’ to kern correctly. Any non-deep letter holds the level steady. The end of the word restores the baseline.

4.4 Word lengths

In reality, word lengths are not selected by the calligraphers word by word, but instead, are chosen based on the justification requirements of a whole line. When a word length is decided according to the line it exists in, this length should be passed to a main macro that calls the glyph macros in order to form the word. This main macro should decide the length of *kashīdas* to be added depending on the minimum length of each letter. For example, that the word under consideration is the one shown in Fig. 13, and that the desired total length of the word is 10 *nuqtās*. In order to calculate the extension or the *tatwīl* parameter between the letters, it subtracts all the minimum lengths of the individual letters. In our example, the head of the *ḥā’* is 4 *nuqtās* wide, and the base of the *dāl* is 3 *nuqtās*, hence the word has a minimum length of only 7 *nuqtās*. In order to stretch it to 10, the added *kashīda* is 3 *nuqtās* wide.

4.5 A final example

This section describes a more illustrative example shown in Fig. 17. This example, showing four instances of the word *sujud*, demonstrates the many properties and benefits of our parameterized font. First, it shows flexibility in stretching and compressing words for line justification purposes. This flexibility is due to two capabilities of the font: dynamic length *kashīdas* and glyph substitution. For a very small line spacing, the *sīn* is written on top of the *ḥā’*, and the *kashīda* after the *ḥā’* is almost zero.

When more space is available, the *kashīda* after the *ḥā’* is stretched and the *senn* connecting the *sīn* and the *ḥā’* is also made slightly longer. Further elongation is made possible by breaking the ligature between *sīn* and *ḥā’*. And finally, the maximum length is obtained by elongating the *kashīda* between the two letters. Theoretically, we could get more stretching of this *kashīda* and even add another one after the *ḥā’*, but calligraphic rules ultimately limit the stretching.

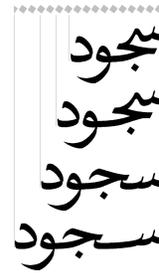


Figure 17: The word *sujud* written with different lengths ranging from 10.5 to 16.5 *nuqtas*.

5 Testing the output

5.1 A GUI to simplify tests

To facilitate the task of testing our ideas, we made a simple graphical user interface (GUI) to write words using our METAFONT programs. This allows us to make changes to parameters and draw words faster than having to edit the code manually. Fig. 18 shows the block diagram describing the operation of the GUI and Fig. 19 shows the different window components of the GUI.

First, the user types in a word or sequence of words in the input word text box, then presses the start button. This parses the input word(s) into a string of letters, removing any space characters. Each character can be selected from a list and its shape determined from the letterform list at the bottom of the screen. Also a length extension can be input in the length text box (default is zero).

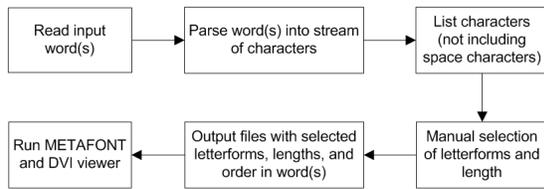


Figure 18: Block diagram describing the operations of the graphical user interface.

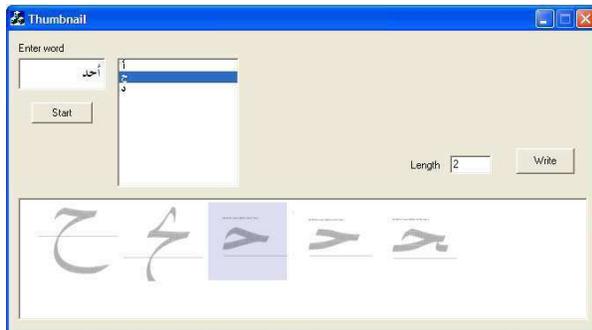


Figure 19: Screenshot of our simple GUI.

When the write button is pressed, the letterforms selected and their extra lengths are written in files, then METAFONT executes the glyph programs. Finally, a DVI previewer opens the resulting output as seen in Fig. 20.



Figure 20: Screenshot of the DVI previewer displaying the output word.

5.2 The survey

In order to test if readers are comfortable with the way our parameterized font looks as compared to other Naskh fonts, we made a list of 16 words, each written in four different Naskh fonts:

- Simplified Arabic,
- Traditional Arabic,
- DecoType Naskh, and
- AlQalam parameterized font.

For a more reliable and unbiased test, the order of fonts used is varied in consecutive rows and all words are set to approximately the same sizes although nominal point sizes of the different fonts are not

exactly equivalent. We selected the 16 words to test three main features:

- connections between letters,
- extension/tatwil of letters, and
- kerning.

Fig. 21 shows the first page of the test.

استبيان				
هذا الاستبيان خاص ببحث عن كتابة الحروف العربية على الحاسب. المطلوب هو تقييم الكلمات التالية بناءً على مدى اوتياحك لها من حيث الشكل. فضلاً ضع علامة على الرقم المناظر لكل كلمة من واحد إلى خمسة حيث أن خمسة تعني أن الكلمة مريحة وواحد تعني أن الكلمة غير مريحة.				
هال	هال	هال	هال	1
(5) (4) (3) (2) (1)	(5) (4) (3) (2) (1)	(5) (4) (3) (2) (1)	(5) (4) (3) (2) (1)	
هدهد	هدهد	هدهد	هدهد	2
(5) (4) (3) (2) (1)	(5) (4) (3) (2) (1)	(5) (4) (3) (2) (1)	(5) (4) (3) (2) (1)	
حال	حال	حال	حال	3
(5) (4) (3) (2) (1)	(5) (4) (3) (2) (1)	(5) (4) (3) (2) (1)	(5) (4) (3) (2) (1)	
هذا	هذا	هذا	هذا	4
(5) (4) (3) (2) (1)	(5) (4) (3) (2) (1)	(5) (4) (3) (2) (1)	(5) (4) (3) (2) (1)	
أوزان	أوزان	أوزان	أوزان	5
(5) (4) (3) (2) (1)	(5) (4) (3) (2) (1)	(5) (4) (3) (2) (1)	(5) (4) (3) (2) (1)	

Figure 21: First page of the test where we ask the readers to rate their comfort.

We then surveyed 29 Arabic readers (14 males and 15 females) with ages ranging from 10 to 70 years. In the survey, readers were asked to evaluate and rate words in terms of their written quality and how comfortable they feel reading the words. A rating of 1 is given to low quality and uncomfortable words, and a rating of 5 is given to words of high written quality and most comfort to the reader.

This test methodology is known as the Mean Opinion Score (MOS), a subjective test often used to evaluate the perceived quality of digital media after compression or transmission in the field of digital signal processing. The MOS is expressed as a single number in the range of 1 to 5, where 1 is lowest quality and 5 is highest quality. In our case, the MOS is a numerical indication of the perceived quality of a written word. Finally, the total MOS is calculated as the arithmetic mean of all the individual scores.

Table 1 shows the MOS scores for individual words of each font and the total average.

Table 1: MOS results for each font.

Line No.	Simplified Arabic	Traditional Arabic	DecoType Naskh	AlQalam Parameterized
1	3.5	2.0	1.7	3.6
2	3.0	3.3	2.7	3.9
3	2.8	2.5	2.4	3.9
4	2.5	3.5	2.2	3.8
5	1.7	2.1	3.4	4.3
6	1.6	2.3	3.6	3.8
7	2.4	2.4	3.9	3.7
8	1.9	1.9	3.6	3.6
9	1.3	2.1	3.9	3.8
10	2.6	3.5	3.1	4.2
11	2.2	1.6	3.6	3.8
12	1.6	2.0	3.4	4.2
13	2.0	2.5	3.3	4.0
14	2.0	1.7	4.0	4.1
15	2.7	3.2	3.1	3.8
16	2.5	2.0	3.8	4.0
MOS	2.3	2.4	3.2	3.9

The results clearly show an increased comfort with the parameterized font with respect to the other popularly used fonts. One feature clearly distinguishing the different fonts is kerning. This is very evident in words 1, 3, 5, 6, 9, 11, 13, and 14 which had kerning applied. The addition of smooth *kashīdas* for extension also results in a large difference in scores as with words 7, 10, and especially 12 which contains a long *kashīda*. The results of word 15 show that the use of complex ligatures is also an important feature for the comfort of readers. We believe that more work and collaboration with calligraphers can yield even better results.

6 Conclusion

The work covered in this paper is just a small step towards the realization of a system to produce output comparable to that created by Arabic calligraphers, and much more work is still needed. We need to complete our design and finish all the required shapes.

There is a strong need for non-engineering research on the readability and legibility of the different kinds of Arabic fonts comparable to the many studies conducted on Latin letters. It is important to categorize the different calligraphers' writing styles as well as the regular computer typefaces: are they easy and fast to read in long texts, or not? Which are better to use in titles or other 'isolated' materials, and which are better for the running text?

By presenting our effort, we hope to open the door for future exciting work from many researchers.

References

- [1] *The Holy Qur'an*. King Fahd Complex for Printing the Holy Qur'an, Madinah, KSA, 1986.
- [2] Fawzy Salem Afify. *ta'leem al-khatt al-'arabi [Teaching Arabic calligraphy]*. Dar Ussama, Tanta, Egypt, 1998.
- [3] Mohamed Jamal Eddine Benatia, Mohamed Elyaakoubi, and Azzeddine Lazrek. Arabic text justification. *TUGboat*, 27(2), January 2007.
- [4] Donald E. Knuth. *Computer Modern Typefaces*, volume E of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986.
- [5] Mahdy Elsayyed Mahmoud. *al-khatt al-'arabi, dirasah tafsiliya muwassa'a [Arabic calligraphy, a broad detailed study]*. Maktabat al-Qur'an, Cairo, Egypt, 1995.
- [6] Thomas Milo. Arabic script and typography: A brief historical overview. In John D. Berry, editor, *Language Culture Type: International Type Design in the Age of Unicode*, pages 112–127. Graphis, November 2002.
- [7] Ameer M. Sherif and Hossam A. H. Fahmy. Parameterized Arabic font development for AlQalam. *TUGboat*, 29(1), January 2008.
- [8] Ahmad Sabry Zayed. *ahdath al-turuq leta'leem al-khotot al-'arabiya [New methods for learning Arabic calligraphy]*. Maktabat ibn-Sina, Cairo, Egypt, 1990.

Smart ways of drawing PSTricks figures

Manjusha Susheel Joshi

Bhaskaracharya Institute of Mathematics

Law College Road,

Pune 411004, India

manjusha dot joshi at gmail dot com

www dot bprim dot org

Abstract

We present a method of using PSTricks in conjunction with free software packages for interactive drawing (Dr. Geo, Dia, Gnuplot, Xfig) to produce various types of figures, via exporting as TEX and modifying the result.

1 Drawing geometric figures

In our institute, we arrange various training programmes for International Maths Olympiad competition, publish the journal *Bona Mathematica*, and produce other mathematical study materials. For all these we need to draw complex geometric figures.

1.1 Accuracy requirements

All the mathematical material is in $\text{L}\text{A}\text{T}\text{E}\text{X}$ format. Faculties require `.eps` files to include in their question papers, notes, books and articles.

In figures, the text used for labels, etc., must all be in the same font. The material is widely distributed among students and others, so from the printing point of view, the figures should be very accurate.

We want to produce such exact figures while not spending excessive amounts of time on drawing them. Also, these figures must be usable in the $\text{L}\text{A}\text{T}\text{E}\text{X}$ sources.

We want to minimize issues such as license, cost, and support and therefore maximize availability of the software. In general, use of free software is encouraged in the institute.

For drawing geometric figures, the free software package Dr. Geo (home page <http://www.gnu.org/software/dr-geo>) is well-suited for us. Figure 1 shows a typical session in progress. Other free software GUI drawing programs include Dia for flowcharts, among other purposes (<http://www.gnome.org/projects/dia>), Gnuplot for function plotting (<http://www.gnuplot.info>), and Xfig for general drawing (<http://www.xfig.org>).

1.2 Convert figures directly to PostScript?

If a figure file is generated by another software program, converted to `ps` or `eps` by an external utility, and then inserted in the $\text{L}\text{A}\text{T}\text{E}\text{X}$ file, we have observed that the font type and size of labels in the converted

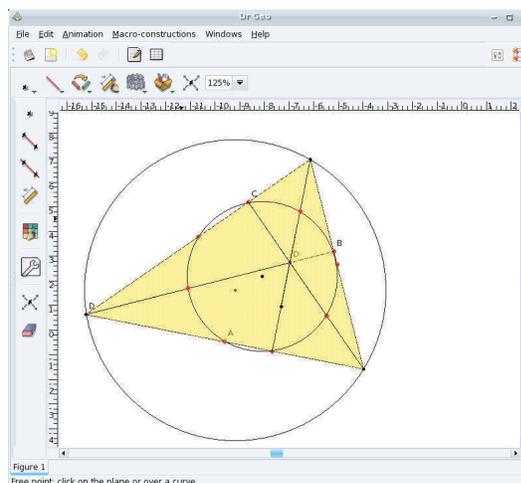


Figure 1: Drawing a figure in Dr. Geo

file is different from that in the $\text{L}\text{A}\text{T}\text{E}\text{X}$ file, and the sharpness of the figure may not be good.

Of course we wish to avoid such problems.

1.3 Export figures as TEX

As an alternative to PostScript export, software such as the above supports exporting figures in TEX format. Advantages of this approach include:

- The ‘Export as TEX file’ options generate the entire code for users.
- The resulting `fig.tex` file has PSTricks code for the figure, which is editable.
- Users may not know every PSTricks command, but they can easily modify the code in useful ways knowing only a small set of commands.
- Users need not calculate (x, y) co-ordinates, as they are already generated according to the figure. One important thing not to worry about.

1.4 Step by step

Here is our process, step by step.

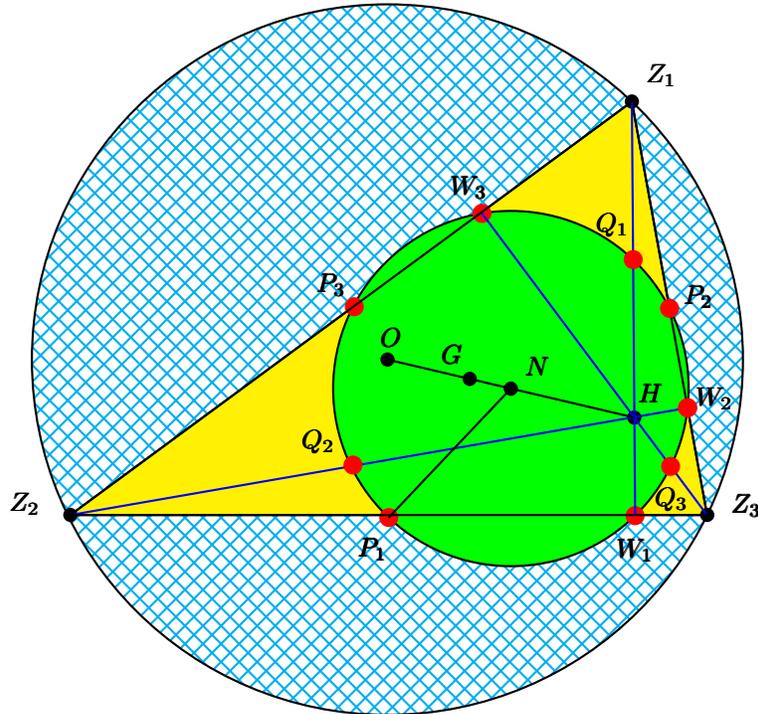


Figure 2: Nine point circle, drawn with modified PSTricks code

- Open a graphical tool for drawing geometric figures, e.g., one of those mentioned above. Draw the figure.
- Label the objects at appropriate places, *as near by as possible*. Save.
- Export as a \TeX figure.
- Open it in any editor or (\LaTeX) IDE to edit. We often use Kile (kile.sourceforge.net).
- Change all the labels in the figure to be in math mode, i.e., enclosed in $\$ \dots \$$, especially labels of vertices.
- If the positions of labels are not perfect, then adjust the x and y coordinates as needed, in the neighbourhood of the x and y which have been calculated by Dr. Geo.

2 Sequence of drawing objects matters

Sometimes two objects will overlap, so that the object drawn earlier becomes invisible. To see an example, refer to Figure 2, a ‘nine point circle’.

Looking at the points of intersection at W_1 and W_3 , we can see the construction clearly, namely, $Z_2Z_3 \perp Z_1W_1$ and $Z_1Z_2 \perp Z_3W_3$. However, at point W_2 , the red dot marker has obscured that $Z_1Z_3 \perp Z_2W_2$. If the dot were drawn before the lines, the lines would be visible, as they are at W_1

and W_3 . (We intentionally left this discrepancy in the figure for expository purposes.)

Drawing sequence is an issue with opaque colors. With transparent colors, the sequence typically does not matter.

3 Example

Figure 2 also serves as a general example showing the success of our workflow. In the figure, nine red points can be seen. These points are constructed using Dr. Geo, i.e., Dr. Geo was responsible for correctly finding these nine intersecting points.

With PSTricks, one can use different colours and textures to highlight points and regions, and place labels at the appropriate co-ordinates and use the same font as \LaTeX file in which figure will be inserted. In this case, we wanted to use the figure on the front page of a journal, and so we needed to make it more colourful.

In a sense, Dr. Geo works here as a front end for PSTricks. With the help of such a front end, users can save considerable time in calculating the co-ordinates for the figure.

To learn about the actual mathematics of the nine point circle, see <http://www.csm.astate.edu/Ninept.html>.

The MetaPost library and LuaTeX

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

Abstract

An introduction to the MetaPost library and its use in LuaTeX.

1 Introduction

If MetaPost support had not been as tightly integrated into ConTeXt as it is, at least half of the projects Pragma ADE has been doing in the last decade could not have been done at all. Take for instance backgrounds behind text or graphic markers alongside text (as seen here). These are probably the most complex mechanisms in ConTeXt: positions are stored, and positional information is passed on to MetaPost, where intersections between the text areas and the running text are converted into graphics that are then positioned in the background of the text. Underlining of text (sometimes used in the educational documents that we typeset) and change bars (in the margins) are implemented using the same mechanism because those are basically a background with only one of the frame sides drawn.

You can probably imagine that a 300 page document with several such graphics per page takes a while to process. A nice example of such integrated graphics is the LuaTeX reference manual, that has an unique graphic at each page: a stylized image of a revolving moon.



Most of the running time integrating such graphics seemed to be caused by the mechanics of the process: starting the separate MetaPost interpreter and having to deal with a number of temporary files. Therefore our expectations were high with regards to integrating MetaPost more tightly into LuaTeX. Besides the speed gain, it also true that the simpler the process of using such use of graphics becomes, the more modern a TeX runs looks and the less problems new users will have with understanding how all the processes cooperate.

This article will not discuss the application interface of the MPlib library in detail; for that there is the manual. In short, using the embedded MetaPost interpreter in LuaTeX boils down to the following:

- Open an instance using `mplib.new`, either to

process images with a format to be loaded, or to create such a format. This function returns a library object.

- Execute sequences of MetaPost commands, using the object's `execute` method. This returns a result.
- Check if the result is valid and (if it is okay) request the list of objects. Do whatever you want with them, most probably convert them to some output format. You can also request a string representation of a graphic in PostScript format.

There is no need to close the library object. As long as there were no fatal errors, the library recovers well and can stay alive during the entire LuaTeX run.

Support for MPlib depends on a few components: integration, conversion and extensions. This article shows some of the code involved in supporting the library. Let's start with the conversion.

2 Conversion

The result of a MetaPost run traditionally is a PostScript language description of the generated graphic(s). When PDF is needed, that PostScript code has to be converted to the target format. This includes embedded text as well as penshapes used for drawing. Here is an example graphic:

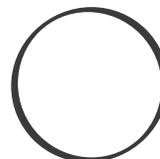


Figure 1

```
draw fullcircle
  scaled 2cm
  withpen pencircle xscaled 1mm yscaled .5mm
  rotated 30 withcolor .75red ;
```

Notice how the pen is not a circle but a rotated ellipse. Later on it will become clear what the consequences of that are for the conversion.

How does this output look in PostScript? In

abridged form, it looks like this:

```

%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: -30 -30 30 30
%%HiResBoundingBox: -29.62 -29.283 29.62 29.283
%%Creator: MetaPost 1.090
%%CreationDate: 2008.09.23:0939
%%Pages: 1
% [preamble omitted]
%%Page: 1 1
0.75 0 0 R 2.55513 hlw rd 1 lj 10 ml
q n 28.34645 0 m
28.34645 7.51828 25.35938
  14.72774 20.04356 20.04356 c
14.72774 25.35938 7.51828
  28.34645 0 28.34645 c
[...]
[0.96077 0.5547 -0.27734 0.4804 0 0] t S Q
P
%%EOF

```

The most prominent code here concerns the path. The numbers in brackets define the transformation matrix for the pen we used. The PDF variant looks as follows:

```

q
0.750 0.000 0.000 rg 0.750 0.000 0.000 RG
10.000000 M
1 j
1 J
2.555120 w
q
  0.960769 0.554701 -0.277351
0.480387 0.000000 0.000000 cm
22.127960 -25.551051 m
  25.516390 -13.813203 26.433849
0.135002 24.679994 13.225878 c
  22.926120 26.316745 18.644486
37.478783 12.775526 44.255644 c
[...]
h S
Q0
g 0 G
Q

```

The operators don't look much different from the PostScript, which is mostly due to the fact that in the PostScript code, the preamble defines shortcuts like `c` for `curveto`. Again, most code involves the path. However, this time the numbers are different and the transformation comes before the path.

In the case of PDF output, we could use TeX itself to do the conversion: a generic converter is implemented in `supp-pdf.tex`, while a converter optimized for ConTeXt MkII is defined in the files whose names start with `meta-pdf`. But in ConTeXt MkIV we use Lua code for the conversion instead. Thanks to Lua's powerful Lpeg parsing library, this gives cleaner code and is also faster. This converter cur-

rently lives in `mllib-pdf.lua`.

Now, with the embedded MetaPost library, conversion goes still differently because now it is possible to request the drawn result and associated information in the form of Lua tables.

```

figure={
  ["boundingbox"]={
    ["llx"]=-29.623992919922,
    ["lly"]=-29.283935546875,
    ["urx"]=29.623992919922,
    ["ury"]=29.283935546875,
  },
  ["objects"]={
    {
      ["color"]={ 0.75, 0, 0 },
      ["linecap"]=1,
      ["linejoin"]=1,
      ["miterlimit"]=10,
      ["path"]={
        {
          ["left_x"]=28.346450805664,
          ["left_y"]=-7.5182800292969,
          ["right_x"]=28.346450805664,
          ["right_y"]=7.5182800292969,
          ["x_coord"]=28.346450805664,
          ["y_coord"]=0,
        },
        ...
      },
      ["pen"]={
        {
          ["left_x"]=2.4548797607422,
          ["left_y"]=1.4173278808594,
          ["right_x"]=-0.70866394042969,
          ["right_y"]=1.2274475097656,
          ["x_coord"]=0,
          ["y_coord"]=0,
        },
        ["type"]="elliptical",
      },
      ["type"]="outline",
    },
  },
}

```

This means that instead of parsing PostScript output, we can operate on a proper datastructure and get code like the following:

```

function convertgraphic(result)
  if result then
    local figures = result.fig
    if figures then
      for fig in ipairs(figures) do
        local llx, lly, urx, ury
          = unpack(fig.boundingBox())
        if urx > llx then
          startgraphic(llx, lly, urx, ury)
          for object in ipairs(fig.objects()) do

```

```

    if object.type == "..." then
        ...
        flushgraphic(...)
        ...
    else
        ...
    end
end
finishgraphic()
end
end
end
end
end
end
end
end

```

Here `result` is what the library returns when one or more graphics are processed. As you can deduce from this snippet, a result can contain multiple figures. Each figure corresponds with a `beginfig ... endfig`. The graphic operators that the converter generates (so-called PDF literals) have to be encapsulated in a proper box so this is why we have:

- `startgraphic`: start packaging the graphic
- `flushgraphic`: pipe literals to \TeX
- `finishgraphic`: finish packaging the graphic

It does not matter what number `beginfig` was passed, the graphics come out in the natural order.

A bit more than half a dozen different object types are supported. The example MetaPost `draw` command above results in an `outline` object. This object contains not only path information but also carries rendering data, like the color and the pen. So, in the end we will flush code like `1 M` which sets the `miterlimit` to 1, or `.5 g` which sets the color to 50% gray, in addition to a path.

Because objects are returned in a way that closely resembles MetaPost's internals, some extra work needs to be done in order to calculate paths with elliptical pens. An example of a helper function in somewhat simplified form is shown next:

```

function pen_characteristics(object)
    local p = object.pen[1]
    local wx, wy, width
    if p.right_x == p.x_coord
        and p.left_y == p.y_coord then
        wx = abs(p.left_x - p.x_coord)
        wy = abs(p.right_y - p.y_coord)
    else -- pyth: sqrt(a^2 + b^2)
        wx = pyth(p.left_x - p.x_coord,
                p.right_x - p.x_coord)
        wy = pyth(p.left_y - p.y_coord,
                p.right_y - p.y_coord)
    end
    if wy/coord_range_x(object.path, wx)
        >= wx/coord_range_y(object.path, wy) then
        width = wy
    end
end

```

```

else
    width = wx
end
local sx, sy = p.left_x, p.right_y
local rx, ry = p.left_y, p.right_x
local tx, ty = p.x_coord, p.y_coord
if width ~= 1 then
    if width == 0 then
        sx, sy = 1, 1
    else
        rx, ry, sx, sy = rx/width, ry/width,
            sx/width, sy/width
    end
end
if abs(sx) < eps then sx = eps end
if abs(sy) < eps then sy = eps end
return sx, rx, ry, sy, tx, ty, width
end

```

If `sx` and `sy` are 1, there is no need to transform the path, otherwise a suitable transformation matrix is calculated and returned. The function itself uses a few helpers that make the calculations even more obscure. This kind of code is far from trivial and as already mentioned, these basic algorithms were derived from the MetaPost sources. Even so, these snippets demonstrate that interfacing using Lua does not look that bad.

In the actual MkIV code things look a bit different because it does a bit more and uses optimized code. There you will also find the code dealing with the actual transformation, of which these helpers are just a portion.

If you compare the PostScript and the PDF code you will notice that the paths looks different. This is because the use and application of a transformation matrix in PDF is different from how it is handled in PostScript. In PDF more work is assumed to be done by the PDF generating application. This is why in both the \TeX and the Lua based converters you will find transformation code and the library follows the same pattern. In that respect PDF differs fundamentally from PostScript.

In the \TeX based converter there was the problem of keeping the needed calculations within \TeX 's accuracy, which fortunately permits larger values than MetaPost can produce. This plus the parsing code resulted in a lot of \TeX code which is not that easy to follow. The Lua based parser is more readable, but since it also operates on PostScript code it too is kind of unnatural, but at least there are fewer problems with keeping the calculations sane. The MPlib based converter is definitely the cleanest and least sensitive to future changes in the PostScript output. Does this mean that there is no ugly code left? Alas, as we will see in the next section, dealing

with extensions is still somewhat messy. In practice users will not be bothered with such issues, because writing a converter is a one time job by macro package writers.

3 Extensions

In Metafun, which is the MetaPost format used with ConTeXt, a few extensions are provided, such as:

- cmyk, spot and multitone colors
- including external graphics
- linear and circular shades
- texts converted to outlines
- inserting arbitrary texts

Until now, most of these extensions have been implemented by using specially coded colors and by injecting so-called specials (think of them as comments) into the output. On one of our trips to a TeX conference, we discussed ways to pass information along with paths and eventually we arrived at associating text strings with paths as a simple and efficient solution. As a result, recently MetaPost was extended by `withprescript` and `withpostscript` directives. For those who are unfamiliar with these new features, they are used as follows:

```
draw fullcircle withprescript "hello"
    withpostscript "world" ;
```

In the PostScript output these scripts end up before and after the path, but in the PDF converter they can be overloaded to implement extensions, and that works reasonably well. However, at the moment there cannot be multiple pre- and postscripts associated with a single path inside the MetaPost internals. This means that for the moment, the scripts mechanism is only used for a few of the extensions. Future versions of MPlib may provide more sophisticated methods for carrying information around.

The MkIV conversion mechanism uses scripts for graphic inclusion, shading and text processing but unfortunately cannot use them for more advanced color support.

A nasty complication is that the color spaces in MetaPost don't cast, which means that one cannot assign any color to a color variable: each colorspace has its own type of variable.

```
color    one ; one := (1,1,0)    ; % correct
cmykcolor two ; two := (1,0,0,1) ; % correct
one := two ;                    % error
fill fullcircle scaled 1cm
    withcolor .5[one,two] ; % error
```

In ConTeXt we use constructs like this:

```
\startreusableMPgraphic{test}
  fill fullcircle scaled 1cm
    withcolor \MPcolor{mycolor} ;
```

```
\stopreusableMPgraphic
\reuseMPgraphic{test}
```

Because `withcolor` is clever enough to understand what color type it receives, this is ok, but how about:

```
\startreusableMPgraphic{test}
  color c ;
  c := \MPcolor{mycolor} ;
  fill fullcircle scaled 1cm withcolor c ;
\stopreusableMPgraphic
```

Here the color variable only accepts an RGB color and because in ConTeXt there is mixed color space support combined with automatic colorspace conversions, it doesn't know in advance what type it is going to get. By implementing color spaces other than RGB using special colors (as before) such type mismatches can be avoided.

The two techniques (coding specials in colors and pre/postscripts) cannot be combined because a script is associated with a path and cannot be bound to a variable like `c`. So this again is an argument for using special colors that remap onto CMYK spot or multi-tone colors.

Another area of extensions is text. In previous versions of ConTeXt the text processing was already isolated: text ended up in a separate file and was processed in a separate run. More recent versions of ConTeXt use a more abstract model of boxes that are preprocessed before a run, which avoids the external run(s). In the new approach everything can be kept internal. The conversion even permits constructs like:

```
for i=1 upto 100 :
  draw btex oeps etex rotated i ;
endfor ;
```

but since this construct is kind of obsolete (at least in the library version of MetaPost) it is better to use:

```
for i=1 upto 100 :
  draw texttext("cycle " & decimal i) rotated i ;
endfor ;
```

Internally a trial pass is done so that indeed 100 different texts will be drawn. The throughput of texts is so high that in practice one will not even notice that this happens.

Dealing with text is another example of using Lpeg. The following snippet of code sheds some light on how text in graphics is dealt with. Actually this is a variation on a previous implementation. That one was slightly faster but looked more complex. It was also not robust for complex texts defined in macros in a format.

```
local P, S, V, Cs = lpeg.P, lpeg.S, lpeg.V,
    lpeg.Cs
```

```

local btex = P("btex")
local etex = P(" etex")
local vtex = P("verbatimex")
local ttex = P("texttext")
local gtex = P("graphicstext")
local spacing = S("\n\r\t\v")^0
local dquote = P('\'')

local found = false

local function convert(str)
  found = true
  return "texttext(\" .. str .. "\")"
end
local function ditto(str)
  return "\" & ditto & \""
end
local function register()
  found = true
end

local parser = P {
  [1] = Cs((V(2)/register
           + V(3)/convert + 1)^0),
  [2] = ttex + gtex,
  [3] = (btex + vtex) * spacing
        * Cs((dquote/ditto + (1-etex))^0)
        * etex,
}

function metapost.check_texts(str)
  found = false
  return parser:match(str), found
end

```

If you are unfamiliar with Lpeg it may take a while to see what happens here: we replace the text between `btex` and `etex` by a call to `texttext`, a macro. Special care is given to embedded double quotes.

When text is found, the graphic is processed two times. The definition of `texttext` is different for each run. For the first run we have:

```

vardef texttext(expr str) =
  image (
    draw unitsquare
      withprescript "tf"
      withpostscript str ;
  )
enddef ;

```

After the first run the result is not really converted, just the outlines with the `tf` prescript are filtered. In the loop over the object there is code like:

```

local prescript = object.prescript
if prescript then
  local special = metapost.specials[prescript]

```

```

  if special then
    special(object.postscript,object)
  end
end

function metapost.specials.tf(specification,
                              object)
  tex.sprint(tex.ctxcatcodes,
             format("\MPLIBsettext{%s}{%s}",
                  metapost.texttext_current,specification))
  if metapost.texttext_current
    < metapost.texttext_last then
    metapost.texttext_current
      = metapost.texttext_current + 1
  end
  ...
end

```

Again, you can forget about the details of this function. What's important is that there is a call out to $\text{T}_{\text{E}}\text{X}$ that will process the text. Each snippet gets the number of the box that holds the content. The macro that is called just puts stuff in a box:

```

\def\MPLIBsettext#1#2%
  {\global\setbox#1\hbox{#2}}

```

In the next processing cycle of the MetaPost code, the `texttext` macro does something different :

```

vardef texttext(expr str) =
  image (
    _tt_n_ := _tt_n_ + 1 ;
    draw unitsquare
      xscaled _tt_w_[_tt_n_]
      yscaled (_tt_h_[_tt_n_] + _tt_d_[_tt_n_])
      withprescript "ts"
      withpostscript decimal _tt_n_ ;
  )
enddef ;

```

This time the (by then known) dimensions of the box storing the snippet are used. These are stored in the `_tt_w_`, `_tt_h_` and `_tt_d_` arrays. The arrays are defined by Lua using information about the boxes, and passed to the library before the second run. The result from the second MetaPost run is converted, and again the prescript is used as trigger:

```

function metapost.specials.ts(specification,
                              object,result)
  local op = object.path
  local first, second, fourth
    = op[1], op[2], op[4]
  local tx, ty
    = first.x_coord, first.y_coord
  local sx, sy
    = second.x_coord - tx, fourth.y_coord - ty

```

```

local rx, ry
  = second.y_coord - ty, fourth.x_coord - tx
if sx == 0 then sx = 0.00001 end
if sy == 0 then sy = 0.00001 end
metapost.flushfigure(result)
tex.sprint(tex.ctxcatcodes,format(
  "\\MPLIBgettext{%f}{%f}{%f}{%f}{%f}{%f}{%s}",
  sx,rx,ry,sy,tx,ty,
  metapost.texttext_current))
...
end

```

At this point the converter is actually converting the graphic and passing PDF literals to TeX. As soon as it encounters a text, it flushes the PDF code collected so far and injects some TeX code. The TeX macro looks like:

```

\def\MPLIBgettext#1#2#3#4#5#6#7%
  {\ctxlua{metapost.sxsy(\number\wd#7,
    \number\ht#7,\number\dp#7)}%
  \pdfliteral{q #1 #2 #3 #4 #5 #6 cm}%
  \vbox to \zeropoint{\vss\hbox to \zeropoint
    {\scale[sx=\sx,sy=\sy]{\raise\dp#7\box#7}%
    \hss}}%
  \pdfliteral{Q}}

```

Because text can be transformed, it needs to be scaled back to the right dimensions, using both the original box dimensions and the transformation of the unitsquare associated with the text.

```

local factor = 65536*(7200/7227)
-- helper for text
function metapost.sxsy(wd,ht,dp)
  commands.edef("sx", (wd ~= 0 and
    1/( wd / (factor))) or 0)
  commands.edef("sy", (wd ~= 0 and
    1/((ht+dp)/(factor))) or 0)
end

```

So, in fact there are the following two processing alternatives:

- tex: call a Lua function that processes the graphic
- lua: parse the MetaPost code for texts and decide if two runs are needed

Now, if there was no text to be found, the continuation is:

- lua: process the code using the library
- lua: convert the resulting graphic (if needed) and check if texts are used

Otherwise, the next steps are:

- lua: process the code using the library
- lua: parse the resulting graphic for texts (in the postscripts) and signal TeX to process these texts afterwards
- tex: process the collected text and put the

result in boxes

- lua: process the code again using the library but this time let the unitsquare be transformed according to the text dimensions
- lua: convert the resulting graphic and replace the transformed unitsquare by the boxes with text

The processor itself is used in the MkIV graphic function that takes care of the multiple passes mentioned before. To give you an idea of how it works, here is how the main graphic processing function roughly looks.

```

local current_format, current_graphic

function metapost.graphic_base_pass(mpsformat, str,
                                     preamble)
  local prepared, done
  = metapost.check_texts(str)
  metapost.texttext_current
  = metapost.first_box
  if done then
    current_format, current_graphic
    = mpsformat, prepared
    metapost.process(mpsformat, {
      preamble or "",
      "beginfig(1); ",
      "_trial_run_ := true ;",
      prepared,
      "endfig ;"
    }, true ) -- true means: trialrun
    tex.sprint(tex.ctxcatcodes,
      "\\ctxlua{metapost.graphic_extra_pass()}")
  else
    metapost.process(mpsformat, {
      preamble or "",
      "beginfig(1); ",
      "_trial_run_ := false ;",
      str,
      "endfig ;"
    } )
  end
end

function metapost.graphic_extra_pass()
  metapost.texttext_current = metapost.first_box
  metapost.process(current_format, {
    "beginfig(0); ",
    "_trial_run_ := false ;",
    table.concat(metapost.text_texts_data(),
      " ;\n"),
    current_graphic,
    "endfig ;"
  } )
end

```

The box information is generated as follows:

```

function metapost.text_texts_data()

```

```

local t, n = { }, 0
for i = metapost.first_box, metapost.last_box
do
  n = n + 1
  if tex.box[i] then
    t[#t+1] = format(
      "_tt_w_[%i]:=f;_tt_h_[%i]:=f;_tt_d_[%i]:=f;",
      n,tex.wd[i]/factor,
      n,tex.ht[i]/factor,
      n,tex.dp[i]/factor
    )
  else
    break
  end
end
return t
end

```

This is a typical example of accessing information available inside T_EX from Lua, in this case information about boxes.

The `trial_run` flag is used at the MetaPost end; in fact the `texttext` macro looks as follows:

```

vardef texttext(expr str) =
  if _trial_run_ :
    % see first variant above
  else :
    % see second variant above
  fi
enddef ;

```

This trickery is not new. We have used it already in ConT_EXt for some time, but until now the multiple runs took way more time and from the perspective of the user this all looked much more complex.

It may not be that obvious, but in the case of a trial run (for instance when texts are found), after the first processing stage, and during the parsing of the result, the commands that typeset the content will be printed to T_EX. After processing, the command to do an extra pass is printed to T_EX also. So, once control is passed back to T_EX, at some point T_EX itself will pass control back to Lua and do the extra pass.

The base function is called in:

```

function metapost.graphic(mpsformat,str,
                          preamble)
  local mpx = metapost.format(mpsformat
                              or "metafun")
  metapost.graphic_base_pass(mpx,str,preamble)
end

```

The `metapost.format` function is part of the `mplib-run` module. It loads the `metafun` format, possibly after (re)generating it.

Now, admittedly all this looks a bit messy, but in pure T_EX macros it would be even more so. Some-

time in the future, the postponed calls to `\ctxlua` and the explicit `\pdf literals` can and will be replaced by using direct node generation, but that requires a rewrite of the internal LuaT_EX support for PDF literals.

The snippets are part of the `mplib-*` files of MkIV. These files are tagged as experimental and will stay that way for a while yet. This is shown by the fact that by now we use a slightly different approach.

Summarizing the impact of M_Plib on extensions, we can conclude that some are done better and some more or less the same. There are some conceptual problems that prohibit using pre- and postscripts for everything (at least currently).

4 Integrating

The largest impact of M_Plib is processing graphics at runtime. In MkII there are two methods: real runtime processing (each graphic triggered a call to MetaPost) and collective processing (between T_EX runs). The first method slows down the T_EX run, the second method generates a whole lot of intermediate PostScript files. In both cases there is a lot of file I/O involved.

In MkIV, the integrated library is capable of processing thousands of graphics per second, including conversion. The preliminary tests (which involved no extensions) involved graphics with 10 random circles drawn with penshapes in random colors, and the throughput was around 2000 such graphics per second on a 2.3 MHz Core Duo:



In practice there will be more overhead involved than in the tests. For instance, in ConT_EXt information about the current state of T_EX has to be passed on also: page dimensions, font information, typesetting related parameters, preamble code, etc.

The whole T_EX interface is written around one process function:

```

metapost.graphic(metapost.format("metafun"),
                 "mp code")

```

Optionally a preamble can be passed as the third argument. This one function is used in several other macros, like:

```

\startMPcode          ... \stopMPcode
\startMPpage          ... \stopMPpage
\startuseMPgraphic{name} ...
\stopuseMPgraphic

```

```

\startreusableMPgraphic{name}...
\stopreusableMPgraphic
\startuniqueMPgraphic {name}...
\stopuniqueMPgraphic

\useMPgraphic{name}
\reuseMPgraphic{name}
\uniqueMPgraphic{name}

```

The user interface is downward compatible: in MkIV the same top-level commands are provided as in MkII. However, the (previously required) configuration macros and flags are obsolete.

This time, the conclusion is that the impact on ConTeXt is immense: The code for embedding graphics is very clean, and the running time for graphics inclusion is now negligible. Support for text in graphics is more natural now, and takes no run-time either (in MkII some parsing in TeX takes place, and if needed long lines are split; all this takes time).

In the styles that Pragma ADE uses internally, there is support for the generation of placeholders for missing graphics. These placeholders are MetaPost graphics that have some 60 randomly scaled circles with randomized colors. The time involved in generating 50 such graphics is (on my machine) some 14 seconds, while in LuaTeX only half a second is needed.



Because LuaTeX needs more startup time and deals with larger fonts resources, pdfTeX is generally faster, but now that we have MPlib, LuaTeX suddenly is the winner.

Putting the Cork back in the bottle—Improving Unicode support in T_EX

Mojca Miklavc

Faculty of Mathematics and Physics, University of Ljubljana

Arthur Reutenauer

GUTenberg, France

<http://tug.org/tex-hyphen>

Abstract

Until recently, all of the hyphenation patterns available for different languages in T_EX were using 8-bit font encodings, and were therefore not directly usable with UTF-8 T_EX engines such as X_ƒT_EX and LuaT_EX. When the former was included in T_EX Live in 2007, Jonathan Kew, its author, devised a temporary way to use them with X_ƒT_EX as well as the “old” T_EX engines. Last spring, we undertook to convert them to UTF-8, and make them usable with both sorts of T_EX engines, thus staying backwardly compatible. The process uncovered a lot of idiosyncrasies in the pattern-loading mechanism for different languages, and we had to invent solutions to work around each of them.

1 Introduction

Hyphenation is one of the most prominent features of T_EX, and since it is possible to adapt it to many languages and writing systems, it should come as no surprise that there were so many patterns created so quickly for so many languages in the relatively early days of T_EX development. As a result, the files that are available often use old and dirty tricks, in order to be usable with very old versions of T_EX. In particular, all of them used either 8-bit encodings or accent macros (`\’e`, `\v{z}`, etc.); Unicode did not yet exist when most of these files were written.

This was a problem when X_ƒT_EX was included in T_EX Live in 2007, since it expects UTF-8 input by default. Jonathan Kew, the X_ƒT_EX author, devised a way of using the historical hyphenation patterns with both X_ƒT_EX and the older extensions of T_EX: for each pattern file `<hyph>.tex`, he wrote a file called `xu-<hyph>.tex` that detects if it is run with X_ƒT_EX or not; in the latter case, it simply inputs `<hyph>.tex` directly, and otherwise, it takes actions to convert all the non-ASCII characters to UTF-8, and then inputs the pattern file.

To sum up, in T_EX Live 2007, X_ƒT_EX used the original patterns as the basis, and converted them to UTF-8 on the fly.

In the ConT_EXt world, on the other hand, the patterns had been converted to UTF-8 for a couple of years, and were converted back to 8-bit encodings by the macro package, depending on the font encoding.

In an attempt to go beyond that and to unify those approaches, we then decided to take over conversions for all the pattern files present in T_EX Live at that time (May 2008), for inclusion in the 2008 T_EX Live release.

2 The new architecture

The core idea is that after converting the patterns to UTF-8, the patterns are embedded in a structure that can make them loadable with both sorts of T_EX engines, the ones with native UTF-8 support (X_ƒT_EX, LuaT_EX) as well as the ones that support only 8-bit input.¹

The strategy for doing so was the following: for each language `<lang>`, the patterns are stored in a file called `hyph-<lang>.tex`. These files contain only the raw patterns, hyphenation exceptions, and comments. They are input by files called `loadhyph-<lang>.tex`. This is where engine detection happens, such as this code for Slovenian:

```
% Test whether we received one or two arguments
\def\testengine#1#2!{\def\secondarg{#2}}
% We are passed Tau (as in Taco or TEX,
% Tau-Epsilon-Chi), a 2-byte UTF-8 character
\testengine T!\relax
% Unicode-aware engines (such as XeTeX or LuaTeX)
% only see a single (2-byte) argument
\ifx\secondarg\empty
\message{UTF-8 Slovenian Hyphenation Patterns}
\else
\message{EC Slovenian Hyphenation Patterns}
\input conv-utf8-ec.tex
\fi
\input hyph-sl.tex
```

The only trick is to make T_EX look at the Unicode character for the Greek capital Tau, in UTF-8 encoding: it uses two bytes, which are therefore read by 8-bit T_EX engines as two different characters; thus

¹ A note on vocabulary: in this article, we use the word “engine” or “T_EX engine” for extensions to the program T_EX, in contrast to macro packages. We then refer to (T_EX) engines with native UTF-8 support as “UTF-8 engines”, and to the others as “8-bit engines”, or sometimes “legacy engines”, borrowing from Unicode lingo.

the macro `\testengine` sees two arguments. UTF-8 engines, on the other hand, see a single character (Greek capital Tau), thus a single argument before the exclamation mark, and `\secondarg` is `\empty`.

If we’re running a UTF-8 T_EX engine, there is nothing to do but to input the file with the UTF-8 patterns; but if we’re running an 8-bit engine, we have to convert the UTF-8 byte sequences to a single byte in the appropriate encoding. For Slovenian, as for most European languages written in the Latin alphabet, it happens to be T1. This conversion is taken care of by a file named `conv-utf8-ec.tex` in our scheme. Let’s show how it works with these three characters:²

- ‘č’ (UTF-8 `<0xc4, 0x8d>`, T1 `0xa3`),
- ‘š’ (UTF-8 `<0xc5, 0xa1>`, T1 `0xb2`),
- ‘ž’ (UTF-8 `<0xc5, 0xbe>`, T1 `0xba`).

In order to convert the sequence `<0xc4, 0x8d>` to `0xa3`, we make the byte `0xc4` active, and define it to output `0xa3` if its argument is `0x8d`.³ The other sequences work in the same way, and the extracted content of `conv-utf8-ec.tex` is thus:⁴

```
\catcode"C4=\active
\catcode"C5=\active
%
\def^^c4#1{%
\ifx#1^^8d^^a3\else % U+010D
\fi}
%
\def^^c5#1{%
\ifx#1^^a1^^b2\else % U+0161
\ifx#1^^be^^ba\else % U+017E
\fi\fi}
% ensure all the chars above have valid lccode's:
\lccode"A3="A3 % U+010D
\lccode"B2="B2 % U+0161
\lccode"BA="BA % U+017E
```

As the last comment says, we also need to set non-zero `\lccodes` for the characters appearing in the pattern files, a task formerly carried out in the pattern file itself.

The information for converting from UTF-8 to the different font encodings has been retrieved from the encoding definition files for L^AT_EX and ConT_EXt, and gathered in files called `<enc>.dat`. The converter files are automatically generated with a Ruby script from that data.

² The only non-ASCII characters in Slovenian.

³ The same method would work flawlessly if the sequence contained three or more bytes—although this case doesn’t arise in our patterns—since the number of bytes in a UTF-8 sequence depends only on the value of the first byte.

⁴ Problems would happen if a T1 byte had been made active in that process, but for reasons inherent to the history of T_EX font encodings, as well as Unicode, this *is never the case for the characters used in the patterns*, a fact the authors consider a small miracle. The proof of this is much too long to be given in this footnote, and is left to the reader.

Here is a table of the encodings we support:

ConT _E Xt	L ^A T _E X	Comments
<code>ec</code>	T1	“Cork” encoding
<code>il2</code>	latin2	ISO 8859-2
<code>il3</code>	latin3	ISO 8859-3
<code>lmc</code>	lmc	montex (Mongolian)
<code>qx</code>	qx	Polish
<code>t2a</code>	t2a	Cyrillic

2.1 Language tags: BCP 47 / RFC 4646

A word needs to be said about the language tags we used. As a corollary to the completely new naming scheme for the pattern files and the files surrounding them, we wanted to adopt a consistent naming policy for the languages, abandoning the original names completely, because they were problematic in some places. Indeed, they used ad hoc names which had been chosen by very different people over many years, without any attempt to be systematic; this has led to awkward situations; for example, the name `ukhyphen.tex` for the British English patterns: while “UK” is easily recognized as the abbreviation for “United Kingdom”, it could also be the abbreviation for “Ukrainian”, and unless one knows all the names of the pattern files by heart, it is not possible to determine what language is covered by that file from the name alone.

It was therefore clear that in order to name files that had to do with different *languages*, we had to use language codes, not country codes. But this was not sufficient either, as can be seen from the example of British English, since it’s not a different language from English.

Upon investigation, it turned out that the only standard able to distinguish all the patterns we had was the IETF “Best Current Practice” recommendation 47 (BCP 47), which is published as RFC documents; currently, it’s RFC 4646.⁵ This addresses all the language variants we needed to tag:

- Languages with variants across countries or regions, like English.
- Languages written in different scripts, like Serbian (Latin and Cyrillic).
- Languages with different spelling conventions, like Modern Greek (which underwent a reform known as *monotonic* in 1982), and German (for which a reform is currently happening, started in 1996).

⁵ In the past, it has been RFC 1766, then RFC 3066, and is currently being rewritten, with the working title RFC 4646bis. RFC 4646 is available at <ftp://ftp.rfc-editor.org/in-notes/rfc4646.txt>, and the current working version of RFC 4646bis (draft 17) at <http://www.ietf.org/internet-drafts/draft-ietf-ltru-4646bis-17.txt>.

A list of all the languages with their tags can be found in appendix A.

3 Dealing with the special cases

There were so many special cases that one might say that the generic case was the special one!

3.1 Pattern files designed for multiple encodings

The first problem we encountered was with patterns that tried to accommodate both the OT1 and the T1 encoding in the same file.

The first language for which this had been done was, historically, German, and the same scheme was subsequently adopted for French, Danish, and Latin. The idea is the following: in each of these languages, there are characters that are encoded at different positions in OT1 and in T1; for German, it is the sharp s ‘ß’; for French, it is the character ‘œ’, etc. In order to deal with that, each pattern that happened to contain one of these characters was duplicated in the file, with intricate macros to ignore them selectively, depending on the font encoding used.

This would have been very awkward to reproduce in our architecture, if at all possible: it would have meant that each word such as, say, “cœur” in French would need to yield two different byte strings in 8-bit mode, for OT1 and T1 (c¹bur and c^fur, respectively). We therefore decided to put the duplicate patterns in a separate file called `spechyp-⟨lang⟩-ot1.tex` that is input only in legacy mode, after the main file `hyph-⟨lang⟩.tex`.

The patterns packaged in this fashion should therefore behave in the same way as the historical files, enabling a few breakpoints with non-ASCII characters in OT1 encoding. We would like to stress, though, that OT1 is definitely not the way to go for these languages. We only supported this behaviour for the sake of compatibility, but we doubt it is very useful: if one uses OT1 for German or French, one would indeed have a few patterns with ‘ß’ or ‘œ’, respectively, but many more patterns, with accented characters, would be missed. In order to take full advantage of the hyphenation patterns, one needs to use T1 fonts.

It has to be noted that in addition, we ended up not using the aforementioned approach in the case of German, because we wanted to account for the ongoing work to improve the German patterns; thus, we decided to use the new patterns with the UTF-8 engines, but not with the 8-bit engines, for compatibility reasons. In the latter case, we simply include the original pattern file in T1 directly, with no conversion whatsoever. For the three other lan-

guages, though (French, Danish and Latin), we used a `spechyp-⟨lang⟩-ot1.tex` file.

3.2 Multiple pattern sets for the same language

Another interesting issue was with Ukrainian and Russian, where different complications arose.

First, the pattern files were also devised for multiple encodings, but in a different manner: here, the encoding is selected by setting the control sequence `\Encoding` before the pattern file is loaded. Depending on the value of that macro, the appropriate conversion file is then input, that works in the same way as our `conv-utf8-⟨enc⟩.tex` files. There is of course a default value for `\Encoding`, which for both languages is T2A,⁶ the most widespread font encoding for Russian and Ukrainian, and the one used in the pattern files; thus, no conversion is necessary if `\Encoding` is kept to its default value.

Then, both Russian and Ukrainian had several pattern files, with different authors and/or hyphenation rules (phonetic, etymological, etc.). Those were selected with a control sequence called `\Pattern`, by default `as` for Russian (by Aleksandr Lebedev), and `mp` for Ukrainian (by Maksym Polyakov).

Both those choices could, of course, be overridden only at format-building time, since the patterns are frozen at that moment.

Finally, they used a special trick, implemented in file `hyph2.tex`, to enable hyphenation inside words containing hyphens, similar to Bernd Raichle’s `hyph1.tex` for T1 fonts.

Those three features had to be addressed in very different ways in our structure: while the first one was irrelevant in UTF-8 mode, it would have implied fundamental changes in our `loadhyph-⟨lang⟩.tex` files for 8-bit engines, since the implicit assumption that any language uses exactly one 8-bit encoding would no longer be met. The second feature was easier to handle, but still demanded additional features in our `loadhyph-⟨lang⟩.tex` files. Finally, the third feature, although certainly very interesting, seemed more fragile than what we felt was acceptable.

Upon deliberation, we then decided to not include those features in the UTF-8 patterns before T_EX Live 2008 was out, but to still enable them in legacy mode, in order to ensure backward compatibility. And thanks to subsequent discussions with Vladimir Volovich, who devised the way the Russian patterns were packaged, and inspired the Ukrainian ones, we could include a list of hyphenated compound words which we put in files called `exhyph-ru.tex`

⁶ Actually `t2a`, lowercase.

and `exhyph-uk.tex`, respectively. The strategy we used is thus:

- In UTF-8 mode, input the UTF-8 patterns, then the `ex-` file.
- In legacy mode, simply input the original pattern file directly.

Therefore, the only feature missing, overall, in T_EX Live 2008, is the ability to choose one’s favorite patterns in UTF-8 mode: for each language, we only converted the default set of patterns to UTF-8. Setting `\Pattern` will thus have no effect in this case, but it will behave as before in 8-bit mode. Now that T_EX Live 2008 has been released we intend to change that behaviour soon, and to enable the full range of features that the original pattern files had.

It should also be noted that in T_EX Live 2007, Bulgarian used the same pattern-loading mechanism, but that there was actually only one possible encoding, and only one pattern file, so there was no real choice, and it was therefore straightforward to adapt the Bulgarian patterns to our new architecture.

4 T_EX Live 2008

The result of our work has been put on CTAN under the package name `hyph-utf8`, and is the basis for hyphenation support in T_EX Live 2008. We don’t consider our work to be finished (see next section), and we welcome any discussion on our mailing-list (`tex-hyphen@tug.org`). We also have a home page at <http://tug.org/tex-hyphen>, to which readers are referred for more information.

The package has been released in the TDS layout, with the T_EX files in `tex/generic/hyph-utf8` and subdirectories. The encoding data and Ruby scripts are available in `source/generic/hyph-utf8`. Some language-specific documentation has been put in `doc/generic/hyph-utf8`.

5 And now . . .

There still are tasks we would like to carry out: the `hypht1.tex / hypht2.tex` behaviour has already been mentioned, and one of the authors has lots of ideas on how to improve Unicode support *yet more* in UTF-8 T_EX engines.

We appeal to pattern authors to make contact with us in order to improve and enhance our package; many of them have already communicated with us, to our greatest pleasure, and we’re confident that our effort will be understood by all the developers dealing with language-related problems.⁷

⁷ The acknowledgement section, had it been as long as the authors would have wished it to be, would have more than doubled the size of this article.

Among the immediate and practical problems is, in particular:

5.1 . . . for something completely different

Babel would need to be enhanced in order to enable different “variants” for at least two languages. One is Norwegian, for which two written forms exist, known as “bokmal” and “nynorsk” (ISO 639-1 `nb` and `nn`, respectively).⁸ At the moment, Babel has only one “Norwegian” language. The second is Serbian, which can be written in both the Latin and the Cyrillic alphabets; these possible variants which are not yet taken into account in Babel.

6 Acknowledgements

First and foremost, we wish to thank wholeheartedly Karl Berry, who supported the project from the beginning and guided us with advice, as well as Hans Hagen, Taco Hoekwater and Jonathan Kew, for their technical help, and, finally, Norbert Preining, who went through the trouble of integrating the new package into T_EX Live.

Appendix A List of supported languages

<code>ar</code> Arabic	<code>la</code> Latin
<code>fa</code> Farsi	<code>mn-cyrl</code> Mongolian
<code>eu</code> Basque	<code>mn-cyrl-x-2a</code> Mongolian (new patterns)
<code>bg</code> Bulgarian	<code>no</code> Norwegian
<code>cop</code> Coptic	<code>nb</code> Norwegian Bokmal
<code>hr</code> Croatian	<code>nn</code> Norwegian Nynorsk
<code>cs</code> Czech	<code>zh-latn</code> Chinese Pinyin
<code>da</code> Danish	<code>pl</code> Polish
<code>nl</code> Dutch	<code>pt</code> Portuguese
<code>eo</code> Esperanto	<code>ro</code> Romanian
<code>et</code> Estonian	<code>ru</code> Russian
<code>fi</code> Finnish	<code>sr-latn</code> Serbian, Latin script
<code>fr</code> French	<code>sr-cyrl</code> Serbian, Cyrillic script
<code>de-1901</code> German, “old” spelling	<code>sh-latn</code> Serbo-Croatian, Latin script
<code>de-1996</code> German, “new” spelling	<code>sh-cyrl</code> Serbo-Croatian, Cyrillic script
<code>e1-monoton</code> Monotonic Greek	<code>sl</code> Slovene
<code>e1-polyton</code> Polytonic Greek	<code>es</code> Spanish
<code>grc</code> Ancient Greek	<code>sv</code> Swedish
<code>grc-x-ibycus</code> Ancient Greek, Ibycus encoding	<code>tr</code> Turkish
<code>hu</code> Hungarian	<code>en-gb</code> British English
<code>is</code> Icelandic	<code>en-us</code> American English
<code>id</code> Indonesian	<code>uk</code> Ukrainian
<code>ia</code> Interlingua	<code>hsb</code> Upper Sorbian
<code>ga</code> Irish	<code>cy</code> Welsh
<code>it</code> Italian	

⁸ The ISO standard also includes a code for “Norwegian”, `no`, although this name is formally ambiguous.

Writing Gregg Shorthand with METAFONT and L^AT_EX

Stanislav Jan Šarman

Computing Centre

Clausthal University of Technology

Erzstr. 51

38678 Clausthal

Germany

Sarman (at) rz dot tu-clausthal dot de

http://www3.rz.tu-clausthal.de/~rzsjs/steno/Gregg.php

Abstract

We present an online system, which converts English text into Gregg shorthand, a phonetic pen writing system used in the U.S. and Ireland.

Shorthand is defined [2] as a method of writing derived from the spelling system and the script of a particular language. In order to approximate the speed of speech:

- The graphic redundancy of written characters and ligatures derived from longhand letters is reduced.
- Silent letters are ignored, and complex orthographic rules are simplified. The spelling only serves as a reference system of what is largely phonetic writing.
- Abbreviations are utilized.

Gregg shorthand was first published in 1888. In the following three sections we want to show how its inventor J. R. Gregg has applied the above general principles in the creation of his shorthand system and how the current system version, the centennial edition [7], can be implemented in METAFONT [5].

1 Simplified writing

1.1 The alphabet of Gregg Shorthand

Signs of consonant phonemes¹ written forward:

(k) (g) (r) (l) (n) (m) (t) (d) (th)(Th)



Signs of consonant phonemes written downward:

(p) (b) (f) (v) (sh) (ch) (jh) (s)(S)



The signs of voiced consonant phonemes are larger versions of their unvoiced opposites. As the “minimal pairs”² of words with phonemes *s* vs. *z*, *th* vs. *dh* and *sh* vs. *zh* are very rare, there are only the unvoiced versions of these signs. On the other hand

¹ Phonemes are denoted in typewriter type, their signs are parenthesized or bracketed.

² such as face vs. phase, sooth vs. soothe, mesher vs. measure, pronunciations of which differ only in one place

two forms of signs exist for *s* and *th*: the right (*s*) , (*th*)  and the left (*S*) , (*Th*) , respectively.

Blended consonant signs:

(ld) (nd) (rd) (tm) (tn) (td)(dd)



can be thought of as built-in ligatures.

Vowel signs:

for sound(s) as in

[a]  at, art, ate 

[e]/[i]  pit, pet, peat 

(o)  pot, port , pour 

(u)  tuck, took , tomb 

Elementary diphthong (triphone) signs:

for sounds as in

[ai]  high 

[aee]  client 

[iie]  create 

When writing shorthand the curvilinear motion of a pen in the slope of longhand traces ellipse or oval-based curves. We shape these elementary shorthand signs in METAFONT as continuous curvature splines with zero curvature at the end points.

Half-consonants *h*, *w*, prefixes and suffixes are realized as markers, e.g. a dot over *o* [i] such as in “he”; *h* means preceding *h*; another dot under (along) the sign (*t*)  such as in  denotes the suffix -ing in “heating”.

1.2 Metaform

Gregg shorthand is an alphabetic system — written words are combined together from elementary signs by joining, so that in general the Gregg shorthand glyphs have the following *metaform*:

$\{ (\{p,\}C\{,s\}) \mid [+]\{V\{,s\}\}^+ \}$

where *V* are circular vowels, *C* denotes (blended) consonants and vowels *u* and *o*, optional *p* and *s*

stand for markers of semiconsonants, prefixes or suffixes. In more detail:

$V ::= a | e | i | ai | aee | iie$
 $C ::= b | d | dd \dots o | u$
 $p ::= h | w | \dots \text{over} | \text{under} \dots$
 $s ::= \text{ing} | \text{ly} | \dots$

Examples:

	metaform	
he	-[h,i,]	ó
heating	-[h,i,](,t,ing)	ó
need	(n)-[i](d)	ó
needed	(n)-[i](dd)	ó
needing	(n)-[i](,d,ing)	ó

The metaform corresponds directly to the METAFONT program code, e.g.:

$(n)-[i](,d,ing) \leftrightarrow I(n,); V(-1,,i,); C(d,ing);$
 i.e. the shorthand glyph of this particular character is initiated with (n), then the signs of right vowel -[i] and the sign of consonant d with suffix ing are appended.

1.3 Joining

We distinguish between joinings where circular signs are involved and joinings without them.

1.3.1 CC Joinings

Examples of CC joinings ordered by their connectivity grade $G^{(0)}-G^{(2)}$ follow:

continuous: \swarrow (d)(n), \curvearrowright (u)(n), \nearrow (t)(s)
 (with turning point!)

tangent continuous: $\swarrow + \curvearrowright = \curvearrowright$ (f)(l),
 $\swarrow + \curvearrowleft = \curvearrowleft$ (p)(r), $\nearrow + \curvearrowleft = \curvearrowleft$ (u)(p),
 $\swarrow + \nearrow = \curvearrowright$ (g)(v)

curvature continuous: $\curvearrowright + \curvearrowleft = \curvearrowright$ (r)(k)

The first two types of joinings are handled by METAFONT means; $G^{(2)}$ continuous connecting can be done only in special cases using Hermite interpolation for Bézier splines, as follows.

If two endpoints z_i , unit tangents d_i , $d_i d_i^t = 1$ and curvatures κ_i , $i = 1, 2$ are given, the control points $z_0^+ = z_0 + \delta_0 d_0$, $z_1^- = z_1 - \delta_1 d_1$ have to be determined from the following system of nonlinear equations [3]:

$$(d_0 \times d_1) \delta_0 = (a \times d_1) - \frac{3}{2} \kappa_1 \delta_1^2 \quad (1)$$

$$(d_0 \times d_1) \delta_1 = (d_0 \times a) - \frac{3}{2} \kappa_0 \delta_0^2 \quad (2)$$

($a = z_1 - z_0$).

If the tangents are parallel, i.e. $(d_0 \times d_1) = 0$ and the curvatures κ_i have opposite signs, the equations (1)–(2) are trivially solvable, so that $G^{(2)}$ joining is possible (here after some kerning):



Figure 1: An example of a $G^{(2)}$ continuous CC joining

1.3.2 Joinings with circular signs

The signs of consonants can be classified into seven categories. Looped vowels before or after a consonant are written within the curves or combined with straight consonants as right curves, so that the following VC, CV prototype joinings can occur:

	\curvearrowright	\curvearrowleft	$-$	\swarrow	\nearrow	\curvearrowright	\curvearrowleft
VC	ó	e	o	o	o	o	o
CV	o	o	o	o	o	o	o

Observe the realization of diphthongs as VC or CV joinings, too:

for sounds as in
 -[a](u) ó how ó
 (o)+[i] o toy o
 -[i](u) o few o

All VC or CV joinings are done in such a manner that both components have identical tangents and curvature equal to zero at the connection point. As an example of $G^{(2)}$ connecting of a consonant sign with a vowel sign consider prepending the sign o of -[a] before \curvearrowright (k) as in o:

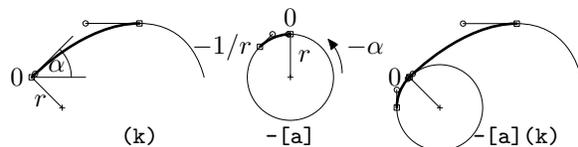


Figure 2: An example of a $G^{(2)}$ continuous VC joining

Here the spline approximation of (k) is such that the curvature equals zero at the end points; this is the case with other consonants, too. The spline approximation of right circle with radius r for the sign -[a] consists of 8 segments such that the curvature $\kappa = 0$ at the top point and $\kappa \approx -1/r$ at the other points. Let α be the angle of the tangent in the zeroth point of (k). Placing the -[a] circle at the point with distance r from the zeroth point of (k) and rotating it by the angle of $-\alpha$ a curvature continuous joining is obtained.

For a $G^{(2)}$ continuous spline approximation of the unit circle with curvature $\kappa = 0$ in one point we

use the traditional Bézier spline circle segment approximation for the segments 1–7 (see the 7th segment on the left) $z^{(1/2)} = 1$, so that $\delta_0 = \delta_1 = \frac{4}{3} \tan(\theta/4)$ and $\kappa_i = -(1 - \sin^4(\theta/4))$. Using this value as left point curvature of the 8th segment (see on the right) and demanding $\kappa_1 = 0$, the Hermite interpolation with the equations (1)–(2) for the unknowns δ_0 und δ_1 can be solved for segments with $\theta < 60^\circ$.

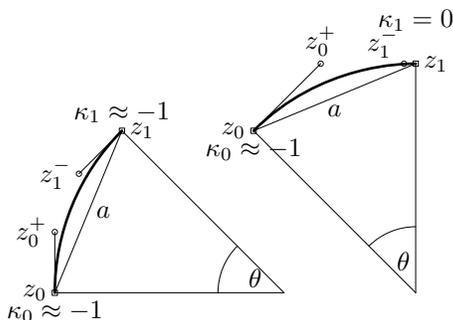


Figure 3: 7th and 8th segment ($\theta = 45^\circ$) of our unit right circle spline approximation

For CVC joinings 7×7 cases are possible in all — many of them can be transformed by reflection and rotation into one another. All these joinings can be made $G^{(2)}$ continuous. It must be decided before writing a right consonant, whether the loop is to be written as a left or as a right loop, compare:
team (t)+[i](m) vs. (m)-[i](t) meet

CVC	~	∪	-	/		()
~							
∪							
-							
/							
(
)							

Technically speaking the shorthand glyphs are realized as

- an array of $G^{(0)}$ – $G^{(2)}$ continuous METAFONT-paths and
- an array of discontinuous marker paths.

Slanting (default 22.5°) and tilting³ of characters is done at the time of their shipping.

³ $G^{(2)}$ continuity is invariant under affine transformations.

2 Phonetic writing

Gregg shorthand uses its own orthography, which must be acquired by the shorthand learner and in a system such as ours, the pronunciation of a word has to be known. We use the Unisyn multi-accent lexicon⁴ with about 118,000 entries [4] comprising British and American spelling variants. The fields of lexicon entries are: spelling, optional identifier, part-of-speech, pronunciation, enriched orthography, and frequency. Examples are:

```
owed;;VBD/VBN; { * ouw }> d >;{owe}>ed>;2588
live;1;VB/VBP; { l * i v } ;{live};72417
live;2;JJ; { l * ae v } ;{live};72417
```

Homonyma cases such as “live” above, in which the pronunciation helps to identify word meaning are much more rarer than the cases in which the use of pronunciation yields homophones resulting in shorthand homographs. Consider the very frequent right, rite, wright, write: { r * ai t } with shorthand glyph (r)+[ai](t) or the heterophones read;1;VB/NN/NNP/VBP; { r * ii d } ;{read};94567 read;2;VBN/VBD; { r * e d } ;{read};94567

both written as (r)+[e|i](d). Thus phonetic writing may speed up the shorthand recording, but the newly-created homographs complicate the deciphering of written notes.

The pronunciation of a word has to be transformed to the above defined metaform, e.g.:

```
needed: { n * ii d }.> I7 d >=> (n)-[i](dd)
needing: { n * ii d }.> i ng >=> (n)-[i](,d,ing)
```

This major task consists of a number of transformations done in this order:

- Elimination of minor (redundant) vowels such as the schwa @r in fewer: { f y * iu }> @r r >. There is a general problem with the schwas (@). Which are to be ruled out and which of them are to be backtransformed⁵ to the spelling equivalent? Consider data: { d * ee . t == @ } => (d)-[a](t)-[a] upon: { @ . p * o n } => (u)(p)(n) Both @ are backtransformed, but stressed o is cancelled.
- Finding of prefixes (un-, re-, ...), suffixes (-ing, -ly, ...), handling of semiconsonants h, w.
- Creation of ligatures such as dd, parenthesizing consonants and bracketing circular vowels.
- Finding the proper orientation of loops.

These transformations are done through a series of cascaded context dependent rewrite rules, realized

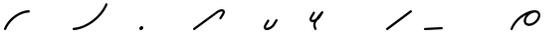
⁴ which itself is a part of a text-to-speech system
⁵ by a lex program

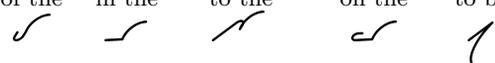
by finite state transducers (FSTs). An example of such a rule is

```
" I7" -> 0 || [ t | d ] " }.>" _ " d >"
which zeroes the string " I7" in the left context of
consonant t or d and the right context of "d >"; i.e.:
{ n * ii d }.> I7 d > => { n * ii d }.> dd >
Effectively this rule creates the ligature (dd) and
can be thought of as the reverse of the phonetic rule
" e" -> " I7" || [ t | d ] " }.>" _ " d >"
which determines the pronunciation of perfect tense
suffix > e d > in the left context of t or d:
{ n * ii d }.> I7 d > <=> { n * ii d }.> e d >
```

3 Abbreviations

The Gregg shorthand, like every other shorthand system since Tironian notes (43 B.C.), takes advantage of the statistics of speaking⁶ and defines about 380 so called *brief forms*. Here are the most common English words:

the and a to of was it in that

 (th) (nd) (a) (t)(o) (o) (u)(S) (t) (n) (th)-[a]

Writing the most common bigrams together, e.g.:
 of the in the to the on the to be


(o)(th) (n)(th) (t)(o)(th) (O)(n)(th) (t)(b)
 spares lifting the pen between words thus increasing shorthand speed.

These bigrams are entries in the dictionary of about 420 *phrases* as well as the following examples: as soon as possible: -[a](s)(n)(S)(p): 

if you cannot: +[i](f)(u)(k)(n): 

if you can be: +[i](f)(u)(k)(b): 

thank you for your order: (th)(U)(f)(u)(d): 

you might have: (u)(m)-[ai](t)(v): 

The abbreviation dictionary is coded in `lexc` [1].

4 text2Gregg

Our text2Gregg software is an online system,⁷ which records input text as Gregg shorthand. L^AT_EX input notation is accepted.

Homonyma variants such as
`latex;1,rubber; { l * ee . t e k s }` 
`latex;2,computing; { l * ee . t e k }` 

⁶ The 15 most frequent words in any text make up 25% of it; the first 100, 60%, ...

⁷ see project URL and also `DEK.php` for the German shorthand DEK counterpart [6]

are entered in the form `latex` (i.e. `latex#1`) and `latex#2`, respectively.

The task of the online conversion of given text to shorthand notes is done in the following four steps:

1. The input stream, stripped of L^AT_EX commands, is tokenized.⁸ There are two kinds of tokens:
 - Punctuation marks, numbers and common words for which a metaform entry in the abbreviation dictionary of *brief forms* and *phrases* exists and
 - the other words.
2. At first for a word of the latter category its pronunciation has to be found in Unisyn lexicon [4]. From this its metaform is built by a program coded as the tokenizer above in the XEROX-FST tool `xfst` [1].
3. In a `mf` run for each of the tokens using its metaform a shorthand glyph (i.e. a METAFONT character) is generated on-the-fly.
4. Then the text is set with L^AT_EX, rendered in the pipeline `→ dvips → gs → ppmtogif` and sent from the server to the browser.

PDF output with better resolution can be generated. Also a `djvu-backend` exists, which produces an annotated and searchable shorthand record. PostScript Type 3 vector fonts can be obtained, too.

References

- [1] Kenneth R. Beesley and Lauri Karttunen. *Finite State Morphology*. CSLI Publications, Stanford, 2003.
- [2] Florian Coulmas. *Blackwell Encyclopedia of Writing Systems*. Blackwell Publishers, 1999.
- [3] C. deBoor, K. Höllig, and M. Sabin. High accuracy geometric hermite interpolation. *Computer Aided Geometric Design*, 4:269–278, 1987.
- [4] Susan Fitt. Unisyn multi-accent lexicon, 2006. <http://www.cstr.ed.ac.uk/projects/unisyn/>.
- [5] Donald E. Knuth. *The METAFONTbook*, volume C of *Computers and Typesetting*. Addison-Wesley Publishing Company, Reading, Mass., 5th edition, 1990.
- [6] Stanislav Jan Šarman. DEK-Verkehrsschrift mit METAFONT und L^AT_EX schreiben. *Die TEXnische Komödie*, to be published.
- [7] Charles E. Zoubek. *Gregg Shorthand Dictionary, Centennial Edition, Abridged Version*. Glencoe Divison of Macmillan/McGraw-Hill, 1990.

⁸ Care has to be taken for proper nouns, which are underlined in Gregg shorthand.

The LuaTeX way: `\framed`

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

Abstract

ConTeXt's `\framed` macro has many flexible options for typesetting a paragraph. This short note discusses its reimplementaion in Lua for ConTeXt MkIV.

1 `\framed` and its width

One of the more powerful commands in ConTeXt is `\framed`. You can pass quite a few parameters that control the spacing, alignment, backgrounds and more. This command is used all over the place (although often hidden from the user) which means that it also has to be quite stable.

Unfortunately, there is one nasty bit of code that is hard to get right. Calculating the height of a box is not that complex: the height that TeX reports is indeed the height. However, the width of box is determined by the value of `\hsize` at the time of typesetting. The actual content can be smaller. In the `\framed` macro by default the width is calculated automatically.

```
\framed
  [align=middle,width=fit]
  {Out beyond the ethernet the spectrum spreads
   \unknown}
```

this shows up as (taken from ‘Casino Nation’ by Jackson Browne):

```
Out beyond the ethernet
the spectrum spreads ...
```

Or take this quote (from ‘A World Without Us’ by Alan Weisman):

```
\hsize=.6\hsize
\framed [align=middle,width=fit]
  {\input weisman }
```

This gives a multi-line paragraph:

```
Since the mid-1990s, humans
have taken an unprecedented
step in Earthly annals by
introducing not just exotic
flora or fauna from one
ecosystem into another, but
actually inserting exotic genes
into the operating systems of
individual plants and animals,
where they're intended to do
exactly the same thing: copy
themselves, over and over.
```

Here the outer `\hsize` was made a bit smaller. As you can see the frame is determined by the widest line. Because it was one of the first features we needed, the code in ConTeXt that is involved in determining the maximum natural width is pretty old. It boils down to unboxing a `\vbox` and stepwise grabbing the last box, penalty, kern and skip. That is, we unwind the box backwards.

However, one cannot grab everything; or, in TeX speak: there is only a limited number of `\lastsomething` commands. Special nodes, such as whatsits, cannot be grabbed and make the analyzer abort its analysis. There is no way that we can solve this in traditional TeX and in ConTeXt MkII.

2 `\framed` with LuaTeX

So how about LuaTeX and ConTeXt MkIV? The macro used in the `\framed` command is:

```
\doreshapeframedbox{do something
                    with \box\framebox}
```

In LuaTeX we can manipulate box content at the Lua level. Instead of providing a truckload of extra primitives (which would also introduce new data types at the TeX end) we delegate the job to Lua.

```
\def\doreshapeframedbox
  {\ctxlua{commands.doreshapeframedbox
          (\number\framebox)}}
```

Here `\ctxlua` is our reserved instance for ConTeXt, and `commands` provides the namespace for commands that we delegate to Lua (so, there are more of them). The amount of Lua code is far smaller than the TeX code (which we will not show here; it's in `supp-box.tex` if you want to see it).

```
function commands.doreshapeframedbox(n)
if tex.wd[n] ~= 0 then
  local hpack = node.hpack
  local free = node.free
  local copy = node.copy_list
  local noflines, lastlinelength, width = 0,0,0
  local list = tex.box[n].list
  local done = false
  for h in node.traverse_id('hlist',list) do
    done = true
```

```

local p = hpack(copy(h.list))
lastlinelength = p.width
if lastlinelength > width then
  width = lastlinelength
end
p.list = nil
free(p)
end
if done then
  if width ~= 0 then
    for h in node.traverse_id('hlist',list) do
      if h.width ~= width then
        h.list = hpack(h.list,width,'exactly')
        h.width = width
      end
    end
  end
  end
  tex.wd[n] = width
end
-- we can also work with lastlinelength
end
end

```

In the first loop we inspect all lines (nodes with type `hlist`) and repack them to their natural width with `node.hpack`. In the process we keep track of the maximum natural width. In the second loop

we repack the content again, this time permanently. Now we use the maximum encountered width which is forced by the keyword `exactly`. Because all glue is still present we automatically get the desired alignment. We create local shortcuts to some node functions which makes it run faster; keep in mind that this is a core function called many times in a regular ConTeXt job.

In looking at ConTeXt MkIV you will find quite a lot of Lua code and often it looks rather complex, especially if you have no clue why it's needed. Think of OpenType font handling which involves locating fonts, loading and caching them, storing features and later on applying them to node lists, etc.

However, once we are beyond the stage of developing all the code that is needed to support the basics, we will start doing the things that relate more to the typesetting process itself, such as the previous code. One of the candidates for a similar Lua-based solution is for instance column balancing. From the previous example code you can deduce that manipulating the node lists from Lua can make that easier. Of course we'll be a few more years down the road by then.

Advanced features for publishing mathematics, in PDF and on the Web

Ross Moore

Mathematics Department, Macquarie University, Sydney, Australia

ross@maths.mq.edu.au

<http://www.maths.mq.edu.au/staff/ross.html>

Abstract

Increasingly, mathematical, scientific and technical information is being distributed by electronic means, but having a high-quality paper printout remains important. We show here examples of techniques that are available for having both high-quality typesetting, in particular of mathematics, as well as useful navigation features and text extraction within electronic documents.

For HTML, we show some aspects of the use of `jsMath` within webpages; e.g., for mathematics journals or conference abstracts. With PDF, as well as the usual bookmarks and internal hyperlinks for cross-references and citations, advanced features include: (i) metadata attachments; (ii) copy/paste and searching for mathematical symbols or the underlying \LaTeX coding; (iii) pop-up images of (floating) figures and tables; (iv) mathematical symbols within bookmarks; (v) bookmarks for cross-referenced locations.

A further feature, particularly useful with mathematics papers, is the ability to make batched searches of the American Math. Society's *MathSciNet* database, allowing hyperlinks to be generated for most bibliography entries.

1 Introduction

The nature of scientific publication is changing: it is becoming increasingly common for articles to be accessed and read on-line, without the need for printing. However, many researchers still prefer to print out an article, having first obtained it as a PDF file, say. Thus it is necessary to produce the PDFs in such a way as to cater for both online and printed formats. For online reading, one needs navigation aids that give quick and easy access to cross-references, citations, metadata, and such. However, with retro-born versions of books and journal articles, the addition of such aids must not have an effect on the original pagination. Furthermore, for long-term digital archiving, accurate metadata and links to access cited materials become especially important.

There are effects that are possible with current web-based and PDF technologies, but which hitherto have not been widely used with scientific articles. The effects were programmed for processing with `pdf- \LaTeX` , along with extra packages and coding to adjust the output produced by \LaTeX internal macros. Almost no further adjustments were made to the body of files used to produce the original printed version of each paper, apart from the imposition of `\label` and `\ref` commands where they had not formerly been used with cross-references. Where papers had been submitted using $\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\text{\TeX}$, or even Plain \TeX , then some extra markup was needed also

with sectioning commands and other environments.

Here such effects are presented in a graphical way, as figures with extensive captions. Some extra text explains aspects that are not immediately apparent from seeing the images alone, along with a short outline of how these were implemented in \LaTeX , in terms of which (internal) macros needed to be patched, and what extra resources were used.

The HTML examples in Section 3 show use of Davide Cervone's `jsMath` software,¹ which can be linked to any website to provide proper typesetting of mathematics, as well as high-quality printing. This solves many of the problems that accompany other methods of displaying mathematics within webpages.

2 PDF files of full articles

The author has produced retro-born versions of all articles from a complete journal volume² as part of a feasibility trial to move from print-only to online access. Bearing in mind the considerations mentioned above, and with `pdf- \TeX` and Unicode support becoming more widely available, much care was taken to incorporate navigational aids that are available

¹ `jsMath` homepage: <http://www.math.union.edu/~dpvc/jsMath/>.

² Bulletin of the Australian Mathematical Society, Volume 72 (2005); freely available online at <http://www.austms.org.au/Bulletin>.

with PDF documents. Several new techniques were developed to make these documents as useful as possible to researchers. These are now described briefly.

- a.** *Copy and Paste of blocks of text which include mathematical symbols (see Figure 1).*

This is achieved by adding a character map (CMap) resource³ to each of the fonts that \TeX uses, for typesetting the text and the mathematics. Such resources do not affect the appearance of the typeset material, but associate each (perhaps accented) letter and mathematical symbol with its Unicode code-point. In this way symbols are given a unique identity which can be used for copy/paste to other applications, and for searching within the PDF itself. Currently the actual result of a copy/paste action may depend on the particular software being used; that is, the PDF-browser used to view the article, and the text-editor, or other software, into which the content is pasted.

The author has produced CMap resources for the following old-style \TeX font encodings: OT1, OML, OMS, OMX. Articles in the journal volumes¹ also used symbols from the AMS fonts MSAM and MSBM, Euler Fraktur fonts, and a few other characters, so CMap resources have been made for their ‘U’ (Unknown) encodings; namely files `umsa.cmap`, `umsb.cmap`, `ueuf.cmap`, `ueufb.cmap`, `ulasy.cmap`, and `upzd.cmap` (Zapf Dingbats) and `upsy.cmap` (Adobe Symbol). Resources have also been created for LY1 (Lucida) and LMR (Lucida Bright Math symbols) encodings.

In the case of the OML encoding, as used with the `cmmi` math-italic font family, the ordinary letters $A, B, \dots, a, b, \dots, z$ are associated with “math alphanumeric symbols” in Unicode Plane 1. Such symbols can be seen as the M, P, x and L within the **TextEdit** window in the middle image of Figure 1. However, bold symbols from `cmmib` fonts use the same OML encoding. Thus CMap resources have been constructed that are specific to the font face and style, rather than just to the encoding. Similarly files `omlmit.cmap` and `omlbit.cmap`, support the `cmmi` and `cmmib` font families respectively. Similarly there is OT1-encoded support for normal, italic, sans-serif (medium and bold) and typewriter alphabets used with mathematics.

A \LaTeX package, called `mmap.sty`, is now available at CTAN.⁴ This package provides these `.cmap` files and coding that causes the CMap resources to be included when the appropriate font is loaded for

³ See Adobe CMap and CID specifications at http://www.adobe.com/devnet/font/pdfs/5014.CIDFont_Spec.pdf.

⁴ ... in the directory location `.../tex-archive/macros/latex/contrib/mmap/`.

use with mathematics. Similar support for the full set of Euler fonts is planned, and other symbol fonts also can be supported.

- b.** *Images of figures and tables which pop up (see Figure 2) near the place in the text where the figure/table has been referenced.*

This feature allows figures and tables to be viewed without changing the PDF page that is displayed. It requires JavaScript⁵ (or ECMAScript) to be enabled within the PDF browser. If the toggled image pops up in a place that is inconvenient for further reading, then it can be shifted to elsewhere on the page. With further developments of the PDF specifications and browser software, the means to move the image could be redesigned to become more intuitive. (Indeed, it would be nice if browsers had a ‘cross-reference spy-glass’ feature, providing a small-sized view of a different part of the same PDF, in response to clicks on the cross-reference anchors.⁶)

In the event that JavaScript has been disabled in the PDF browser, so that the pop-up mechanism won’t work, the toggle button should not appear and the underlying cross-reference hyperlink should work as usual. Unfortunately, not all PDF browsers implement this properly.⁷ Even worse, `Xpdf` and `eVince`, prior to the v0.8.3 release (4 June 2008) of the Poppler library, would not even load documents built with `pdf \TeX` containing form fields; when built with more recent versions of the library, these now work as intended.

Implementing this pop-up feature was done by altering the expansion of the standard \LaTeX macros `\figure` and `\table`, thus changing the way that float contents are handled, and of `\ref` for building the toggle buttons. The first run of the \LaTeX job is largely unchanged, apart from recording the number of floats encountered as the expansion of a macro `\hasfloats` within the `.aux` file, followed by a macro `\testforfloats`. On the next \LaTeX run, as the `.aux` file is read `\testforfloats` is encountered, triggering code that reads the number of floats and causes extra packages to be loaded, such as `pdftricks.sty`, `insdljs.sty` for inserting JavaScript coding into a PDF, and `pdfpopup.sty` which has coding

⁵ See <http://en.wikipedia.org/wiki/JavaScript>.

⁶ In fact the Skim browser for Macintosh (Mac OS X) has such a feature, providing an extra window to be opened, focusing on that portion of a document surrounding the target of a cross-reference hyperlink. This tool could be improved by respecting any destination *view* that may have been specified within the PDF.

⁷ For example, Apple’s Preview browser, at least up to version 3.0.9, neither supports JavaScript, nor respects the button flags. Hence not only do pop-up images fail to work, but also the underlying hyperlink cannot be accessed.

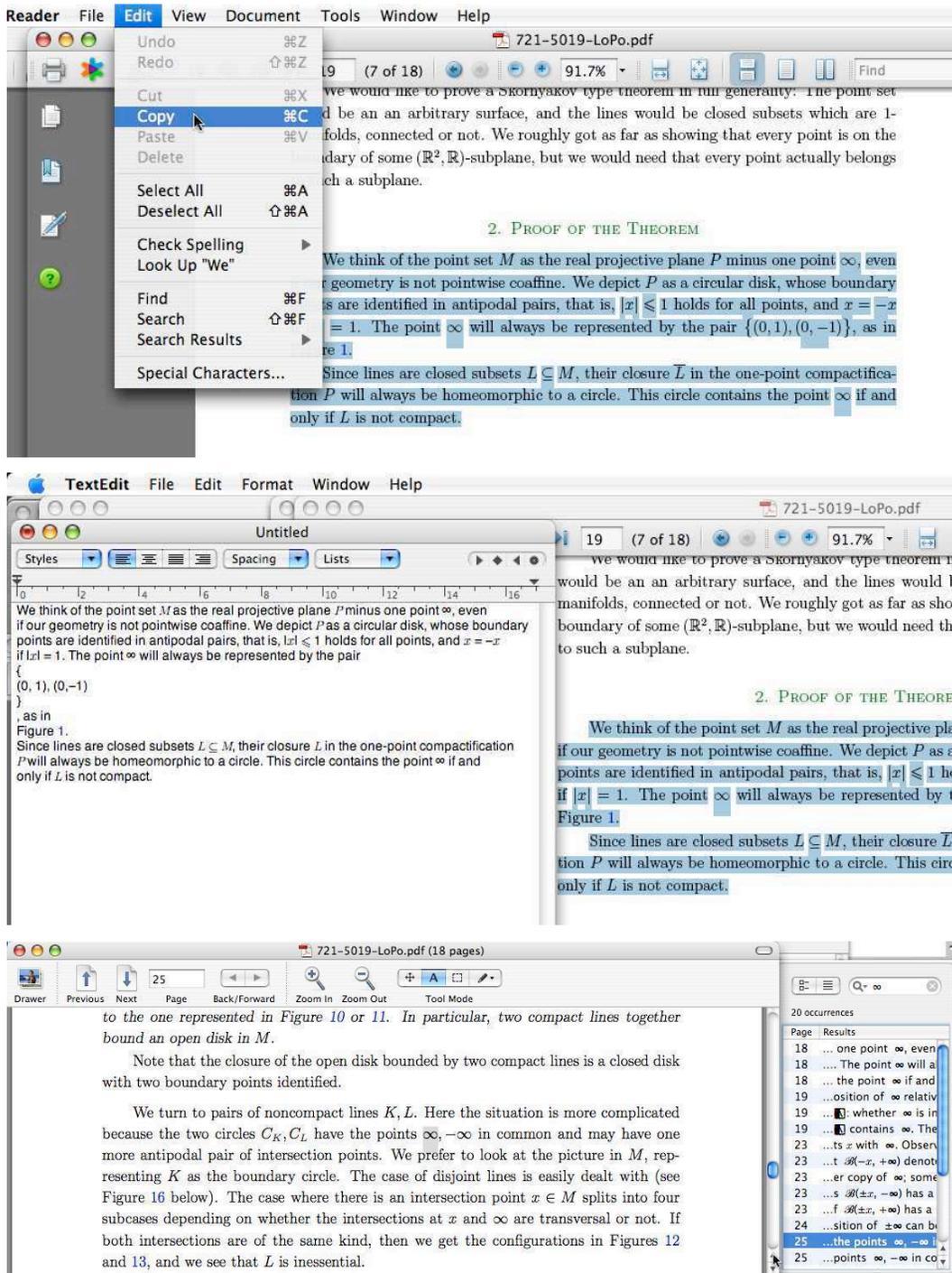


Figure 1: Copy/Paste of mathematical content: the middle image shows the result of pasting the contents that have been selected and copied, as shown in the upper image. The actual result of a copy/paste action may depend on the particular software being used; that is, the PDF-browser used to view the article. The lower image shows how searching for mathematical symbols is done, with suitably enabled PDF viewers.

Each of the two cases splits into subcases depending on the position of ∞ relative to \bar{L} : whether ∞ is in \bar{L} or not in \bar{L} , and in the latter case, which of the complementary components of \bar{L} contains ∞ . The resulting five possibilities are depicted in Figures 2 through 6.

The components of the complement of a line relative to M . Essential lines do not separate the point set M ; their complement is a punctured disk or an intact disk depending on whether the line is compact or not. An inessential

the pair formed by the real projective plane and one of its lines. In this case, the line L will be called *essential*. The complement $P \setminus \bar{L}$ is an open disk.

A similar procedure (needing extra corrections) may be applied in the case of two disjoint circles and results in a proof that (P, \bar{L}) is homeomorphic to the pair formed by the real projective plane with a conic. In this case, the line L will be called *inessential*. The complement of \bar{L} in P is a disjoint union of an open disk and an open Moebius strip in this case. Our first aim is to show that inessential lines do not occur (Proposition 5).

Each of the two cases splits into subcases depending on the position of ∞ relative to \bar{L} : whether ∞ is in \bar{L} or not in \bar{L} , and in the latter case, which of the complementary components of \bar{L} contains ∞ . The resulting five possibilities are depicted in Figures 2 through 6.

The figures also show the connected components of the complement of a line relative to M . Essential lines do not separate the point set M ; their complement is a punctured

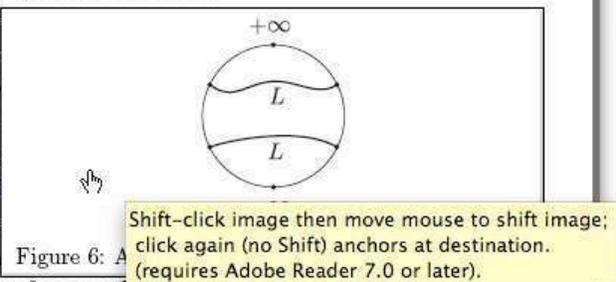


Figure 6: A compact inessential line of second type.

map, or $\pi^{-1}(L)$ is a single circle $C = -C$, and π restricted to C is the unique two-fold

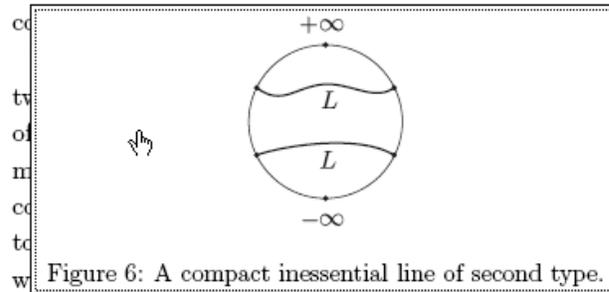


Figure 6: A compact inessential line of second type.

A similar procedure (needing extra corrections on the annulus defined by C and $-C$) may be applied in the case of two disjoint circles and results in a proof that (P, \bar{L}) is homeomorphic to the pair formed by the real projective plane with a conic. In this case, the line L will be called *inessential*. The complement of \bar{L} in P is a disjoint union of an open disk and an open Moebius strip in this case. Our first aim is to show that inessential lines do not occur (Proposition 5).

Each of the two cases splits into subcases depending on the position of ∞ relative to \bar{L} : whether ∞ is in \bar{L} or not in \bar{L} , and in the latter case, which of the complementary components of \bar{L} contains ∞ . The resulting five possibilities are depicted in Figures 2 through 6.

The figures also show the connected components of the complement of a line relative

Figure 2: Pop-up figures & tables: the images show how when the mouse hovers over a reference to a figure, it highlights a button to toggle showing and hiding a floating image. This image is moveable in case its natural position is inconvenient; e.g., obscuring where you wish to read.

for building buttons to show and hide the pop-up images.

Now when a float is encountered, the full contents of the environment is written out into a file, named according to the `figure` or `table` number, as part of a new \LaTeX job which includes a preamble loading most of the same packages as in the main job. This subsidiary job is run, using \LaTeX followed by `dvips/ps2eps/epstopdf` to get a single-page PDF with correct bounding box. The file is loaded back into the main job using `\pdfrefximage` within a `\setbox`, and its PDF object reference number is recorded for later use in placing the image on a JavaScript button field, when an appropriate `\ref` occurs. The object reference is also written into the `.aux` file as the expansion of a macro having its name derived from the figure number. This allows the reference to be used on the next run, with any `\ref` command that occurs before the float has been encountered, and also allows checking to see whether the reference number has changed, in which case a message is written to the `.log` file warning that another \LaTeX run is required. Finally, the tokens for the float environment are recovered and processed normally.

On subsequent \LaTeX runs, the image files do not need to be rebuilt, but are loaded from the PDF images created on an earlier run. If editing of the main document source changes the order of floats, the toggle buttons may become associated with the wrong images. Simply delete those images from the current directory; the correct ones should be generated afresh on the next \LaTeX run. In case the content of a float contains `\ref` and `\eqref` commands or citations, the subsidiary job that creates an image also loads a copy of the `.aux` file from the main job, which copy was made at the end of the previous run. It's possible that cross-references have not fully stabilised, so simply delete any affected image; after two more runs it will have been regenerated and included.

If browser software had a 'cross-reference spy-glass' feature (as suggested in the first paragraph of this item), then not only would there be no need for the PDF to contain JavaScript coding for the extra buttons, but also there would not be doubling-up of the information contained in figure and table floats. Furthermore, such a pop-up-like feature would 'just work' also for cross-references to section headings, numbered equations, etc., as well as to the floats, and perhaps also for citations and 'back-references' from the bibliography (see Figure 7, for example). This is surely the way that such a feature ought to be implemented; ideally it should not be nec-

essary for a scientific document to include explicit programming which controls how its content be displayed, but just have declarations of which browser-supplied functions are to be used. The implementation presented here is mainly to demonstrate the usefulness and practicality of such a 'pop-up' feature for cross-referenced material, so that browser vendors might be encouraged to incorporate a similar feature within their own publicly-available software. However, there are certainly other, simpler uses for pop-ups to show extra images that are not found elsewhere among the usual pages of a document.

c. Extended use of bookmarks (see Figure 3), . . .

Use of bookmarks is quite common for the major sections of a document; this is automatic when using `\usepackage{hyperref}` with a \LaTeX document. This is here extended further to creating bookmarks for definitions, Theorems, Lemmas, etc., and also for figures, tables, and some equation displays. To avoid the bookmark window becoming too cluttered, only those equations that have actually been cross-referenced within the document are given their own bookmark.⁸

Having such bookmarks means that there are named destinations with the PDF at all the important places for the structure and content of the document. Furthermore these names are available in a separate file, so potentially this can be used to construct hyperlinks directly to these important places. This could be extremely useful in the context of a digital archive.

For figures and tables, a meaningful string to be the textual anchor in the 'Outline' window is obtained as the first sentence in the caption. This is obtained by reading the caption from the `.lof` or `.lot` file and parsing to locate the first full stop ('.'). With Theorems, Lemmas, Propositions, etc., the anchor-text uses the appropriate numbering, as seen in Figure 3. The limiting of bookmarks to only those referenced is achieved by patching the `\@setref` internal macro to implement a 'memory' of referenced labels. A line is written into the `.aux` file; this defines a macro, with name derived from the label. The coding for placing equation numbers is patched so that `\df@tag` now also places an anchor, and a bookmark for this anchor when the memory indicates that the equation has been referenced — which is known on the 2nd and subsequent \LaTeX runs.

⁸ It can be argued that if an equation is not referenced then it doesn't need an equation number. However, many articles have been written where the author has not followed this maxim. For creation of bookmarks, this maxim has been programmed-in.

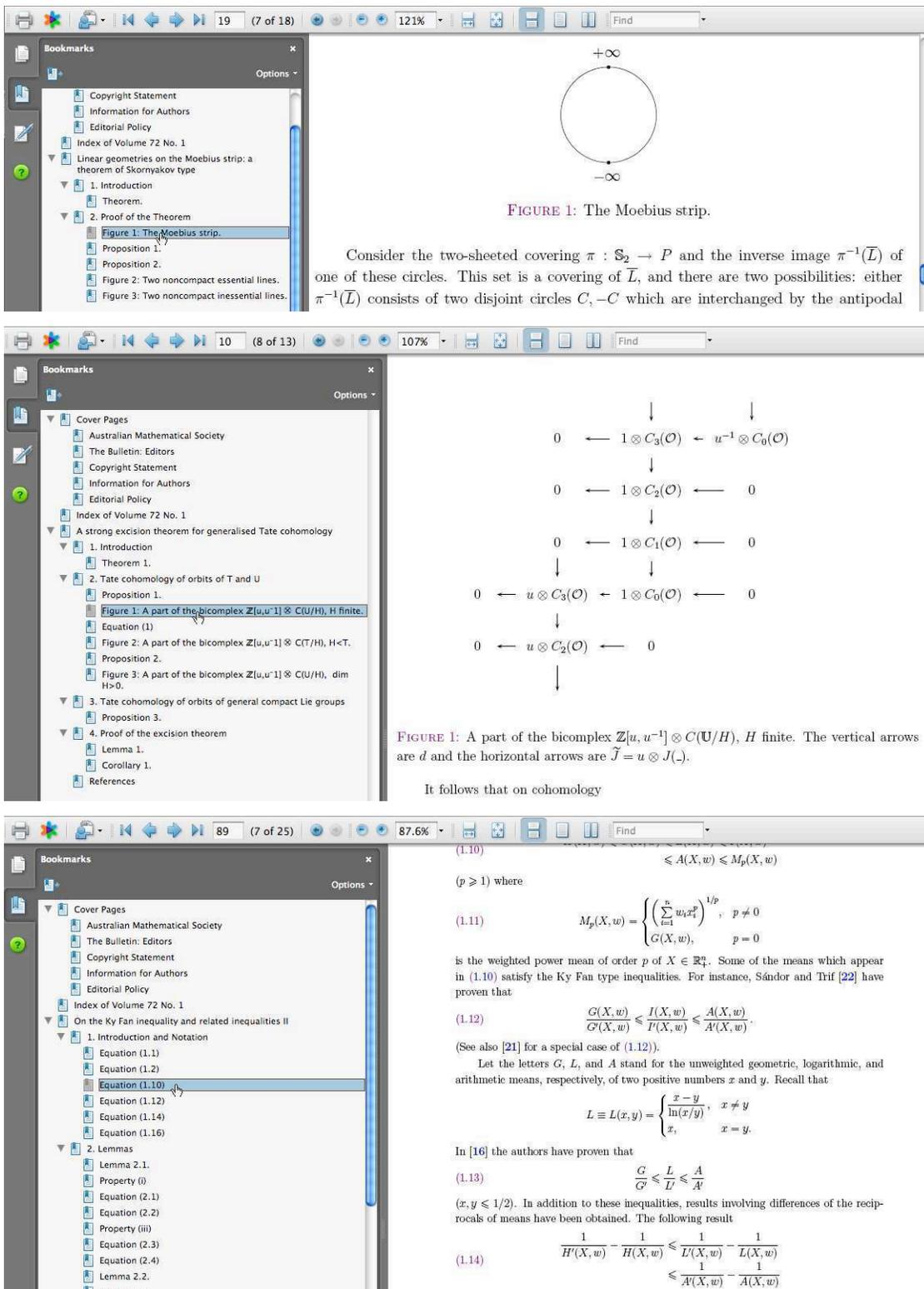


Figure 3: Bookmarks: these three images, taken from different articles, illustrate various aspects of the automatically created ‘bookmarks’. These provide easy access to the important parts of the document, including front-matter as well as sections of the mathematical article itself; such as definitions, theorems, proofs, remarks and figures. The middle image shows that mathematical symbols can be used within bookmarks, while from the lower one it can be seen that a bookmark is not produced for every numbered equation, but only for those that are cross-referenced within the article itself.

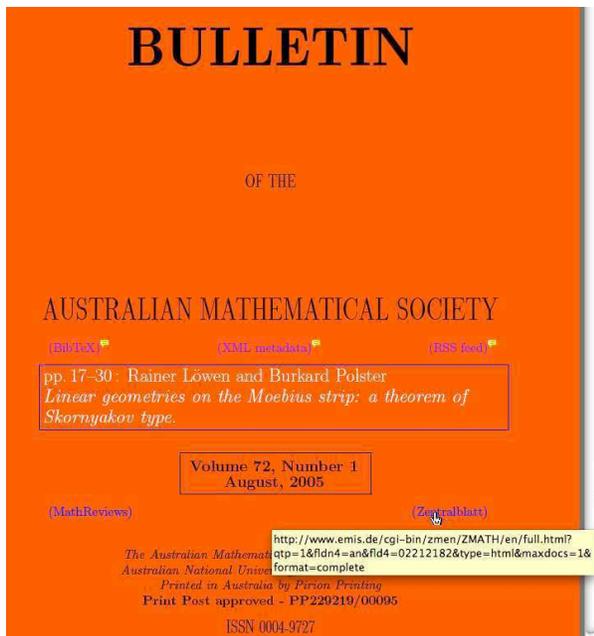


Figure 4: Cover page: this includes hyperlinks to recover the embedded metadata attachments, in various formats. Also there are links to the websites of the *Mathematical Reviews* and *Zentralblatt Math* reviewing agencies, where this article has been reviewed. The large button enclosing the title links directly to the start of the article proper.

d. ... including support for mathematical symbols and some super-/subscripts.

The identifying strings displayed in the bookmarks window allow for the full range of Unicode codepoints. This means that mathematical symbols can appear, which is most appropriate for figure captions, and mathematics is used with (sub-)section titles and theorem names, say. Even though the Unicode specification does not allow for complete alphabets of super-scripted and sub-scripted letters, an attempt is made to make good use of those that are available. Some further work is needed on this aspect of bookmarks.

e. Inclusion of attachments (see Figure 4), containing metadata for the article, in various useful formats.

The metadata for a scientific paper is important for various purposes, not least of which is citation within future works. By distributing a document with its own metadata, common difficulties can be avoided, such as the incorrect spelling of an author’s name, or wrong affiliation, etc., or just getting the page numbers wrong. These PDFs come with text-file attachments containing various amounts of

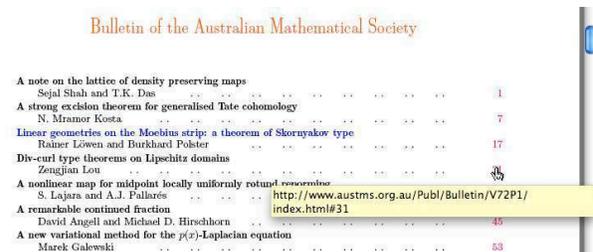


Figure 5: Index page: containing links to the Australian Mathematical Society’s website, for accessing each of the articles from the same journal issue.

metadata in useful formats: (i) a BibTeX entry suitable for adding to a .bib file; (ii) a short HTML-formatted entry, suitable for an RSS feed; (iii) an extensive metadata file in XML format, which includes the complete bibliography of the article, as well as its own publishing details.

Each of these metadata files is generated “on-the-fly” by suitable TeX macro coding, using the information provided for typesetting the PDF document itself. Thus, barring mistakes in the various format translations required for this, the metadata is guaranteed to be consistent with what appears in print.⁹ Access to these files is made easy by anchors on the cover page (see Figure 4) located above the white text of the article title and authors, which itself anchors a hyperlink to the start of the article proper.

When available, database codes for reviews of the article are included with the BibTeX and XML metadata files. In this case, the cover page has further anchors, for hyperlinks which give direct access to the review at Math Reviews¹⁰ and Zentralblatt-MATH.¹¹ The same typesetting run that produces the PDF also builds an HTML webpage for the article.¹² This presents all the metadata and has the same hyperlinks, and more. Some format translations are also required when building such webpages.

Part of the metadata for an article is the context in which it has been published; namely, the complete journal issue. For this, we have chosen to

⁹ This is a useful feature for freshly published articles, but is not really appropriate for a digital library or preprint archive, which would presumably have its own database of metadata already prepared and checked for the documents that it serves.

¹⁰ A subscription to MathSciNet is required to make use of this hyperlink.

¹¹ Unregistered users have reduced access to the features available at this site.

¹² <http://www.austms.org.au/Publ/Bulletin/V72P1/721-5019-LoPo/index.shtml>

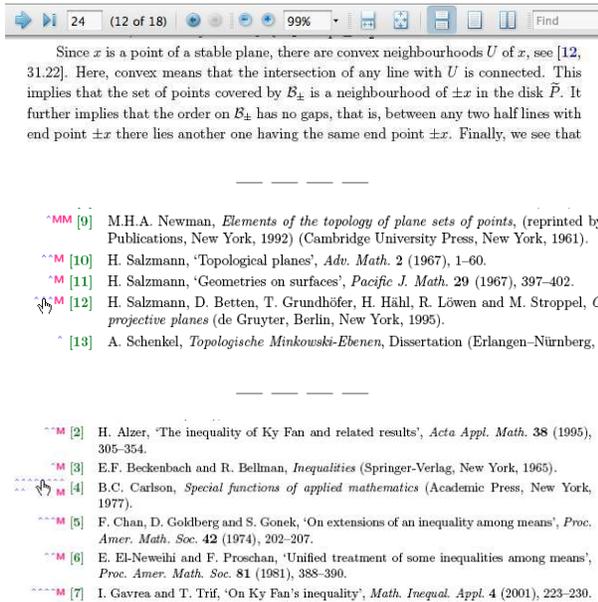


Figure 6: Back-reference hyperlinks: the middle image indicates how the caret (\wedge) in the left margin is the anchor for an active hyperlink, which jumps to the location where the particular reference has been cited. The upper image shows the resulting change of focus. As there can be several citations of the same item, the carets are right-aligned, with up to eight in a row, as in the lower image.

include the complete front-matter that would appear in the printed version of the journal. The images of Bookmarks in Figure 3 show the kind of material that is included: Editors, Copyright Statement, Information for Authors, etc. Of course the Index page (see Figure 5) lists all other articles appearing within the same issue. For each article there is an active hyperlink, using the page number as the visible anchor, that directs a web browser to the public page at the Australian Mathematical Society’s website where the article’s abstract can be read, and its metadata (including references) examined. Also, the name of the article itself is the anchor for another hyperlink to the start of the article proper.

f. Hyperlinking from the bibliography to the place within the text (i.e. back-referencing) where the citation occurred (see Figure 6), and to reviews at MathSciNet (see Figure 7).

Including back-references is not new, nor is having hyperlinks within the bibliography, when such are supplied by the article’s author. However, for articles where the original printed version did not have these features, there is the problem of how to include the extra information without upsetting the pagina-

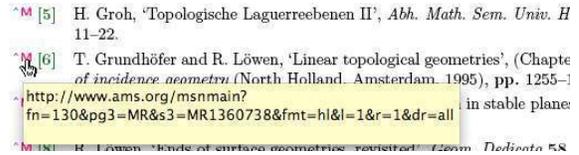


Figure 7: Hyperlinks to MathSciNet: the small raised M is the anchor for a hyperlink that connects to the American Mathematical Society’s website. It links to the review of this cited bibliography item. It is possible to have multiple such links, as can be seen in entry [9] in the middle image of Figure 6.

tion. Figure 7 shows an elegant solution that places the hyperlink anchors discreetly into the left margin. The caret-accent character has been chosen to be suggestive of an upward link; that is, to the material preceding the bibliography, which is the main body content.

The second problem, which is perhaps the more difficult one, is that of how to automate the collection of data required to build hyperlinks to reviewing services, such as MathSciNet and Zentralblatt-MATH, or other online archives of scientific material. For these PDFs the author made use of a small program called `bmref`, which is essentially a ‘batched’ version of MathSciNet’s `Mref` tool.¹³ Provided by Patrick Ion (Associate Editor of Mathematical Reviews, AMS), this short Perl program sends a carefully constructed XML file as a query to the MathSciNet database, as an HTTP ‘POST’ to `http://www.ams.org/batchmref`. The result returned is an XML file containing the same data, but with extra fields added for (i) the number of matches found, (ii) the reference numbers (MR-number) of found matches, and (iii) full bibliographic information, for each bibliographic item included in the original submission.

This allows the MR-numbers to be obtained for all items in the bibliography (well, all those that have been reviewed), with a single submission. Each MR-number is sufficient to build the desired hyperlink. Of course this is not 100% reliable, and some searching at MathSciNet can uncover MR-numbers for items that were not found in the automated search; but the bulk¹⁴ of the job is done automatically. T_EX coding was developed to analyse the author-supplied bibliography prior to constructing the XML file for use with `bmref`. Taking advantage of

¹³ See `http://www.ams.org/mathscinet-mref`.

¹⁴ Of 516 separate cited items from 51 papers, 67 were not found automatically; 19 of these were found with some manual searching. The remainder were to journals not covered at MathSciNet, or to unpublished theses, etc.

A note on the boundedness of Bergman-type operators on mixed norm spaces

Zengjian Lou
 Department of Mathematics
 Shantou University
 Shantou Guangdong 515063
 P. R. China
zjlou@stu.edu.cn

Abstract

We prove the boundedness of Bergman-type operators on mixed norm spaces $L^{p,q}(\varphi)$ for $0 < q < 1$ and $0 < p \leq \infty$ of functions on the unit ball of C^n with an application to Gleason's problem.

[Download the article in PDF format](#) (size 91 Kb)

Australian Mathematical Publishing Association Inc. © [Australian MS](#)



AustMS
AUSTRALIAN MATHEMATICAL SOCIETY

Member login
[AustMS Home](#) | [ANZIAM](#) | [Contact us](#)

[About us](#)
[Membership](#)
[Publications](#)
[Resources](#)
[Careers](#)
[News](#)
[Events](#)

The block structure of complete lattice ordered effect algebras

Gejza Jenča
 Department of Mathematics
 Faculty of Electrical Engineering and Information Technology
 Ilkovičova 3
 812 19 Bratislava
 Slovakia
gejza.jenca@stuba.sk

Received 4 May 2005; revised 28 June 2006
 Communicated by M. Jackson

This research is supported by grant VEGA G-1/3025/06 of MŠSR. This work was supported by the Slovak Research and Development Agency under the contract No. APVV-0071-06.

Abstract

We prove that every for every complete lattice-ordered effect algebra E there exists an orthomodular lattice $O(E)$ and a surjective full morphism $\phi_E : O(E) \rightarrow E$ which preserves blocks in both directions: the (pre)image of a block is always a block. Moreover, there is a 0,1-lattice embedding $\phi_E^* : E \rightarrow O(E)$

Figure 8: jsMath in webpages: the image at right illustrates the high quality and proper scaling of mathematical expressions within a webpage that uses the jsMath applet software. This contrasts starkly with the image at left, displaying poorer quality and lack of scalability in the static images, used with older web technologies.

TeX's `\write18` feature to run external commands and await the reply, the whole process can be fully automated and integrated with the typesetting run: craft the XML file, send to MathSciNet, analyse the result, extract the MR-numbers, then make these available for creation of hyperlinks. After a successful result of one such run, there is no need for the same tasks to be repeated on subsequent typesetting runs.

The bibliographic information returned can be requested to be in any of the usual formats: `bibtex`, `amsrefs`, `TeX`, `html` or as an HTML hyperlink. Thus this information could be used to check the bibliographic details provided by the author, or could even replace it altogether. This was not done with these tests, due to the desire to have the online content be the same as what was printed; however, if any factual errors were noticed (such as incorrect Volume or page numbers) then these were corrected.

In the hope of finding capabilities comparable to `bmref`, the author also searched Zentralblatt-MATH and other journal archives, for the availability of online tools for batched searches — unfortunately without success. Searches for a single article could be automated, but with fuzzy-matching this would result in multiple hits, requiring significant extra processing to determine whether the sought-after article actually had been located at that site.

In the context of a digital library or preprint archive, there may well be a large database of meta-

data readily available which could be easily searched instead. Indeed then it might be possible to include links not just to reviews of an article, but to the article itself. Some appropriate icon or symbol would then be used to indicate the kind of information to be found at the target of the hyperlink.

3 HTML pages for abstracts, etc.

At the end of 2007, the Australian Mathematical Society made a complete change in the hosting arrangements of its website,¹⁵ as well as a change in the publication arrangements for its journals. There were still several journal issues¹⁶ that were not covered by the new arrangements, for which PDFs were still to be served from the Society's site. This necessitated the need for abstract pages which were publicly available, having hyperlinks to the PDFs which are accessible only with a subscription. The job of creating these pages fell to the current author, in the rôle of web-editor for the Society.

The same techniques that were used for the Bulletin PDFs were used for this task, only now there was no full article PDF being produced. Thus the principal outputs were the HTML pages and metadata files. MR-numbers were obtained in the same

¹⁵ Australian Mathematical Society's website: <http://www.austms.org.au/>.

¹⁶ *J. Austral. Math. Soc.*, Vol. 83 (1) & (2): <http://www.austms.org.au/Publ/JAustMS/>, and ANZIAM Journal, Vol. 49 (1) & (2): <http://www.austms.org.au/Publ/ANZIAM/>.

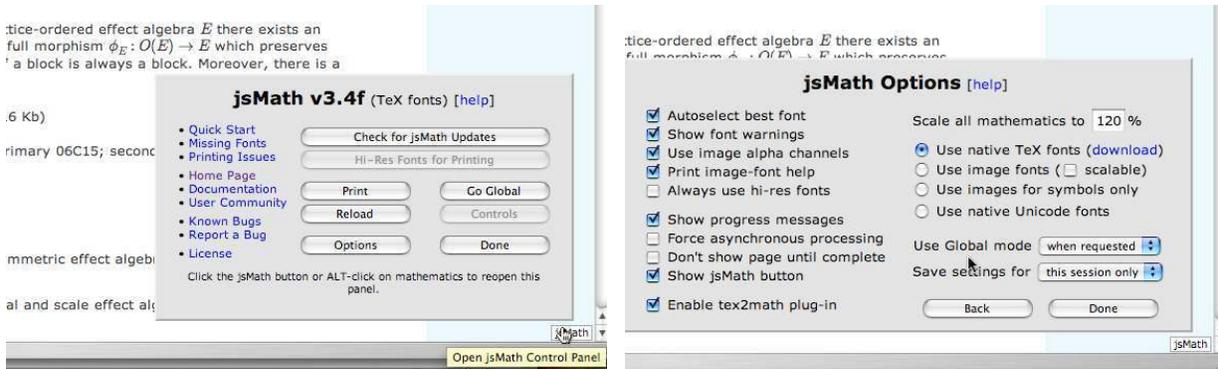


Figure 9: Help, Options & fonts with jsMath: the image at right shows how to bring up the jsMath control panel, displaying the ready availability of options and help features. In the Options panel, shown at right, one sees the great flexibility in the way different fonts or images may be used to construct the mathematics. Best quality is obtained by downloading and installing the jsMath fonts, or using local Unicode fonts.

way as described above, only now the reference data was presented in a different way, as L^AT_EX-formatted .bb1 files. There were other differences in the metadata too, requiring some minor adaptations of coding used previously.

A significant addition however, was the choice to use jsMath for the mathematical expressions that appeared in titles, abstracts and occasionally within the names of cited papers. Davide Cervone’s jsMath software¹ is a JavaScript applet that is effectively a cut-down version of a T_EX compiler that does proper typesetting of stand-alone¹⁷ mathematical expressions. The author had used this before with the ≈ 5000 abstract submissions for the ICIAM07¹⁸ Congress, and had made several suggestions for bug-fixes and other improvements that Cervone willingly implemented. As well as providing a better quality presentation of mathematics within webpages, jsMath also gives proper printing using fonts rather than images, and solves the problem of rescaling the mathematics to suit a web-surfer’s choice of font size. Figure 8 displays this, by giving a comparison with a page produced using older methods.

A lot of help is readily available, as Figure 9 shows. The jsMath ‘Options’ panel allows various choices of fonts to use with the mathematics being shown. Specially prepared T_EX fonts can be downloaded. When installed in the local OS, these can be used with pages from any website that employs jsMath. This not only gives the best possible quality of image (since it leverages font-rendering machinery on the local operating system), but also speeds up

processing since less information needs to be downloaded from that site. Alternatively, a local Unicode font could be used for similar speed gains, but the resulting layout of complicated mathematical expressions might not be as finely tuned as with the jsMath T_EX fonts. Now Copy/Paste and searching refer to the Unicode code-points for mathematical symbols, whereas otherwise these operations would use the position in traditional T_EX encodings.

Conclusion

Here we have described several advanced features applicable in particular to the electronic publication and presentation of mathematical papers, but which have much wider utility. The emphasis is more on the nature of these features rather than on any specific means of implementation, since various forms of implementation can be possible appropriate to the particular circumstances of distribution and production. The existence of example documents² is “proof of concept” that these are achievable with current T_EX software. Indeed these can serve as a test bed for some of the features where there are inconsistent levels of support within current browsers; e.g., Copy/Paste and searching with regard to mathematical symbols, handling of overlaid annotations, spy-glass views, etc.

Hopefully, in the near future, some of these ideas will become standard practice, with consistent support across browser software, for the benefit of academics and researchers in mathematics and of the scientific community generally.

¹⁷ It doesn’t do full pages, nor handle counters, cross-referencing, citations, etc.

¹⁸ Browse at the ICIAM07 timetable: <http://www.iciam07.ethz.ch/timetable/>.

Multidimensional text

John Plaice

School of Computer Science and Engineering
The University of New South Wales
UNSW SYDNEY NSW 2052, Australia
plaice (at) cse dot unsw dot edu dot au

Blanca Mancilla

School of Computer Science and Engineering
The University of New South Wales
UNSW SYDNEY NSW 2052, Australia
mancilla (at) cse dot unsw dot edu dot au

Chris Rowley

Mathematics and Statistics
The Open University in London
1-11 Hawley Crescent, Camden Town
London NW1 8NP, UK
c.a.rowley (at) open dot ac dot uk

Abstract

The standard model of text, based on XML and Unicode, assumes that documents are trees whose leaves are sequences of encoded characters. This model is too restrictive, making unnecessary assumptions about documents, text and the processing applied to these.

We propose instead that text and documents be encoded as tuples, i.e., sets of dimension-value pairs. Those dimensions used for content are split into the property dimensions, which are named by elements of an unstructured set, and the indexing dimensions, which form a structured set, often enumerated.

Using our approach allows natural solutions for a wide range of encoding requirements: encoding of documents at multiple levels of abstraction (glyph, character, word, stem-declension pair, compound word, etc.); encoding by linear, tree, DAG and multidimensional structures. Our model is upwardly compatible with existing approaches.

1 Introduction

In this article, we present a new model for manipulating documents in which every structure is encoded as a *tuple*, a set of dimension-value pairs. The simpler elements are ordinary tuples encoding basic information, while more complex elements encode mappings from structured index sets towards simpler elements.

The advantage of this new model is that it allows documents to be encoded in many different ways, taking into account logical structure, visual structure and linguistic analysis. Furthermore, the proposed model is upwardly compatible with existing practice.

This model is one result of a research project

into the nature of text initiated by authors Plaice and Rowley [5, 6]. The current standard computer model of documents assumes that the structure of a text is a tree — normally encoded using XML — whose leaves are sequences of Unicode characters and where the intermediate nodes contain sets of attribute-value pairs to define properties. The conclusion of the aforementioned research was that the current model makes the assumption that text is simply something to be shuffled around, possibly chopped up for rendering purposes, but that it has no structure of its own; furthermore, the origin of the view of a document as some form of stream of bytes can be traced directly back to the near-simultaneous invention of the typewriter and the

telegraph. The relatively recent distinction between “character” and “glyph”, where character is an abstraction from glyph, still makes the assumption that the visual presentation of text is the key, despite the fact that, for example, English, Amharic and Chinese are normally encoded at, respectively, the letter, syllable and (sub)word levels.

As large collections of documents are brought together, so that they can be searched, it is clear that this “standard” model leaves something to be desired. Searching through a multilingual database of texts requires substantial linguistic support, and here it becomes essential that the different languages be handled at similar levels. Moreover, in linguistics, texts are often encoded using parse trees and attribute-valued matrices, often DAGs (directed acyclic graphs) or digraphs (directed graphs).

As for the structure of documents containing text, the tree structure itself is not always appropriate. Although it is true that a two-dimensional table can be encoded as a tree, this is only true by imposing ordering on its two dimensions (either by rows or by columns); they are not naturally encoded by a purely hierarchical structure but by its antithesis, a completely crossed structure. Moreover, two-dimensional tables are often used to visualize multidimensional data, whose encoding truly requires a multidimensional data structure such as those investigated in a different context by one of the authors [1], not just to capture and simplify the semantics, but to ensure an efficient and tractable storage mechanism.

Importantly, what distinguishes the electronic document from all previous forms of document is that it is recreated every time that it is read, listened to, studied or processed. As a result, with a slight change of parameters, it can be recreated differently from any previous occasion. The standard model completely breaks down for these kinds of situations: it must resort to programming the document and, in so doing, loses the possibility of having the document being properly indexed for searchability. (See [4] for further discussion.)

The tuple structure that we are proposing allows us, as shall be shown below, to define a number of different structures, including ordered streams and trees, DAGs and multidimensional structures. For example, a sentence in a document can be encoded as a sequence of characters, as a parse tree with words as leaves, or as a “feature structure” from linguistics known as an attribute-valued matrix. Indeed, it could easily be encoded as all of the above, with appropriate link structures connecting the components.

The structure of this paper is as follows. We present a brief analysis of, and show the current limitations of, various existing models for the encoding of texts and documents in §2. We follow with a presentation of our new model (§3) using an extended example. We then (§4) describe some features of its use to declined word-stem sequences, parse trees, hierarchical document structure, attribute-value matrices and tables. We conclude with a discussion of future work.

2 Existing models

We examine in this section three approaches to dealing with text. Although not exhaustive, it does provide us with indications of where a more complete model should be heading.

2.1 XML documents

An XML document is typically encoded as a tree. In some sense, this is all there is to say, but a proper understanding of XML requires examining not just the obvious tree structure, but also the structures of the nodes and the leaves in these trees.

For each element in an XML document, there is a possibly empty list of attribute-value pairs, and a possibly empty list of child elements. The leaves of the tree consist of PCDATA (Parsed Character Data) or CDATA (Character Data), in both cases sequences of characters from the Unicode or some other character set, with PCDATA being parsed by the XML parser.

Therefore XML, often presented as a simple encoding, actually requires four data structures to describe a document:

- the tree;
- the list of elements;
- the attribute-value list; and
- the character sequence.

In addition, there are arbitrary restrictions on attributes which limit their usability: attributes cannot contain multiple values, nor can they contain tree structures.

Our model includes such XML tree-based documents but it can also handle non-hierarchical structures that do not necessarily have any natural XML encoding.

2.2 Linguistics AVMs

In linguistics, it is common to model written language using structures consisting of a tree and an associated attribute-value structure [3, p. 16]. For example, here is the parse tree for the sentence “Mary seems to sleep.”:

```
[ S1: [ NP2: Mary
        VP1: [ V1: seems
                VP3: [ Aux3: to
                        VP3: [ V3: sleep ]]]]]
```

The associated structure, called an attribute-value matrix (AVM), is presented below.

Note: The subscripts in the parse tree correspond to the nodes in the AVM. Node 2 appears twice in the AVM, being the subject of both node 1 and node 3.

$$\begin{array}{c}
 \boxed{1} \left[\begin{array}{c}
 \text{subj} \quad \boxed{2} \left[\begin{array}{c}
 \text{agr} \quad \left[\begin{array}{c}
 \text{pers} \quad 3\text{rd} \\
 \text{num} \quad \text{sg}
 \end{array}
 \right] \\
 \text{pred} \quad \text{mary}
 \end{array}
 \right] \\
 \text{comp} \quad \boxed{3} \left[\begin{array}{c}
 \text{subj} \quad \boxed{2} \\
 \text{pred} \quad \text{sleep} \\
 \text{tense} \quad \text{none}
 \end{array}
 \right] \\
 \text{pred} \quad \text{seem} \\
 \text{tense} \quad \text{pres}
 \end{array}
 \right]
 \end{array}$$

Because of such shared structures, AVMs are often considered to be DAGs. However, it is possible to have cyclical structures in an AVM: Consider the noun phrase “the man that I saw”, whose parse tree is here:

```
[ NP1: [ Det: the
        N': [ N: man
              S'2: [ Comp: that
                    S: [ NP: I
                        VP: [ V: saw ]]]]]]]
```

In the following AVM, a cyclical structure is needed to describe the “filler-gap” dependency in the relative clause [3, p. 19]:

$$\boxed{1} \left[\begin{array}{c}
 \text{def} \quad + \\
 \text{pred} \quad \text{man} \\
 \text{comp} \quad \boxed{2} \left[\begin{array}{c}
 \text{pred} \quad \text{saw} \\
 \text{subj} \quad \left[\begin{array}{c}
 \text{pred} \quad \text{pro} \\
 \text{obj} \quad \boxed{1}
 \end{array}
 \right]
 \end{array}
 \right]
 \end{array}
 \right]$$

Our model naturally encodes both such parse trees and these AVMs.

2.3 Tables

Xinxin Wang and Derick Wood [7] developed a general model for tables, in which a table is a mapping from a multidimensional coordinate set to sets of contiguous cells. For them, the two-dimensional format commonly used to present a table is not the internal format. Here is an example of the use of a 3-dimensional coordinate set from their paper:

	Assignments			Examinations		Grade
	Ass1	Ass2	Ass3	Midterm	Final	
1991						
Winter	85	80		60	75	75
Spring	80	65	75	60	70	70
Fall	80	85		55	80	75
1992						
Winter	85	80	70		75	75
Spring	80	80			75	75
Fall	75	70	65	60	80	70

It appears that, in 1991 alone, Assignment 3 was identical across the three terms but for 1992, we can see no such simple explanation of the larger box since it seems to amalgamate this assignment with an examination.

In this example and using their notation, the three dimensions and their value sets are as follows:

$$\begin{array}{l}
 \text{Year} = \{1991, 1992\} \\
 \text{Term} = \{\text{Winter}, \text{Spring}, \text{Fall}\} \\
 \text{Marks} = \left\{ \begin{array}{l}
 \text{Assignments} \cdot \text{Ass1}, \\
 \text{Assignments} \cdot \text{Ass2}, \\
 \text{Assignments} \cdot \text{Ass3}, \\
 \text{Examinations} \cdot \text{Midterm}, \\
 \text{Examinations} \cdot \text{Final}, \\
 \text{Grade}
 \end{array} \right\}
 \end{array}$$

A small change of syntax would transform this into an example of our model.

3 The new model

In this section we present an extended descriptive illustration of the model, using an example, rather than a detailed formal model.

There is only one basic structure in this model: the *tuple*, which is defined as a set of *dimension-value* pairs. This is not a new data structure, and it has many different names in different formalisms: *dictionary* in PostScript, *hash-array* in Perl, *map* in C++, *association list* in Haskell, *attribute-value list* in XML, and *tuple* in Standard ML and Linda.

We begin by proposing a possible encoding for the sentence “L^AT_EX 2_ε is neat.”

```
[ type: sentence
  numberWord: 3
  endPunctuation: [ type: unichar, code: 002E ]
  0: [ type: TeXlogo
      TeXcode: "\LaTeX\thinspace2\loweript
                \hbox{\small$\varepsilon$}]"
      SimpleForm: [type: digiletters
                   unicharstring: "LaTeX2e" ]]
  1: [ type: word
      numberChar: 2
      0: [ type: unichar, code: 0069 ]
```

```

1: [ type: unichar, code: 0073 ] ]
2: [ type: word
    numberChar: 4
    2: [ type: unichar, code: 0061 ]
    1: [ type: unichar, code: 0065 ]
    0: [ type: unichar, code: 006E ]
    3: [ type: unichar, code: 0074 ] ] ]

```

In this example, there are 12 tuples:

- 1 sentence,
- 1 T_EX logo,
- 1 character string consisting of a “word and digits”,
- 2 words, and
- 7 Unicode characters.

The different dimensions in the example play different rôles. In fact, there are three categories of dimensions:

- The *typing dimensions* allow one to distinguish the different kinds of tuple. Thus all tuples must have a typing dimension. In the example, a special dimension, called **type**, provides the information. This is the standard solution, although we do not exclude the use of further special dimensions, say **subtype** or **version**, for further clarification.
- The *property dimensions* are used to store any type of information about the tuple. In the example, these dimensions are:
 - **numberWord** and **endPunctuation**, for tuples of type **sentence**;
 - **TeXcode** and **SimpleForm**, for the T_EXlogo tuples;
 - **unicharstring** for **digiletters** tuples;¹
 - **numberChar** for **word** tuples;
 - **code** for **unichar** tuples.
- The *indexing dimensions* are used to access the substructures of a tuple (the content of the tuple) via an indexing mechanism or structure. The set of all the indexing dimensions available to a given type of tuple can be very large, conceptually infinite, and will often carry a complex structure.

In the following sections we will extensively develop examples of how the structure of these indexing dimensions can be used to encode complex systems. In the example, both the **sentence** and **word** tuples use the natural numbers (\mathbb{N}) to enumerate their content; thus the indexing dimensions available are the natural

¹ This could be replaced by a more formal tuple of type **digisymbol** that is like **word**: being an indexed collection of **unichar** tuples that can also contain digits.

numbers, whose structure is the unique countable well-ordering.

More precisely, the property dimensions form an unstructured set, whilst the indexing dimensions form a subset of a structured set. We can, for example, write a tuple of type **typespec** that defines the sets needed for the dimensions in **sentence** tuples:

```

[ type:      typespec
  tupletype: sentence
  typedim:   {type}
  proppdim:  {numberWord, endPunctuation}
  indexdim:  N
]

```

Note that, although the set of possible index dimensions \mathbb{N} is the infinite set of all natural numbers, any given tuple will use only a finite number of numbers as indices. Also, it is important that, for example, the order of the characters in a word is defined by the ordering of their indexing dimensions as natural numbers, not the order in which they appear in the written form of the “word tuple”. In this example we write the characters in the order of the numbers of their Unicode slots (such an ordering may be very efficient for certain applications).

In all of these sets, of dimensions and possible values, both equality and membership must be computable. In all usable examples, of course, equality-testing and other necessary set operations should be at worst of polynomial time complexity for reasonably sized sets.

The tuples used as values may be of any type, thus, for example, they could include strings, files, programs, and so on.

There can be other interesting interplays between dimensions and values. Consider, for example, the Unicode character tuple:

```
[ type: unichar, code: 002E ]
```

Here the value 002E is a value used to index the Unicode character database and, as Bella [2] has shown, that database can be understood as a single tuple indexed by the natural numbers whose entries are themselves tuples containing various kinds of information about each Unicode character.

The tuples used in this model are *conceptual*: they can be implemented — both as data structures in running programs and as sequentialised files on disk — in many different ways. Depending on the exact applications, algorithms and programming languages and environments, some solutions are more appropriate than others.

4 Examples

We hope it is now clear that our tuple structure

can encode any sort of simple record or entry. In this section, we indicate with a few more examples how the tuple structure can encode different kinds of data structure.

As was described in the previous section, there are typing, property and indexing dimensions in a tuple. If we restrict ourselves to the indexing dimensions, then the tuple can be considered to be a (partial) function from an index set (the structured set of indexing dimensions) to a set of values, which are in general themselves tuples, interpreting a singleton value as a type of tuple `singleton` and indexing set of size 1 `{0}`. If, in the sense of datatypes, these singleton values are strongly typed, then these `singleton` values will need a `datatype` property dimension.

- *Declined word-stem sequences*: To enhance document search or to effect grammatical analysis, it is common to *stem* words, separating the stem and the prefixes or suffixes of the words. We would then end up with entries such as:

```
[ type:      verb
  language:  English
  stem:      carry
  mood:      indicative
  tense:     present perfect
  voice:     active
  person:    3rd
  number:    singular
  unicharstring: "has carried" ]
```

or

```
[ type:      noun
  language:  French
  stem:      pomme
  gender:    feminine
  number:    plural
  unicharstring: "pommes" ]
```

Should the system not be able to parse such a word/phrase, then it will store only its `unicharstring` until it is appropriately updated.

- *Parse trees*: The result of the natural language parsing of a sentence is a richer text structure that is often encoded in a tree structure.

Below is a possible parse tree for the sentence “Mary seems to sleep.”, first presented in the section on AVMs (§2):

```
[ type: sentence
  NP:  Mary
  VP:  [ type: verbPhrase
        V:  seems
        VP: [ type: verbPhrase
              Aux: to
              VP: [ type: verb
                    V:  sleep ]]]]
```

- *Hierarchical document structure*: The traditional book with frontmatter, chapters, sections and subsections is a typical example of a document tree. This hierarchy can be extended downwards to paragraphs, sentences, phrases, words and characters.
- *Attribute-value matrices*: As explained in §2, these contain shared structures. Below is the AVM for the last example sentence.

```
[ type: AVMentries
  numberEntries: 3
  1: [ type: AVM
      subj: 2
      comp: 3
      pred: seem
      tense: pres ]
  2: [ type: AVM
      agr: [ type: AVM
            pers: 3rd
            num: sg ]
      pred: mary ]
  3: [ type: AVM
      subj: 2
      pred: sleep
      tense: none ] ]
```

- *Tables*: The encoding of tables by Wang and Wood is our final example of the model. Here is a possible encoding, where the dimensions are allowed to range over a set of possible values in order to encode the boxed values.

```
[ type: WWtable
  [ type: WWindex
    year: 1991
    term: Winter
    mark: Assignments.Ass1
  ] : 85
  ...
  [ type: WWindex
    year: 1991
    term: Winter..Fall
    mark: Assignments.Ass3
  ] : 75
  ...
  [ type: WWindex
    year: 1992
    term: Winter..Spring
    mark: Assignments.Ass3..Midterm
  ] : 70
  ...
  [ type: WWindex
    year: 1992
    term: Fall
    mark: Grade
  ] : 70
]
```

5 Conclusions

The model introduced in this paper, based on general tuples, is as simple as possible whilst being flexible enough to encode a large range of different approaches to the study and manipulation of text in all of its forms, as well as to support the encoding and linguistic tools such as language dictionaries.

However, the real power of the model is the idea of the index, built right into the model, which provides access to any piece of data, thereby supporting the specification of algorithms and hypertext-like links between document components.

The infinite index sets correspond to iterators into containers, as used, for example, in the C++ STL (Standard Template Library) for generic programming. Furthermore, using these sets it is even possible that certain tuples are, conceptually, countable infinite (like lists in a functional programming language), with the components being evaluated in a lazy manner, on-demand.

In future papers we shall show how this unifying data model makes it easy to combine in a single system myriad ways of editing, storing, manipulating and presenting text and to manipulate all of these together.

References

- [1] R. A. Bailey, Cheryl E. Praeger, C. A. Rowley, and T. P. Speed. Generalized wreath products of permutation groups. *Proc. Lond. Math. Soc.*, s3-47(1):69–82, July 1983.
- [2] Gábor Bella. *Modélisation du texte numérique multilingue: vers des modèles généraux et extensibles fondés sur le concept de textème*. PhD thesis, Télécom Bretagne, Brest, France, 2008.
- [3] Mark Johnson. *Attribute-Value Logic and the Theory of Grammar*. Center for the Study of Language and Information, Stanford University, 1988.
- [4] Blanca Mancilla and John Plaice. Possible worlds versioning. *Mathematics in Computer Science*, 2008. In press.
- [5] John Plaice and Chris Rowley. Characters are not simply names, nor documents trees. In *Glyph and Typesetting Workshop*, East Asian Center for Informatics in Humanities, Kyoto University, 2003. <http://coe21.zinbun.kyoto-u.ac.jp/papers/ws-type-2003/009-plaice.pdf>.
- [6] Chris Rowley and John Plaice. New directions in document formatting: What is text? In *Glyph and Typesetting Workshop*, East Asian Center for Informatics in Humanities, Kyoto University, 2003. <http://coe21.zinbun.kyoto-u.ac.jp/papers/ws-type-2003/001-rowley.pdf>.
- [7] Xinxin Wang and Derick Wood. A conceptual model for tables. In Ethan V. Munson, Charles K. Nicholas, and Derick Wood, editors, *PODDP*, volume 1481 of *Lecture Notes in Computer Science*, pages 10–23. Springer, 1998.

Data mining: Role of T_EX files

Manjusha Susheel Joshi

Bhaskaracharya Institute of Mathematics
Pune 411004, India
manjusha dot joshi (at) gmail dot com
<http://www.bprim.org>

1 Background

In a recent data mining project, I was trying to understand how to extract important words from a document. I realised that while writing a document, an author usually emphasizes important words by using italics, bold face, underlining, or quotes. This is a general observation for electronic documents.

While working on the project to try to find out appropriate information from the document set, I was looking for a better file format. In this regard, the T_EX file format attracted my attention.

Nowadays most research journals accept T_EX files from authors. Journals then put pdf files of the articles or abstracts on the web site, or submit them for printing.

$$\text{Authors} \xRightarrow{\text{T}_{\text{E}}\text{X}} \text{research journals} \xRightarrow{\text{PDF}} \text{User}$$

At present, the research journals typically classify papers by the year of publication or volume of the issue in which a given paper is published.

Almost always, users have access only to PDF files, and the journals do not publish the T_EX sources of the articles in any way.

2 Situation

Journals mostly have their own style files that take care of their abstract formatting, section heading style, headers, footers, and so on. They generally support keywords, citation, index, content, etc. All these features make the T_EX file a very special document — special in the sense that one can extract ‘feature words’ from the document relatively easily.

Specifically, words in the index, abstract, section headings, and emphasized words in the document body are words which we can call *feature words* of the document. So for such T_EX files, these feature words can be extracted, and submitted along with the T_EX file to the journal’s website.

Now, suppose we have available a collection of such T_EX files for a year. Then from all the feature words associated with the T_EX files, a program can collect feature words, understand which word is from which group and make groups of these documents based on clustering techniques (http://en.wikipedia.org/wiki/Data_clustering).

When users access the particular year of the journal, they can also see the overall topics easily, and a large set of keywords to help navigate through the articles. Even though authors provide keywords now, they usually merely highlight the topic or main theme of the article. Here we are considering more feature words from the document.

If a user asks for some particular keyword, since all the articles are already grouped according to their topics, a search program can show the user corresponding articles, by looking at the feature word data. Thus, searches can be faster and better when T_EX documents are available.

$$\text{User} \xRightarrow{\text{Query}} \text{Journal website} \implies \text{Specific paper}$$

3 Why T_EX files and not PDF in general?

1. PDF files are rather heavy in size, while T_EX files are light.
2. One can collect feature words when the file is in T_EX format. T_EX files are plain text, so rules to process them are fairly easy to design. For instance, we can find boldface words in the T_EX file with the rule ‘Search for the pattern ‘`{\bf`’ and save words until the matching `}`. Another example: ‘Ignore words starting with `\`’. Ultimately we can collect the actual content of the document. Once collection is over, we do not need to process the T_EX file again.
3. Text extracted from PDF files often doesn’t understand ‘fi’ or ‘ff’ ligatures properly; moreover, Greek letters α, β, \dots are not understood by present text extractors.
4. For figures, text extractors typically find ASCII codes instead of text; many times we have observed garbage when a figure is present in the text file.
5. Submission of the feature word file along with the PDF file is possible. In fact, each PDF file can be represented by the collection of feature words for that file. When a user makes a query, instead of searching in the entire PDF file, searching can be done in only these keyword collections, which would be considerably faster and produce more relevant results.

4 Why T_EX files and not other markup?

Consider HTML files: for line breaks, paragraphs, even for extra space, explicit commands are required, which makes the source full of commands which do not hold any information with respect to the content of the article.

On the other hand, if we look at a typical T_EX file, it uses fewer formatting commands in comparison to an HTML file. For example, paragraphs are indicated simply by blank lines. Thus, there is less disturbance when extracting text from the source.

5 User wishes

Suppose a user wants to search for a general concept in a repository. This would usually require a full text search, which is time-consuming. To speed up the search, what if the documents were already clustered by subject or concepts?

If we can classify documents beforehand, the search process could be more like this:

- Do we have the query results already saved? If so, return them.
- If not, the query is made against the ‘cluster representatives’, described below, to return the appropriate documents.

6 Challenges

This leads to the question of how to form such concepts or clusters beforehand, without knowing the query. How should such clusters be represented, and how do we find the representatives?

In a document source file, the author highlights words with additional, perhaps invisible, markup. These words presumably help to represent the document. How do we capture these representatives?

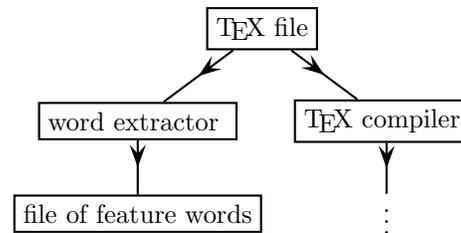
From a PDF file, text extraction is not simple, as we have seen. Another common format, RTF (rich text format), is even less straightforward.

HTML format is somewhat better, although the additional commands mentioned above complicate the job.

There are many commands with which the T_EX compiler collects important information from the T_EX file to highlight in the output file. Some well-known examples:

- For emphasis, L^AT_EX uses commands such as `{\bf ...}`, `\textbf{...}`. So we can identify emphasized words fairly easily.
- To add a word to an index: `\index{word}`.
- For section headings: `\section{...}`.

So we are assuming that the (L^A)T_EX compiler understands that some words are to be highlighted for some reason. We are interested in these words given some special importance by the author. Thus, the general picture looks like this:



Suppose there is a file of 2000 words, out of which we found say 300 words that are special words. Clearly we can increase speed of searches.

7 Looking ahead

We can cluster documents based on their feature words. For example, we might try this rule:

If documents have more than 70% of the feature words in common, group them together.

Now clusters will be defined according to their common feature words, 70% in this case. The remaining 30% of words for each document in the cluster can be a secondary representative of the cluster.

To handle a query, we can search in the primary representation of the clusters and if not found, search in secondary representations. This will make our search even faster.

Journals can provide this facility to their users, and it would be useful for other areas where T_EX files are in use.

Abstracts

Editor's note: Many of the conference presentations are available at <http://www.river-valley.tv/conferences/tug2008> in video form, thanks to Kaveh Bazargan and River Valley Technologies.

— * —

MathTran and \TeX as a web service

Jonathan Fine

In 2006/7 I developed and set up the public MathTran web service www.mathtran.org. This was done with funding from JISC and the Open University. It provides translation of \TeX -notation formulas into high-quality bitmaps. In April 2008 it served 48,000 images a day for a growing range of sites. After tuning, the server could provide about 2 million images a day. It takes about 10 milliseconds to process a small formula.

MathTran images contain rich metadata, including the \TeX source for the image and the `dvi` and `log` outputs due to that source. This makes it straightforward to edit and resize such images, or convert them to another format, such as SVG or PostScript.

MathTran, used with JavaScript, makes it considerably easier to put mathematics on a web page. In particular, the author of the page does not need to install any special software, and does not have to store thousands of image files.

The MathTran project is now focussed on the authoring of mathematical content. It has produced a prototype instant preview document editor. Funded by the 2008 Google Summer of Code, Christoph Hafemeister is developing JavaScript to provide autocompletion for commands and analysis of \TeX errors, all integrated with an online help system embedded in the web page. Separate work is focussed on developing MathTran plugins for WYSIWYG editor web-page components.

This talk will present progress and prospects. It will also discuss some of the broader implications for the \TeX community and software, such as

- People using \TeX without installing \TeX on their machine.
- Help components for web pages.
- Integration with third-party products.
- Standards for \TeX -notation mathematics.
- Learning and teaching \TeX .

Why we need \LaTeX 3

Jonathan Fine

The \LaTeX 3 project started in 1992. Since then, much has changed. XML has replaced SGML and along with X/HTML has become the dominant markup language. CSS has replaced explicit style attributes in HTML pages, and is now a widely understood and used language for specifying design. Internet access is considerably more widespread, the web has gone from 1.0 to 2.0, Microsoft has replaced IBM, Linux went from nothing in 1991 to an

open-source standard, and Google is on track to replace Microsoft.

In 1997 the \LaTeX 3 project said that \LaTeX 3 would provide:

- A new *input document syntax*, that aligns with SGML/XML.
- A new *class file interface*, that aligns with SGML/XML.
- A new *style-designer interface* that can work with a *visually-oriented, menu-driven specification system*.
- An *effective interactive help system* for document authors.
- *Thoroughly documented* and *modular* source code.

These goals are still worth achieving. This talk will focus on some recent progress, and in particular:

- Use of key-value syntax within tags.
- Separation of parsing from processing.
- An improved development environment.
- On-line interactive help systems.

Lua \TeX , MPLib, and random punk

Hans Hagen

We use new Lua \TeX and MPLib features to generate random characters from Donald Knuth's punk font. This was the 'surprise' talk on the TUG'08 program. The full paper will appear in a future issue of MAPS.

Image handling in Lua \TeX

Hartmut Henkel

The Lua language allows for defining new variable types, and Lua \TeX uses this concept for types like 'node' and 'font'. In this talk an image library as part of the Lua \TeX engine is presented, built around a new 'image' type, giving extended image handling and embedding capabilities. The image primitives inherited from pdf \TeX are still fully functional for compatibility.

First the process of image embedding and its limitations using the pdf \TeX primitives is described. Then, after a short introduction about Lua libraries, the 'image' type of Lua \TeX is presented together with the set of new Lua functions for image handling, and their use is illustrated by examples. As work is still ongoing, possible future extensions are discussed as well.

Lua \TeX : what has been done, and what will be done

Taco Hoekwater

At TUG 2007 in San Diego, the first beta version of Lua \TeX was presented. This year the team presents a version where significant parts of the \TeX -Lua API are stable. This talk will give an overview of the components that make up Lua \TeX : what libraries do we have and what callbacks are available. The team has some ideas about the next stages of development and these will be presented as well.

The galley Module or: How I Learned to Stop Worrying and Love the Whatsit

Morten Høgholm

TeX has a well-deserved good reputation for its line breaking algorithm, which has found its way into other software over the years. When it comes to inter-paragraph material such as penalties, skips and whatsits, things start getting murky as TeX provides little help in this area, especially on the main vertical list where most of the action is.

This article describes the `galley` module which seeks to control line breaking as well as taking care of inter-paragraph material being added at the right time. In other words, `galley` can assist packages such as `breqn`, whose goal is to construct paragraph shapes on the fly while taking current ones into account as well as ensuring the output routine doesn't get tricked by penalties, skips and whatsits appearing in places where they could allow breakpoints where none are intended.

Minion Math: The design of a new math font family

Johannes Küster

“Minion Math” is a set of mathematical fonts I have developed over the past 6 years. Designed as an add-on package to Adobe's Minion Pro font family, it consists of 20 OpenType fonts (4 weights, times 5 optical sizes). In future releases it will cover the complete Unicode math symbols, and more.

In the design I tried to remove constraints and to avoid flaws and shortcomings of other math fonts, with the aim of creating the most comprehensive and versatile set of math fonts to date.

Here I present the main design principles of Minion Math, and the most important design decisions I took. I will show samples of the fonts and will compare the fonts to other math fonts as well.

Trademark attribution: Minion is either a trademark or registered trademark of Adobe Systems Incorporated in the United States and/or other countries and used under license.

Multiple simultaneous galleys: A simpler model for electronic documents

Blanca Mancilla, John Plaice, Toby Rahilly

We present a general model for electronic documents supporting parallel containers of content, tied together through link components. This model is usable for a wide range of documents, including simple textual documents with footnotes and floats, complex critical editions with multiple levels of footnotes and critical apparatus, maps with multiple layers of visual presentation, and music scores.

This model is inspired from the C++ Standard Template Library, whose basis is that Containers + Iterators + Algorithms = Programs. In our approach, the ‘iterators’ are pointers into the parallel containers, keeping track of callouts for notes, floats, and parallel links.

The data structures required for this model are remarkably simple, and will allow the rapid development of many different kinds of algorithms.

Windows of opportunity: A (biased) personal history of two decades of L^AT_EX development — Are there lessons to be learned?

Frank Mittelbach

Looking back at twenty-odd years involvement in L^AT_EX development and maintenance the author highlights the (in his opinion) most important milestones and pitfalls.

- What are significant events that came at the right moment?
- Which important events came at the wrong moment?
- What were the biggest failures and why?

From this data the article attempts to draw conclusions as to how the future of L^AT_EX could be shaped in a way beneficial to everybody involved and what needs to happen to make this possible.

A pragmatic toolchain: TeX and friends and friends of friends

Steve Peter

In this talk, we present the toolchain used to produce the award-winning Pragmatic Bookshelf titles (<http://www.pragprog.com>) and examine some of the pleasures and pitfalls encountered using TeX, XML, XSLT, Ruby and other open technologies.

Parallel typesetting

Toby Rahilly, John Plaice, Blanca Mancilla

We present the general mechanism by which logical content, arranged in multiple interacting containers, can be typeset into a set of visual substrates. The overall algorithm is iterative, with the successive iterations refining a multidimensional context that parameterises the behavior of the algorithm.

Each iteration consists of three parts. First, each visual substrate is informed which parts of which logical containers are to be placed thereon. Second, in parallel, the content placed in the substrates is typeset. Third, the resulting layout in each substrate is assessed for goodness, thereby resulting in the refinement to the overall context.

In the talk, we will present the theory and the practice behind this algorithm.

Three typefaces for mathematics

Daniel Rhatigan

This paper examines the issues involved in the design of typefaces for mathematics. After a brief discussion of some of the typographic and technical requirements of maths composition, three case studies in the development of maths types are presented: Times 4-line Mathematics Series 569, a complement to the Times New Roman text types as set with Monotype equipment; American Mathematical Society Euler, an experimental design intended

to contrast against non-mathematical typefaces set with T_EX; and Cambria Math, designed in concert with a new text face to take advantage of new Microsoft solutions for screen display and maths composition.

In all three cases, the typefaces were created to show the capabilities of new technological solutions for setting maths. The technical advances inherent in each font are shown to be as central to its function as its visual characteristics.

By looking at each typeface and technology in turn, and then comparing and contrasting the issues that are addressed in each case, it becomes apparent that even though certain challenges are overcome with technical advances, the need to consider the specific behaviours of type in a maths setting remains constant.

See <http://www.ultrasparky.org/school> for the complete paper and other typographical items.

Medical pedigrees with T_EX and PStricks: New advances and challenges

Boris Veytsman, Leila Akhmadeeva

A medical pedigree is an important tool for researchers, clinicians, students and patients. It helps to diagnose many hereditary diseases, estimate risks for family members, etc. Recently we reported a comprehensive package for automatic pedigree drawing. Since then we have

extended the algorithm for a number of complex cases, including correct drawing of consanguinic relationships, twins and many others.

In this talk we review the facilities of the current version of the program and the new challenges in computer-aided drawing of medical pedigrees.

We try to make the talk interesting to T_EXnicians by discussing the experience of design a T_EX-based application working in a “real world”.

Observations of a T_EXnician for hire

Boris Veytsman

Several years ago the author was tempted by extremely cheap rates for TUGboat advertisements, and declared *urbi et orbi* he was a T_EX consultant. This audacious step led to many interesting experiences. Some results of this work were published on CTAN and listed at <http://borisv.lk.net/latex.html> (the list includes both commissioned packages and the ones I wrote for my own purposes).

In this talk I report on my past projects, big and small, and discuss the lessons learned from my journeys in the fascinating world of publishers, editors and authors. I describe writing book and journal styles, communication with customers and other issues relevant for T_EX consulting.



Aalborg University, Department of Mathematical Sciences, Aalborg, Denmark

American Mathematical Society, Providence, Rhode Island

Aware Software, Inc., Midland Park, New Jersey

Banca d'Italia, Roma, Italy

Center for Computing Sciences, Bowie, Maryland

Certicom Corp., Mississauga, Ontario, Canada

CSTUG, Praha, Czech Republic

Florida State University, School of Computational Science and Information Technology, Tallahassee, Florida

IBM Corporation, T J Watson Research Center, Yorktown, New York

Institute for Defense Analyses, Center for Communications Research, Princeton, New Jersey

MacKichan Software, Washington/New Mexico, USA

Marquette University, Department of Mathematics, Statistics and Computer Science, Milwaukee, Wisconsin

Masaryk University, Faculty of Informatics, Brno, Czech Republic

Moravian College, Department of Mathematics and Computer Science, Bethlehem, Pennsylvania

MOSEK ApS, Copenhagen, Denmark

New York University, Academic Computing Facility, New York, New York

Princeton University, Department of Mathematics, Princeton, New Jersey

Springer-Verlag Heidelberg, Heidelberg, Germany

Stanford Linear Accelerator Center (SLAC), Stanford, California

Stanford University, Computer Science Department, Stanford, California

Stockholm University, Department of Mathematics, Stockholm, Sweden

Université Laval, Ste-Foy, Québec, Canada

Universiti Tun Hussein Onn Malaysia, Pusat Teknologi Maklumat, Batu Pahat, Johor, Malaysia

University College, Cork, Computer Centre, Cork, Ireland

University of Delaware, Computing and Network Services, Newark, Delaware

University of Oslo, Institute of Informatics, Blindern, Oslo, Norway

Vanderbilt University, Nashville, Tennessee

Calendar

2008

- Sep 16–19 ACM Symposium on Document Engineering, São Paulo, Brazil. www.documentengineering.org
- Sep 17–21 Association Typographique Internationale (ATypI) annual conference, “The Old · The New”, St. Petersburg, Russia. www.atypi.org
- Oct 4–5 Oak Knoll Fest XV, “Celebrating a ‘Hot Metal Man’”, honoring Henry Morris and his 50th anniversary in printing, New Castle, Delaware. www.oakknoll.com/fest
- Oct 6 Journée GUTenberg & Assemblée générale, “XML et T_EX”, Centre FIAP, Paris, France. www.gutenberg.eu.org/manifestations
- Oct 10–12 American Printing History Association 2008 annual conference, “Saving the History of Printing”, Grolier Club and Columbia University, New York, New York. www.printinghistory.org
- Oct 16–18 Guild of Book Workers, Standards of Excellence Annual Seminar, Toronto, Ontario. palimpsest.stanford.edu/byorg/gbw
- Oct 18 GuIT meeting 2008 (Gruppo utilizzatori Italiani di T_EX), Pisa, Italy. www.guit.sssup.it/guitmeeting/2008
- Oct 18 NTG 42nd meeting, Dordrecht, Netherlands. www.ntg.nl/bijeen/bijeen42.html
- Oct 21 “Glasgow 501: Out of Print”, an illustrated talk at St Bride Library, London, England. stbride.org/events
- Oct 25–27 The Sixth International Conference on the Book, Catholic University of America, Washington, DC. b08.cg-conference.com
- Nov 7 “Letterpress: a celebration”, conference at St Bride Library, London, England. stbride.org/events
- Dec 8–10 XML-in-Practice 2008, Arlington, Virginia. www.idealliance.org/xml2008

2009

- Jan 8–10 College Book Art Association Biennial Conference, “Art, Fact, and Artifact: The Book in Time and Place”, University of Iowa Center for the Book, Iowa City, Iowa. uicb.grad.uiowa.edu/uicb-cbaa-conference
- Apr 29– May 3 Bacht_EX 2009: 17th Bacht_EX Conference, Bachotek, Poland. For information, visit www.gust.org.pl/BachtTeX/2009
- May 15– Aug 15 “Marking Time”: A traveling juried exhibition of books by members of the Guild of Book Workers. Minnesota Center for Book Arts, Minneapolis. Sites and dates are listed at palimpsest.stanford.edu/byorg/gbw
- Jun 22–25 Digital Humanities 2009, Association of Literary and Linguistic Computing / Association for Computers and the Humanities, University of Maryland. www.mith2.umd.edu/dh09
- Jun 23–27 SHARP 2009, “Tradition & Innovation: The State of Book History”, Society for the History of Authorship, Reading & Publishing, Toronto, Ontario. www.sharpweb.org
- Jun 24–27 DANTE: Exhibitor at LinuxTag, Berlin, Germany. www.dante.de

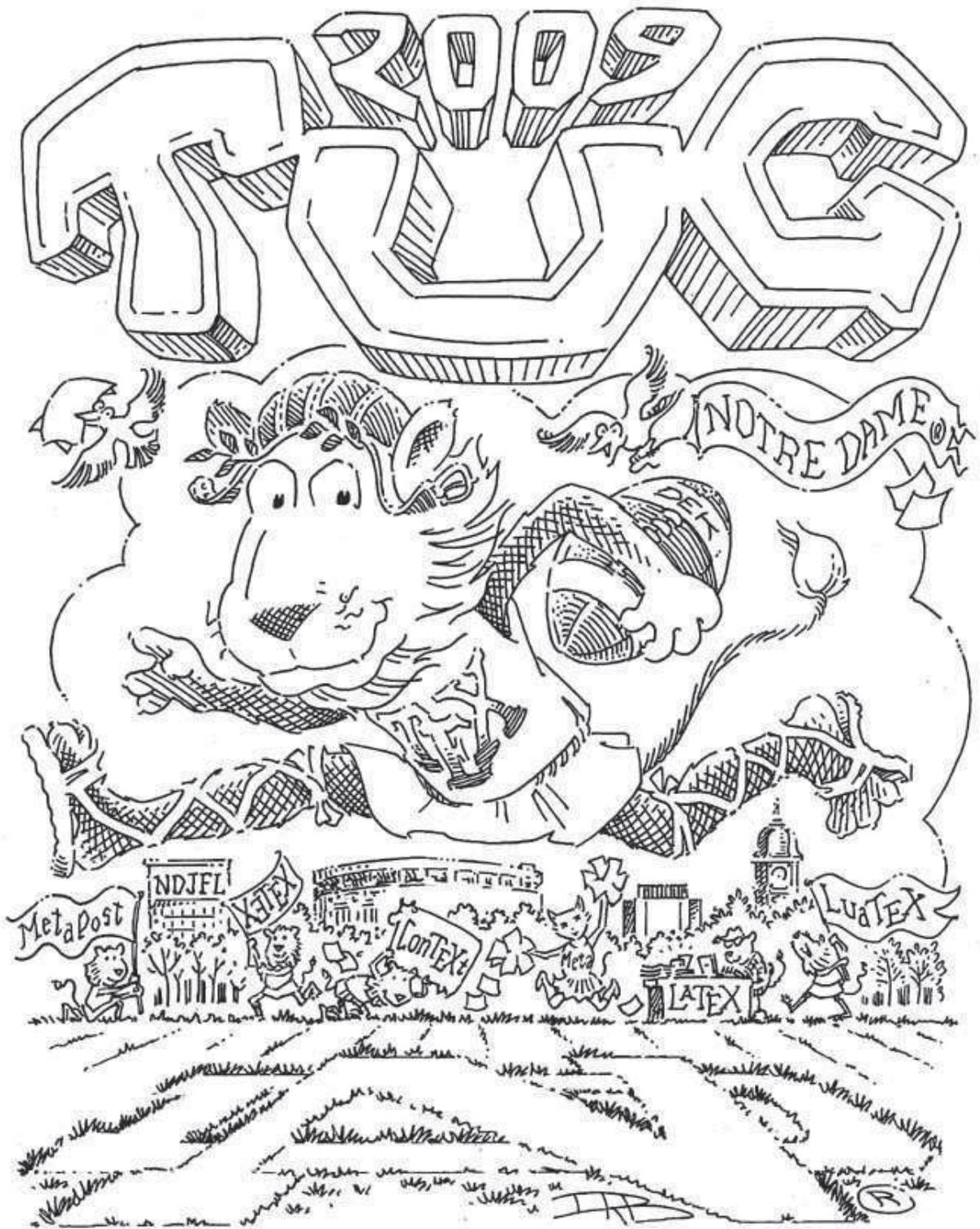
TUG 2009**University of Notre Dame, Notre Dame, Indiana**

- Jul 28–31 The 30th annual meeting of the T_EX Users Group. www.tug.org/tug2009
-
- Aug 3–7 SIGGRAPH 2009, “Network Your Senses”, New Orleans, Louisiana. www.siggraph.org/s2009
- Aug 24–28 EuroT_EX 2009 and 3rd ConT_EXt meeting, The Hague, The Netherlands. www.ntg.nl/EuroTeX2009

Status as of 15 September 2008

For additional information on TUG-sponsored events listed here, contact the TUG office (+1 503 223-9994, fax: +1 206 203-3960, e-mail: office@tug.org). For events sponsored by other organizations, please use the contact address provided.

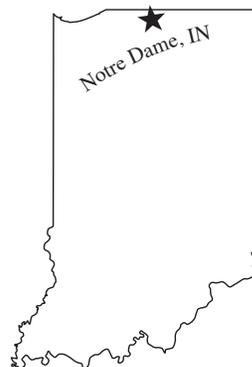
An updated version of this calendar is online at www.tug.org/calendar.



University of Notre Dame
Notre Dame, IN 46556 USA

July 28–31, 2009

Sponsored by the T_EX Users Group,
Notre Dame Journal of Formal Logic,
and the University of Notre Dame



<http://tug.org/tug2009>

T_EX Consultants

The information here comes from the consultants themselves. We do not include information we know to be false, but we cannot check out any of the information; we are transmitting it to you as it was given to us and do not promise it is correct. Also, this is not an official endorsement of the people listed here. We provide this list to enable you to contact service providers and decide for yourself whether to hire one.

TUG also provides an online list of consultants at <http://tug.org/consultants.html>.

Khalighi, Vafa

4/34 Sorrell Street
Parramatta NSW 2150 / Australia
+006 1420969496
Email: [m.aicart \(at\) menta.net](mailto:m.aicart@menta.net)
Web: <http://www.vafakhalighi.com>

I am a professional and experienced L^AT_EXacker/L^AT_EXnician and have been using T_EX and mates for 10 years. I typeset English, Persian, Arabic and mathematics typesetting in those languages as well as drawing any diagrams (mathematics diagrams, physics diagrams, game diagrams, musical notes and chemistry diagrams). My favourite language for drawing is PSTricks, however I am also familiar with xy-pic and MetaPost.

Martinez, Mercè Aicart

Tarragona 102 4^o 2^a
08015 Barcelona, Spain
+34 932267827
Email: [m.aicart \(at\) menta.net](mailto:m.aicart@menta.net)
Web: <http://www.edilatex.com>

We provide, at reasonable low cost, T_EX and L^AT_EX typesetting services to authors or publishers worldwide. We have been in business since the beginning of 1990. For more information visit our web site.

Peter, Steve

310 Hana Road
Edison, NJ 08817
+1 732 287-5392
Email: [speter \(at\) dandy.net](mailto:speter@andy.net)

Specializing in foreign language, linguistic, and technical typesetting using T_EX, L^AT_EX, and ConT_EXt, I have typeset books for Oxford University Press, Routledge, and Kluwer, and have helped numerous authors turn rough manuscripts, some with dozens of languages, into beautiful camera-ready copy. I have extensive experience in editing, proofreading, and

writing documentation. I also tweak and design fonts. I have an MA in Linguistics from Harvard University and live in the New York metro area.

Shanmugam, R.

No. 38/1 (New No. 65), Veerapandian Nagar, Ist St.
Choolaimedu, Chennai-600094, Tamilnadu, India
+91 9841061058

Email: [rshanmugam \(at\) yahoo.com](mailto:rshanmugam@yahoo.com)

As a Consultant I provide consultation, technical training, and full service support to the individuals, authors, corporates, typesetters, publishers, organizations, institutions, etc. and I also support to leading BPO/KPO/ITES/Publishing companies in implementing latest technologies with high level of automation in the field of Typesetting/Prepress/Composition, ePublishing, XML2PAGE, WEBTechnology, DataConversion, Digitization, Cross-media publishing, etc. I have sound knowledge in creating Macros/Styles/Templates/Scripts and Conversions with automation using latest softwares in industry.

Sievers, Martin

Max-Planck-Str. 6–8, 54296 Trier, Germany
+1 49 651 81009-780

Email: [Martin \(at\) TeXBeratung.com](mailto:Martin@TeXBeratung.com)

Web: <http://www.TeXBeratung.com>

As a mathematician I offer T_EX and L^AT_EX services and consulting for the whole academic sector and everybody looking for a high-quality output. From setting up entire book projects to last-minute help, from creating citation styles to typesetting your math, tables or graphics — just contact me with information on your project.

Veytsman, Boris

46871 Antioch Pl.
Sterling, VA 20164
+1 703 915-2406

Email: [borisv \(at\) lk.net](mailto:borisv@lk.net)

Web: <http://borisv.lk.net>

T_EX and L^AT_EX consulting, training and seminars. Integration with databases, automated document preparation, custom L^AT_EX packages, conversions and much more. I have about twelve years of experience in T_EX and twenty-five years of experience in teaching & training. I have authored several packages on CTAN and published papers in T_EX related journals.

2009 T_EX Users Group Election

Barbara Beeton
for the Elections Committee

The positions of TUG President and of eight members of the Board of Directors will be open as of the 2009 Annual Meeting, which will be held in July 2009 at the University of Notre Dame, Notre Dame, Indiana.

The current President, Karl Berry, has stated his willingness to stand for re-election. The directors whose terms will expire in 2009 are: Steve Grathwohl, Jim Hefferon, Klaus Höppner, Dick Koch, Arthur Ogawa, Steve Peter, and Dave Walden. One additional director position is currently unoccupied. Continuing directors, with terms ending in 2011, are: Barbara Beeton, Jon Breitenbacher, Kaja Christiansen, Susan DeMeritt, Ross Moore, Cheryl Ponchin, and Philip Taylor.

The election to choose the new President and Board members will be held in Spring of 2009. Nominations for these openings are now invited.

The Bylaws provide that “Any member may be nominated for election to the office of TUG President/to the Board by submitting a nomination petition in accordance with the TUG Election Procedures. Election . . . shall be by written mail ballot of the entire membership, carried out in accordance with those same Procedures.” The term of President is two years.

The name of any member may be placed in nomination for election to one of the open offices by submission of a petition, signed by two other members in good standing, to the TUG office at least two weeks (14 days) prior to the mailing of ballots. (A candidate’s membership dues for 2009 will be expected to be paid by the nomination deadline.) The term of a member of the TUG Board is four years.

A nomination form follows this announcement; forms may also be obtained from the TUG office, or via <http://tug.org/election>.

Along with a nomination form, each candidate must supply a passport-size photograph, a short biography, and a statement of intent to be included with the ballot; the biography and statement of intent together may not exceed 400 words. The deadline for receipt at the TUG office of nomination forms and ballot information is **1 February 2009**.

Ballots will be mailed to all members within 30 days after the close of nominations. Marked ballots must be returned no more than six (6) weeks following the mailing; the exact dates will be noted on the ballots.

Ballots will be counted by a disinterested party not part of the TUG organization. The results of the election should be available by early June, and will be announced in a future issue of *TUGboat* as well as through various T_EX-related electronic lists.

2009 TUG Election — Nomination Form

Only TUG members whose dues have been paid for 2009 will be eligible to participate in the election. The signatures of two (2) members in good standing at the time they sign the nomination form are required in addition to that of the nominee. **Type or print** names clearly, using the name by which you are known to TUG. Names that cannot be identified from the TUG membership records will not be accepted as valid.

The undersigned TUG members propose the nomination of:

Name of Nominee: _____

Signature: _____

Date: _____

for the position of (check one):

TUG President

Member of the TUG Board of Directors

for a term beginning with the 2009 Annual Meeting, **July 2009**.

Members supporting this nomination:

1. _____
(please print)

(signature) _____
(date)
2. _____
(please print)

(signature) _____
(date)

Return this nomination form to the TUG office (FAXed forms will be accepted). Nomination forms and all required supplementary material (photograph, biography and personal statement for inclusion on the ballot) must be received in the TUG office no later than **1 February 2009**.¹ It is the responsibility of the candidate to ensure that this deadline is met. Under no circumstances will incomplete applications be accepted.

- nomination form
- photograph
- biography/personal statement

T_EX Users Group **FAX:** +1 206 203-3960
Nominations for 2009 Election
P. O. Box 2311
Portland, OR 97208-2311
U.S.A.

¹ Supplementary material may be sent separately from the form, and supporting signatures need not all appear on one form.

Table of Contents (ordered by difficulty)**Introductory**

- 418 *Dave Crossland* / Why didn't METAFONT catch on?
 • personality types, METAFONT's interface, and font vs. typeface design
- 352 *Peter Flynn* / TUG 2008: T_EX's 30th birthday
 • general review of the TUG 2008 conference
- 444 *Manjusha Joshi* / Smart ways of drawing PSTricks figures
 • using GUI programs with PSTricks for geometric figures
- 362 *Jonathan Kew* / T_EXworks: Lowering the barrier to entry
 • report on a new T_EX front end focused on ease-of-use and simplicity

Intermediate

- 380 *Taco Hoekwater* / MetaPost developments: MPlib project report
 • translation of MetaPost functionality to a re-usable library
- 480 *Manjusha Joshi* / Data mining: Role of T_EX files
 • extracting feature words from T_EX files to improve searching
- 356 *Niall Mansfield* / How to develop your own document class—our experience
 • practical techniques for creating custom classes and styles
- 376 *Joe McCool* / A newbie's experiences with Lilypond, Lilypond-book, L^AT_EX and Perl
 • producing nicely typeset traditional Irish music
- 421 *Karel Piška* / Creating cuneiform fonts with MetaType1 and FontForge
 • development process and results for a cuneiform font collection
- 413 *Chris Rowley* / Vistas for T_EX: liberate the typography! (Part I)
 • extracting core T_EX typesetting functionality for reuse
- 372 *Joachim Schrod* / Xindy revisited: Multi-lingual index creation for the UTF-8 age
 • introduction to and rationale for xindy's main features

Intermediate Plus

- 383 *Hans Hagen* / The T_EX–Lua mix
 • introduction to the combination of T_EX and the embedded scripting language Lua
- 365 *Jérôme Laurens* / Direct and reverse synchronization with SyncT_EX
 • synchronizing between T_EX input and output via modifying the base engine
- 454 *Mojca Miklavc* and *Arthur Reutenauer* / Putting the Cork back in the bottle—Improving Unicode support in T_EX
 • recasting hyphenation patterns to support both UTF-8 and 8-bit encodings
- 464 *Ross Moore* / Advanced features for publishing mathematics, in PDF and on the Web
 • using PDF and JavaScript to improve mathematics presentation and navigation
- 392 *Krisztián Póczy*, *Mihály Biczó* and *Zoltán Porkoláb* / docx2tex: Word 2007 to T_EX
 • free XML-based conversion software from Word 2007 (OOXML) to T_EX
- 435 *Ameer Sherif* and *Hossam Fahmy* / Meta-designing parameterized Arabic fonts for AlQalam
 • using METAFONT to render Arabic with calligrapher-level quality
- 426 *Ulrik Vieth* / Do we need a 'Cork' math font encoding?
 • review of OpenType and Unicode math features, subsuming 8-bit encodings

Advanced

- 462 *Hans Hagen* / The LuaT_EX way: `\framed`
 • generalizing paragraph manipulation in LuaT_EX
- 446 *Hans Hagen* / The MetaPost library and LuaT_EX
 • using the new standalone MetaPost library from LuaT_EX and ConT_EXt
- 401 *Jean-Michel Huppen* / Languages for bibliography styles
 • language comparison of bst, nbst, Perl, DSSSL, XSLT, etc.
- 474 *John Plaice*, *Blanca Mancilla* and *Chris Rowley* / Multidimensional text
 • a theoretical underpinning of documents as generalized tuples
- 458 *Stanislav Jan Šarman* / Writing Gregg Shorthand with METAFONT and L^AT_EX
 • an online system converting English text to Gregg shorthand

Reports and notices

- 350 *TUG 2008 conference information*
- 351 *TUG 2008 conference program*
- 482 *Abstracts* (Fine, Hagen, Henkel, Hoekwater, Høgholm, Küster, Mancilla et al., Mittelbach, Peter, Rahilly et al., Rhatigan, Veytsman & Akhmadeeva, Veytsman)
- 484 Institutional members
- 485 Calendar
- 486 TUG 2009 announcement
- 487 T_EX consulting and production services
- 488 TUG 2009 election

TUGBOAT

Volume 29, Number 3 / 2008
TUG 2008 Conference Proceedings

TUG 2008	350	Conference program, delegates, and sponsors
	352	Peter Flynn / <i>TUG 2008: T_EX's 30th birthday</i>
L^AT_EX	356	Niall Mansfield / <i>How to develop your own document class — our experience</i>
Software & Tools	362	Jonathan Kew / <i>T_EXworks: Lowering the barrier to entry</i>
	365	Jérôme Laurens / <i>Direct and reverse synchronization with SyncT_EX</i>
	372	Joachim Schrod / <i>Xindy revisited: Multi-lingual index creation for the UTF-8 age</i>
	380	Taco Hoekwater / <i>MetaPost developments: MPlib project report</i>
	383	Hans Hagen / <i>The T_EX–Lua mix</i>
	376	Joe McCool / <i>A newbie's experiences with Lilypond, Lilypond-book, L^AT_EX and Perl</i>
	392	Krisztián Pócza, Mihály Biczó and Zoltán Porkoláb / <i>docx2tex: Word 2007 to T_EX</i>
	401	Jean-Michel Hufflen / <i>Languages for bibliography styles</i>
Dreamboat	413	Chris Rowley / <i>Vistas for T_EX: liberate the typography! (Part I)</i>
Fonts	418	Dave Crossland / <i>Why didn't METAFONT catch on?</i>
	421	Karel Píška / <i>Creating cuneiform fonts with MetaType1 and FontForge</i>
	426	Ulrik Vieth / <i>Do we need a 'Cork' math font encoding?</i>
	435	Ameer Sherif and Hossam Fahmy / <i>Meta-designing parameterized Arabic fonts for AlQalam</i>
Graphics	444	Manjusha Joshi / <i>Smart ways of drawing PSTricks figures</i>
	446	Hans Hagen / <i>The MetaPost library and LuaT_EX</i>
Philology	454	Mojca Miklavec and Arthur Reutenauer / <i>Putting the Cork back in the bottle — Improving Unicode support in T_EX</i>
	458	Stanislav Jan Šarman / <i>Writing Gregg Shorthand with METAFONT and L^AT_EX</i>
Macros	462	Hans Hagen / <i>The LuaT_EX way: \framed</i>
Electronic Documents	464	Ross Moore / <i>Advanced features for publishing mathematics, in PDF and on the Web</i>
	474	John Plaice, Blanca Mancilla and Chris Rowley / <i>Multidimensional text</i>
	480	Manjusha Joshi / <i>Data mining: Role of T_EX files</i>
Abstracts	482	Abstracts (Fine, Hagen, Henkel, Hoekwater, Høgholm, Küster, Mancilla et al., Mittelbach, Peter, Rahilly et al., Rhatigan, Veytsman & Akhmadeeva, Veytsman)
News	485	Calendar
	486	TUG 2009 announcement
Advertisements	487	T _E X consulting and production services
TUG Business	484	TUG institutional members
	488	TUG 2009 election

Table of Contents (ordered by difficulty)**Introductory**

- 418 *Dave Crossland* / Why didn't METAFONT catch on?
 • personality types, METAFONT's interface, and font vs. typeface design
- 352 *Peter Flynn* / TUG 2008: T_EX's 30th birthday
 • general review of the TUG 2008 conference
- 444 *Manjusha Joshi* / Smart ways of drawing PSTricks figures
 • using GUI programs with PSTricks for geometric figures
- 362 *Jonathan Kew* / T_EXworks: Lowering the barrier to entry
 • report on a new T_EX front end focused on ease-of-use and simplicity

Intermediate

- 380 *Taco Hoekwater* / MetaPost developments: MPlib project report
 • translation of MetaPost functionality to a re-usable library
- 480 *Manjusha Joshi* / Data mining: Role of T_EX files
 • extracting feature words from T_EX files to improve searching
- 356 *Niall Mansfield* / How to develop your own document class — our experience
 • practical techniques for creating custom classes and styles
- 376 *Joe McCool* / A newbie's experiences with Lilypond, Lilypond-book, L^AT_EX and Perl
 • producing nicely typeset traditional Irish music
- 421 *Karel Piška* / Creating cuneiform fonts with MetaType1 and FontForge
 • development process and results for a cuneiform font collection
- 413 *Chris Rowley* / Vistas for T_EX: liberate the typography! (Part I)
 • extracting core T_EX typesetting functionality for reuse
- 372 *Joachim Schrod* / Xindy revisited: Multi-lingual index creation for the UTF-8 age
 • introduction to and rationale for xindy's main features

Intermediate Plus

- 383 *Hans Hagen* / The T_EX–Lua mix
 • introduction to the combination of T_EX and the embedded scripting language Lua
- 365 *Jérôme Laurens* / Direct and reverse synchronization with SyncT_EX
 • synchronizing between T_EX input and output via modifying the base engine
- 454 *Mojca Miklavc and Arthur Reutenauer* / Putting the Cork back in the bottle — Improving Unicode support in T_EX
 • recasting hyphenation patterns to support both UTF-8 and 8-bit encodings
- 464 *Ross Moore* / Advanced features for publishing mathematics, in PDF and on the Web
 • using PDF and JavaScript to improve mathematics presentation and navigation
- 392 *Krisztián Pócza, Mihály Biczó and Zoltán Porkoláb* / docx2tex: Word 2007 to T_EX
 • free XML-based conversion software from Word 2007 (OOXML) to T_EX
- 435 *Ameer Sherif and Hossam Fahmy* / Meta-designing parameterized Arabic fonts for AlQalam
 • using METAFONT to render Arabic with calligrapher-level quality
- 426 *Ulrik Vieth* / Do we need a 'Cork' math font encoding?
 • review of OpenType and Unicode math features, subsuming 8-bit encodings

Advanced

- 462 *Hans Hagen* / The LuaT_EX way: `\framed`
 • generalizing paragraph manipulation in LuaT_EX
- 446 *Hans Hagen* / The MetaPost library and LuaT_EX
 • using the new standalone MetaPost library from LuaT_EX and ConT_EXt
- 401 *Jean-Michel Huppen* / Languages for bibliography styles
 • language comparison of bst, nbst, Perl, DSSSL, XSLT, etc.
- 474 *John Plaice, Blanca Mancilla and Chris Rowley* / Multidimensional text
 • a theoretical underpinning of documents as generalized tuples
- 458 *Stanislav Jan Šarman* / Writing Gregg Shorthand with METAFONT and L^AT_EX
 • an online system converting English text to Gregg shorthand

Reports and notices

- 350 *TUG 2008 conference information*
- 351 *TUG 2008 conference program*
- 482 *Abstracts* (Fine, Hagen, Henkel, Hoekwater, Høgholm, Küster, Mancilla et al., Mittelbach, Peter, Rahilly et al., Rhatigan, Veytsman & Akhmadeeva, Veytsman)
- 484 Institutional members
- 485 Calendar
- 486 TUG 2009 announcement
- 487 T_EX consulting and production services
- 488 TUG 2009 election