

# TUGBOAT

Volume 22, Number 3 / September 2001  
2001 Annual Meeting Proceedings

	115	TUG 2001 Program
	117	Participants at the 22nd Annual TUG Meeting
<b>A T<sub>E</sub>X Odyssey</b>	118	Hans Hagen / <i>Where will the odyssey bring us?</i>
	119	Mimi Jett / <i>Future of publishing, Part 2</i>
	120	William Richter / <i>Integrating T<sub>E</sub>X into a document imaging system</i>
	131	Arthur Ogawa / <i>REV<sub>T</sub>E<sub>X</sub> version 4.0, an authoring package by the American Physical Society</i>
	134	Anita Schwartz (Chair) / <i>The T<sub>E</sub>X History Panel</i>
	136	Hans Hagen / <i>Using T<sub>E</sub>X for high end typesetting</i>
	137	Peter Flynn / <i>T<sub>E</sub>X—a mass market product? Or just an image in need of a makeover?</i>
	140	David Tulett / <i>L<sup>A</sup>T<sub>E</sub>X for Windows: a user's perspective</i>
<b>PDF and T<sub>E</sub>X</b>	146	Hàn Thê Thành / <i>Margin kerning and font expansion with pdf<sub>T</sub>E<sub>X</sub></i>
	149	Ross Moore / <i>PDF presentations using the Marslide package</i>
	160	Hans Hagen / <i>Using T<sub>E</sub>X to enhance your presentations</i>
	161	Donald P. Story / <i>Techniques of introducing document-level JavaScript into a PDF file from a L<sup>A</sup>T<sub>E</sub>X source</i>
	168	Ross Moore / <i>Online self-marking quizzes, pdf<sub>T</sub>E<sub>X</sub>, exerquiz</i>
	180	Martin Schröder / <i>Using pdf<sub>T</sub>E<sub>X</sub> in a PDF-based imposition tool</i>
	181	Nelson Beebe / <i>pdf<sub>T</sub>E<sub>X</sub> Panel</i>
<b>Graphics, XML, and MathML</b>	188	Ross Moore / <i>Adobe plugin for WARMreader</i>
	197	Stephen Oliver / <i>The T<sub>E</sub>Xspec tool for computer-aided software engineering</i>
	204	William Hammond / <i>GELLMU: A bridge for authors from L<sup>A</sup>T<sub>E</sub>X to XML</i>
	208	Bob Caviness / <i>Creating Math Web Documents (Workshop)</i>
<b>Fonts and Tools</b>	209	Alan Hoenig / <i>Typesetting Hebrew with T<sub>E</sub>X</i>
	216	Alan Hoenig / <i>Modernizing Computer Modern</i>
	220	Nelson Beebe / <i>Fonts Panel</i>
	228	Michael Downes / <i>Managing multiple TDS trees</i>
	238	Michael Doob / <i>Installing a CTAN mirror on your desktop</i>
	240	Richard Koch / <i>Installing T<sub>E</sub>Xshop</i>
	247	William Adams / <i>Font installation: Agfa/Eaglefeather to Linotype Zapfino</i>
<b>News &amp; Announcements</b>	251	Calendar
	253	TUG 2003 Announcement
<b>TUG Business</b>	254	Institutional members
<b>Advertisements</b>	255	T <sub>E</sub> X consulting and production services
	256	Just Published: T <sub>E</sub> X Reference Manual by David Bausum
cover3		Blue Sky Research

## TeX Users Group

### Memberships and Subscriptions

*TUGboat* (ISSN 0896-3207) is published quarterly by the TeX Users Group, 1466 NW Naito Parkway, Suite 3141, Portland, OR 97209-2820, U.S.A.

2001 dues for individual members are as follows:

- Ordinary members: \$75.
- Students: \$45.

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. Individual membership is open only to named individuals, and carries with it such rights and responsibilities as voting in TUG elections. For membership information, visit the TUG web site: <http://www.tug.org>.

*TUGboat* subscriptions are available to organizations and others wishing to receive *TUGboat* in a name other than that of an individual. Subscription rates: \$85 a year, including air mail delivery.

Periodical-class postage paid at Portland, OR, and additional mailing offices. Postmaster: Send address changes to *TUGboat*, TeX Users Group, 1466 NW Naito Parkway, Suite 3141, Portland, OR 97209-2820, U.S.A.

### Institutional Membership

Institutional Membership is a means of showing continuing interest in and support for both TeX and the TeX Users Group. For further information, contact the TUG office ([office@tug.org](mailto:office@tug.org)).

*TUGboat* © Copyright 2001, TeX Users Group

Permission is granted to make and distribute verbatim copies of this publication or of individual items from this publication provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this publication or of individual items from this publication under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this publication or of individual items from this publication into another language, under the above conditions for modified versions, except that this permission notice may be included in translations approved by the TeX Users Group instead of in the original English.

Copyright to individual articles is retained by the authors.

Printed in U.S.A.

## Board of Directors

Donald Knuth, *Grand Wizard of TeX-arcana*<sup>†</sup>

Mimi Jett, *President*<sup>\*+</sup>

Kristoffer Rose<sup>\*+</sup>, *Vice President*

Don DeLand<sup>\*+</sup>, *Treasurer*

Arthur Ogawa<sup>\*+</sup>, *Secretary*

Barbara Beeton

Karl Berry

Kaja Christiansen

Susan DeMeritt

Stephanie Hogue

Judy Johnson<sup>+</sup>

Ross Moore

Patricia Monohon

Cheryl Ponchin

Petr Sojka

Philip Taylor

Raymond Goucher, *Founding Executive Director*<sup>†</sup>

Hermann Zapf, *Wizard of Fonts*<sup>†</sup>

<sup>\*</sup>member of executive committee

<sup>+</sup>member of business committee

<sup>†</sup>honorary

### Addresses

General correspondence,  
payments, etc.

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

Delivery services,  
parcels, visitors

TeX Users Group  
1466 NW Naito Parkway  
Suite 3141  
Portland, OR 97209-2820  
U.S.A.

### Telephone

+1 503 223-9994

### Fax

+1 503 223-3960

### Electronic Mail

(Internet)

General correspondence,  
membership, subscriptions:  
[office@tug.org](mailto:office@tug.org)

Submissions to *TUGboat*,  
letters to the Editor:  
[TUGboat@tug.org](mailto:TUGboat@tug.org)

Technical support for  
TeX users:  
[support@tug.org](mailto:support@tug.org)

To contact the  
Board of Directors:  
[board@tug.org](mailto:board@tug.org)

### World Wide Web

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

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

### Problems not resolved?

The TUG Board wants to hear from you:  
Please email to [board@tug.org](mailto:board@tug.org)

TeX is a trademark of the American Mathematical Society.

2001 Annual Meeting Proceedings

TeX Users Group  
Twenty-second Annual Meeting  
Newark, Delaware, August 12-16, 2001

# TUGBOAT

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

TUGBOAT EDITOR      BARBARA BEETON  
PROCEEDINGS EDITORS      KARL BERRY  
   MIMI BURBANK

VOLUME 22, NUMBER 3      •      SEPTEMBER 2001  
PORTLAND      •      OREGON      •      U.S.A.

## **TUGboat**

The September 2001 issue (Vol. 22, No. 3) contains the Proceedings of the 2001 TUG Annual Meeting. One issue remains in the 2001 volume year.

For the 2002 membership year, the communications of the TeX Users Group will be published as one double issue and two regular issues. One issue (probably Vol. 23, No. 1) is expected to contain the proceedings of the TUG 2002 Annual Meeting.

We are unfortunately not able to set a definitive schedule for the appearance of the next few issues.

*TUGboat* is distributed as a benefit of membership to all members.

Submissions to *TUGboat* are reviewed by volunteers and checked by the Editor before publication. However, the authors are still assumed to be the experts. Questions regarding content or accuracy should therefore be directed to the authors, with an information copy to the Editor.

### **Submitting Items for Publication**

Owing to the lateness of the present issue, and the scarcity of material submitted for future issues, suggestions will be accepted and processed as received.

Manuscripts should be submitted to a member of the *TUGboat* Editorial Board. Articles of general interest, those not covered by any of the editorial departments listed, and all items submitted on magnetic media or as camera-ready copy should be addressed to the Editor-in-Chief, Barbara Beeton, to the Managing Editor, Robin Laakso, or to the Production Manager, Mimi Burbank.

The *TUGboat* “style files”, for use with either plain TeX or L<sup>A</sup>TeX, are available from CTAN. For authors who have no network access (browser or FTP), they will be sent on request; please specify which is preferred. Send e-mail to [TUGboat@tug.org](mailto:TUGboat@tug.org), or write or call the TUG office.

This is also the preferred address for submitting contributions via electronic mail.

### **Reviewers**

Additional reviewers are needed, to assist in checking new articles for completeness, accuracy, and presentation. Volunteers are invited to submit their names and interests for consideration; write to [TUGboat@tug.org](mailto:TUGboat@tug.org).

## **TUGboat Editorial Board**

Barbara Beeton, *Editor-in-Chief*  
Robin Laakso, *Managing Editor*  
Mimi Burbank, *Production Manager*  
Victor Eijkhout, *Associate Editor, Macros*  
Jeremy Gibbons, *Associate Editor*,  
*“Hey — it works!”*

Alan Hoenig, *Associate Editor, Fonts*  
Christina Thiele, *Associate Editor*,  
*Topics in the Humanities*

### **Production Team:**

Barbara Beeton, Mimi Burbank (Manager), Robin Fairbairns, Michael Sofka, Christina Thiele

### **Other TUG Publications**

TUG is interested in considering additional manuscripts for publication. These might include manuals, instructional materials, documentation, or works on any other topic that might be useful to the TeX community in general. Provision can be made for including macro packages or software in computer-readable form. If you have any such items or know of any that you would like considered for publication, send the information to the attention of the Publications Committee at [tug-pub@tug.org](mailto:tug-pub@tug.org) or in care of the TUG office.

### **TUGboat Advertising and Mailing Lists**

For information about advertising rates, publication schedules or the purchase of TUG mailing lists, write or call the TUG office.

### **Trademarks**

Many trademarked names appear in the pages of *TUGboat*. If there is any question about whether a name is or is not a trademark, prudence dictates that it should be treated as if it is. The following list of trademarks which appear in this issue may not be complete.

METAFONT is a trademark of Addison-Wesley Inc. PostScript is a trademark of Adobe Systems, Inc.

TeX and  $\mathcal{A}\mathcal{M}\mathcal{S}$ -TeX are trademarks of the American Mathematical Society.

UNIX is a registered trademark of X/Open Co. Ltd.

# TUG 2001 Program

Monday, August 13, 2001

A T<sub>E</sub>X Odyssey

Time	Talk	Speaker
09:00–09:15	<b>Official Opening and Welcome</b>	Mimi Jett
09:15–10:00	T <sub>E</sub> X 2001: Where will the odyssey bring us?	Hans Hagen
10:00–10:30	Future of Publishing, Part 2	Mimi Jett
10:30–11:00	BREAK	
11:00–11:30	Integrating T <sub>E</sub> X into a Document Imaging System	William Richter
11:30–12:00	REVT <sub>E</sub> X version 4.0, an authoring package by the American Physical Society	Art Ogawa
12:00–13:30	LUNCH	
13:00–13:30	<b>PANEL: T<sub>E</sub>X History</b>	Anita Schwartz (Chair)
13:30–14:15	Using T <sub>E</sub> X for high end typesetting	Hans Hagen
14:15–14:45	T <sub>E</sub> X—a mass market product? Or just an image in need of a makeover?	Peter Flynn
14:45–15:15	BREAK	
15:15–16:00	L <sup>A</sup> T <sub>E</sub> X for Windows: A User's Perspective	David Tulett

Tuesday, August 14, 2001

PDF and T<sub>E</sub>X

Time	Talk	Speaker
09:00–09:30	Margin kerning and font expansion with pdfT <sub>E</sub> X	Hàn Thê Thành
09:30–10:00	PDF presentations using the Marslide package	Ross Moore
10:00–10:30	BREAK	
10:30–11:15	Using T <sub>E</sub> X to enhance your presentations	Hans Hagen
11:15–11:45	Presentations with pdfT <sub>E</sub> X using PDFscreen	C. V. Rahdakrishnan (presented by Martin Schröder)
11:45–13:30	LUNCH	
13:30–14:15	Techniques of Introducing Document-level JavaScript into a PDF file from a L <sup>A</sup> T <sub>E</sub> X source	Don Story
14:15–14:45	Online self-marking Quizzes, pdfT <sub>E</sub> X, exerquiz	Ross Moore
14:55–15:15	BREAK	
15:15–15:45	Using pdfT <sub>E</sub> X in a PDF-based imposition tool	Martin Schröder
15:45–16:00	<b>PANEL: pdfT<sub>E</sub>X</b>	Hans Hagen (chair)

**Wednesday, August 15, 2001**  
**Graphics, XML, and MathML**

Time	Talk	Speaker
09:00–09:45	<b>TUG Business Meeting</b>	
09:45–10:30	Adobe plugin for WARMreader	Ross Moore
10:30–11:00	BREAK	
11:00–11:45	The T <sub>E</sub> Xspec Tool for Computer Aided Software Engineering	Stephen Oliver
11:45–12:30	GELLMU: A Bridge for Authors from L <sup>A</sup> T <sub>E</sub> X to XML	William Hammond
12:30–13:30	LUNCH	
13:30–15:00	Creating Math Web Documents	Bob Caviness
15:00–15:15	BREAK	
15:15–16:00	Online DEMO: MathML processing	Hans Hagen
18:00	<b>Banquet:</b> The Arsenal at Old Newcastle	

**Thursday, August 16, 2001**  
**Fonts and Tools**

Time	Talk	Speaker
09:00–09:30	Typesetting Hebrew with T <sub>E</sub> X	Alan Hoenig
09:30–10:00	Modernizing Computer Modern	Alan Hoenig
10:00–10:30	BREAK	
10:30–11:30	<b>PANEL: Fonts</b>	Nelson Beebe (Chair)
11:30–12:00	Managing Multiple TDS Trees	Michael Downes
12:00–13:00	LUNCH	
13:00–13:45	Installing a CTAN mirror on your desktop	Michael Doob
13:45–14:30	Installing T <sub>E</sub> Xshop	Richard Koch
14:30–15:30	Font Installation: Agfa/Eaglefeather to Linotype Zapfino	William Adams
15:30–16:00	BREAK	
16:00–16:15	CLOSING & WRAP-UP: TUG 2002—KERALA, INDIA	

# Participants at the 22nd Annual TUG Meeting August 12–16, 2001 University of Delaware, Newark, Delaware, USA

**William Adams**

ATLIS Graphics & Design, Camp Hill, PA, USA

**Frederick Bartlett**

Springer-Verlag, New York, NY, USA

**John Beach**

Columbus, OH, USA

**Nelson Beebe**

University of Utah, USA

**Barbara Beeton**

American Mathematical Society, USA

**Richard T. Bumby**

Highland Park, NJ, USA

**Susan DeMeritt**

IDA/CCR La Jolla, USA

**Michael Doob**

University of Manitoba, Winnipeg, Manitoba, Canada

**Michael Downes**

American Mathematical Society, USA

**Frank Ganz**

Springer-Verlag, New York, NY, USA

**Steve Grathwohl**

Duke University Press, USA

**Andreas Guelzow**

Concordia University College of Alberta, Edmonton, AB, Canada

**Hans Hagen**

PRAGMA Advanced Document Engineering, The Netherlands

**James Haines**

Society for Industrial & Applied Math, Philadelphia, PA, USA

**Barbara Hamilton**

IDA-CCR Princeton, NJ, USA

**William Hammond**

University of Albany, NY, USA

**Hán Thê Thánh**

Masaryk University, Czech Republic

**Melissa Harrison**

Far Field Associates, LLC, Snohomish, WA, USA

**Jim Hefferon**

Saint Michael's College, Colchester, VT

**Alan Hoenig**

Knowledge Equity, Inc., USA

**Stephanie Hogue**

Lansdale, PA, USA

**Mirko Janc**

Technical Typesetting Inc., Baltimore, MD, USA

**Mimi Jett**

IBM, USA

**Judy Johnson**

Personal T<sub>E</sub>X, Inc., USA

**Richard Koch**

University of Oregon, Eugene, OR, USA

**Robin Laakso**

T<sub>E</sub>X Users Group, USA

**Jenny Levine**

Duke University Press, USA

**Bernice Lipkin**

Sachs L plus B Associates, Bethesda, MD, USA

**Elizabeth Loew**

T<sub>E</sub>Xniques, Inc., Cambridge, MA, USA

**Tom Lucas**

MCR Inc., Mayfield Village, OH, USA

**Paul Mailhot**

PreT<sub>E</sub>X, Inc., Halifax, Nova Scotia, Canada

**Wendy McKay**

California Institute of Technology, USA

**Lothar Meyer-Lerbs**

Bremen, Germany

**Bruce Miller**

NIST, MCS/ITL, Gaithersburg, MD, USA

**Thomas Miller**

University of Wisconsin-Madison, WI, USA

**Margaret Mitchell**

Springer-Verlag, New York, NY, USA

**Patricia Monohon**

University of California, San Francisco, USA

**Ross Moore**

Macquarie University, Sydney, Australia

**P. Narayanaswami**

Memorial University of Newfoundland Canada

**Bob Neveln**

Widener University, Chester, PA, USA

**Art Ogawa**

T<sub>E</sub>X Consultants, Three Rivers, CA, USA

**Stephen Oliver**

University of Manitoba and Atomic Energy Canada, Ltd., Canada

**Cheryl A. Ponchin**

Institute for Defense Analyses, Princeton, NJ, USA

**J.C.N. Rajendra**

Eastern University, Sri Lanka, Chenkalady, Batticalor, Sri Lanka

**William Richter**

Texas Life Insurance Company, USA

**Chris Rowley**

Open University, London, UK

**Volker Schaa**

Dante e.V., Germany

**Lakshmi Sadasiv**

Academic Press, Burlington, MA, USA

**Martin Schröder**

T<sub>E</sub>X Merchandising, Germany

**Anita Schwartz**

University of Delaware, DE, USA

**William Slough**

Eastern Illinois University, Charleston, IL, USA

**Michael Sofka**

Rensselaer Polytechnic Institute, Troy, NY, USA

**John L. Spiegelman**

John L. Spiegelman Publication Services, Jenkintown, PA, USA

**Don P Story**

The University of Akron, OH, USA

**James P. Summe**

Centers for Medicare & Medicaid Services, DHHS, Baltimore, MD, USA

**Matthew Swift**

Allston, MA, USA

**Paul Travis**

MacKichan Software, Inc., Bainbridge Island, WA, USA

**David Tulett**

David Memorial University of Newfoundland, St. John's, NF, Canada

**Alan Wetmore**

Army Research Laboratory, USA

**Jiri Zlatuska**

University, Brno, Czech Republic

# TeX '2001: Where Will the Odyssey Bring Us?

Hans Hagen

Pragma ADE, GH Hasselt, The Netherlands

[pragma@wxs.nl](mailto:pragma@wxs.nl)

<http://www.pragma-ade.com>

## Abstract

If there is one thing that the 2001 Space Odyssey movie has demonstrated, it is probably that even on a relatively short time-span, predictions are hard to make. In spite of much research, some predictions have turned out the opposite while other developments were not foreseen. When Knuth wrote TeX, one of his assumptions was that in 100 years from their incarnation, people should still be able to reproduce his books from the sources. Can we look that far ahead when it comes to typesetting? What will happen in the (near) future with publishing? Is there a pattern in today's developments? Will all typographic problems be solved? In this talk I will try to reflect on these issues from the perspective of today's TeX based publishing.

## The Future of Publishing, Part 2

Mimi Jett

IBM Research

[jett@us.ibm.com](mailto:jett@us.ibm.com)

<http://www.software.ibm.com/techexplorer>

### **Abstract**

At the beginning of the past decade, (TUG'90 in T<sub>E</sub>Xas), we explored the future of publishing with an eye on the role T<sub>E</sub>X and the T<sub>E</sub>X community might play in the evolution of digital publishing. The idea of open software was fostering open communication between all players along the supply chain, from author through editorial and production. Did this dream materialize? Beyond the imagination, but the real benefits are yet to come. Now we see the collaboration between disparate groups and individuals from academia, business, and government resulting in powerful open standards that will shape not only digital publishing, but interaction and communication worldwide.

# Integrating T<sub>E</sub>X into a Document Imaging System

William M. Richter

Texas Life Insurance Company, 900 Washington Avenue, Waco, TX 76703, USA

[hcswmr@texlife.com](mailto:hcswmr@texlife.com)

## Abstract

T<sub>E</sub>Xmerge is an application programming interface for merging variable data into a pre-existing T<sub>E</sub>X document. This paper introduces the API and discusses its application in a document production and imaging environment.

## Introduction

Modern computer hardware and software has made possible the construction of “document-imaging” systems. These systems maintain large repositories of documents in electronic form. In production environments of many large companies and in particular the life insurance industry, a significant percentage of printed documents are produced electronically in an automated fashion, usually by merging variable data into an existing document with some fixed structure. Storing scanned images of these electronically produced print documents wastes time, computing resources, and disk storage space. It is useful to address the problem of document storage along with the related problem of electronically formatting and producing printed documents. Then the choice of document formatter can be made such that the formatting engine used to produce printed documents may be reused to display those same documents in a document imaging environment.

T<sub>E</sub>X has been used as an important component in building a document production and imaging system at Texas Life Insurance Company. T<sub>E</sub>X’s macro facilities, conditional typesetting, text-based source files, a robust page formatting mechanism, and pre-compiled format files allow it to play a central role in the system. T<sub>E</sub>Xmerge, a C-language API, was developed to allow variable data to be merged, under program control, with static T<sub>E</sub>X source documents containing special merge tags to produce a final output document. This API is used to prepare policy contracts, produce automated client correspondence, as well as in interactive document preparation and in application-specific document production. Documents produced via the T<sub>E</sub>Xmerge API are filed in the imaging system using a minimalist approach. T<sub>E</sub>X form files are stored once and separately from all document instances. Variable data along with a pointer to its associated T<sub>E</sub>X form file is all that comprise a stored document instance. When

the document is displayed a “just-in-time” compile technique is used to reconstruct the document’s .dvi file which is converted to PostScript for display purposes.

T<sub>E</sub>X has additional attributes that make it an excellent choice as document formatting engine. The ability to convert raster bitmaps to T<sub>E</sub>X fonts allow complex letter-head/footer macros to be developed and easily used in a fashion that lends itself to effective revision management. Incorporation of a Code 2-of-5 scalable barcode font has enabled printed forms and documents that are returned to the company from external individuals to be recognized by the document imaging scanner and automatically filed in the imaging system.

## T<sub>E</sub>Xmerge API

T<sub>E</sub>Xmerge is a C-language API for merging variable data into a pre-existing T<sub>E</sub>X document (referred to as the “form”). The API is simple, light-weight, and easy to integrate into applications.

The API consists of a small number of functions, here listed in appendix A in the normal order of use. Most functions return an integer result code. Zero implies successful return. The integer result may be passed to the function `TeXmerge_GetErrorString()` to retrieve the corresponding error text string.

**Example C Program** A straightforward application of the T<sub>E</sub>Xmerge API is illustrated in fig. 1. This program,

1. Creates an associative array with three elements.
2. Sets the elements to have names `THISVAR`, `THATVAR`, and `ANOTHERVAR`, respectively.
3. Opens an output file.
4. Merges the associative array into an existing T<sub>E</sub>X file called `test_form.tex` to create a temporary file called `temp.tex`.
5. Closes the output file and processes it for viewing with `xdvi`.

```

#include "stdio.h" #include "TeXmerge.h"

int main(int argc, char **argv) { TeXmergeName_t *array; int
count=3; int ret; FILE *fp; char *out_pathname="temp.tex"; char
*form_pathname="test_form.tex";

    array = TeXmerge_AllocNames(count);
    TeXmerge_SetArrayEntry("THISVAR", "some value", &array[0]);
    TeXmerge_SetArrayEntry("THATVAR", "blah, blah", &array[1]);
    TeXmerge_SetArrayEntry("ANOTHERVAR", "la-te-dah", &array[2]);
    ret = TeXmerge_OpenOutput(out_pathname, &fp, 0);
    if (ret != TXM_OK) {
        fprintf(stderr, "TeXmerge_OpenOutput(%s): %s\n", out_pathname,
            TeXmerge_GetErrorString(ret));
        return(1);
    }
    ret = TeXmerge(form_pathname, array, count, 0);
    if (ret != TXM_OK) {
        fprintf(stderr, "TeXmerge(%s): %s\n", form_pathname,
            TeXmerge_GetErrorString(ret));
        return(1);
    }
    TeXmerge_CloseOutput(fp);
    TeXmerge_View(out_pathname, TXM_WAIT);
    return(0);

```

**Figure 1:** C-language example application of the T<sub>E</sub>Xmerge API

---

```

\batchmode
\def\THISVAR{some value}
\def\THATVAR{blah, blah}
\def\ANOTHERVAR{la-te-dah}
\input test_form.tex
\bye

```

---

**Figure 2:** A T<sub>E</sub>X file produced by the sample C program in fig. 1

After the call to `TeXmerge_CloseOutput()` the contents of `temp.tex` would appear as in fig. 2.

`test_form.tex` can have any T<sub>E</sub>X code of your choosing, including invocations of `\THISVAR`, `\THATVAR`, and `\ANOTHERVAR`.

**Python Binding** A Python<sup>1</sup> binding for the T<sub>E</sub>Xmerge API is also available. A re-implementation of the previous C-code is given in fig 3.

Notes:

1. The Python version is cleaner.
2. Error detection via return values has been replaced with Python's exception mechanism; i.e., instead of methods returning integer result codes, they throw exceptions of the appropriate

<sup>1</sup> <http://www.python.org>

type which may be caught via the `try/except` construct.

3. The `TeXmergeName_t` arrays used in the C-code example just use simple Python dictionary objects. As a result the nagging `count` integer with tracks the number of array elements is no longer needed.
4. Constants defined in `TeXmerge.h` are accessed as attributes of the Python `TeXmerge` module.

### T<sub>E</sub>Xmerge in Production

Figure 4 depicts data flow between applications using T<sub>E</sub>Xmerge and applications relating to the document imaging system (DIS). Most of the lines in the figure represent, not the flow of documents, but the flow of data necessary to build documents. Data from the policy administration system feeds a number of print-producing applications. Print producers come in four "flavors".

**Bulk and Custom Print Producers** create large volumes of documents such as annual reports to policy-holders, billings for premium due, and automated client correspondence. Most of these documents are from one to three pages in length and usually consist of three standard parts: a client copy, an agent copy and a file copy. All three parts have

```
import TeXmerge import sys

array = {'THISVAR': 'some value',
        'THATVAR': 'blah, blah',
        'ANOTHERVAR': 'la-te-dah'}

out_pathname = 'temp.tex' form_pathname = 'test_form.tex' try: fp
= TeXmerge.openOutput(out_pathname) except IOError, errmsg:
    sys.stderr.write('TeXmerge.openOutput(%s): %s\n' % (out_pathname, errmsg))
TeXmerge.merge(form_pathname, array, fp) TeXmerge.closeOutput(fp)
TeXmerge.view(out_pathname, TeXmerge.TXM_NOWAIT)
```

Figure 3: Python example.

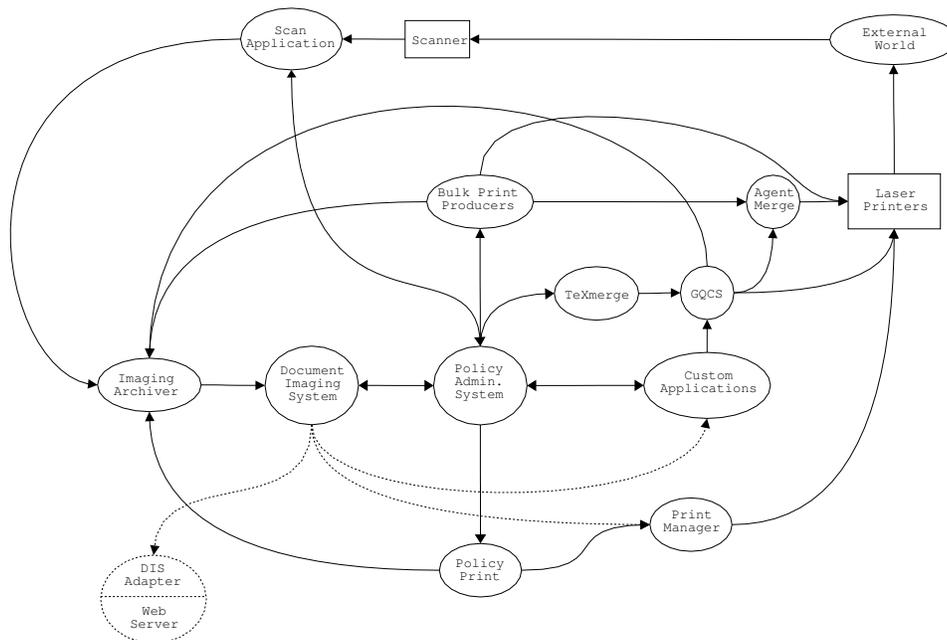


Figure 4: Intra-application flow of document data.

content in common, but the agent and file copies may have additional information.

**Policy Print** is a customized application which produces policy contracts. Contents of policy contract documents vary by product and state in which the policy is issued. Typical contracts are 20 to 30 pages in length when printed duplex and their structure can be complex formatting-wise. A number of pages contain variable tabular information, certain pages must draw paper from an alternate input paper source on the printer, and still other pages must be landscape oriented. Because of the volume of print produced by the policy print ap-

plication<sup>2</sup> its output is routed to a print manager which tracks what documents are to be printed and performs the task of driving print streams to multiple printers. Work is currently under way to deposit a copy of policy contracts into the DIS. Also under construction is a mechanism which will allow the print manager to access the DIS “on-the-fly” as a document is being printed to retrieve an image copy of the original “application for insurance” and insert it into the print stream.<sup>3</sup>

<sup>2</sup> As of this writing, Texas Life produces around 2,000 policy contracts per month.

<sup>3</sup> This and other trickery such as selecting alternate input paper sources on the printer are accomplished through especially malicious abuse of the `special` macro.

**General Queue Collection System** Before discussing the last two print producer types we introduce the General Queueing Collection System (GQCS). Multi-user interactive applications produce multi-part documents that need to be saved for later processing or printing. GQCS removes the details of output handling from print producing programs and becomes a central point for collecting data related to print requests. It collects the tagged data for these documents and stores it in *sub-queues*. A single GQCS daemon can serve multiple sub-queues. Usually it serves three for the standard image, agent, and client copies of a document. At night the contents of each sub-queue is extracted in bulk to output files, each file being forwarded to its destination system for final processing. Documents in the image sub-queue are routed the image archiver for storage in the DIS. Documents in the agent sub-queue are routed to the agent merge system which correlates items destined to a given agent into a consolidated print stream that is printed and mailed as a single bundle to the agent. Documents in the client sub-queue are printed and mailed to the external world.

**Interactive Print Producers** The final two print producer types are interactive in nature. *Custom Applications* are stand-alone systems that integrate T<sub>E</sub>Xmerge to produce printed documents associated with transactions executed against the policy administrative system. Like the bulk print producers, these programs create documents that usually consist of the standard client / agent / file copies.

**The T<sub>E</sub>Xmerge Application** The remaining print producer is *T<sub>E</sub>Xmerge*. It was the first application developed to use the T<sub>E</sub>Xmerge API, and its purpose is to generate documents interactively from pre-configured “form” letters. A detailed data flow diagram of T<sub>E</sub>Xmerge is shown in fig. 5. T<sub>E</sub>Xmerge uses a configuration file (XMC file) to declare the constituent parts of a document and to which sub-queue of a GQCS daemon each part of the document should be deposited.

For illustrative purposes consider the following files which together make a T<sub>E</sub>Xmerge document.

*Example.xmc:*

---

```
Example_client.tex  client
Example_agent.tex   agent
Example_image.tex   image
```

---

*Example\_client.tex:*

---

```
% Example_client.tex - the client copy
\stdLetterHead
```

---

```
\input Example_com.tex
\stdFooter
```

---

*Example\_agent.tex:*

---

```
% Example_client.tex - the agent copy
\stdLetterHead
\input Example_com.tex
\vfil \centerline{* AGENT COPY *}\vfil\stdFooter
```

---

*Example\_image.tex:*

---

```
% Example_image.tex - the image copy
\stdLetterHead
\input Example_com.tex
\vfil \centerline{* FILE COPY *}\vfil\stdFooter
```

---

*Example\_com.tex:*

---

```
% Example_com.tex - part of the document
%                               common to all three parts
\texmergevar THISVAR \texmergevar THATVAR
\texmergevar ANOTHERVAR
```

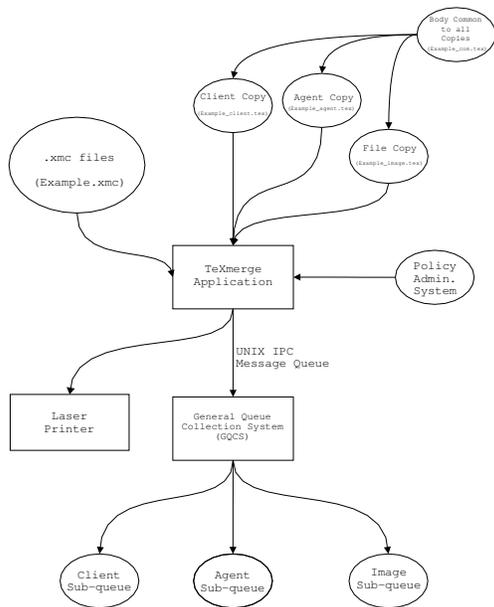
---

This is just a sample document to show how the multi-part document mechanism operates and how merge variables work. Here we insert \THISVAR. and now \THATVAR, and finally \ANOTHERVAR.

---

These four example .tex files and the .xmc file correspond directly with the entities shown in fig. 5. The environment variable \$TEXMBASEDIR connects the T<sub>E</sub>Xmerge application to the directory where the .xmc and .tex files reside. It looks for .xmc files in a subdirectory named xmc within \$TEXMBASEDIR and for .tex files in a subdirectory named tex within \$TEXMBASEDIR.

At startup T<sub>E</sub>Xmerge scans the XMC directory and assumes the files it finds there are XMC configuration files and lists them in a chooser box as in fig. 6. When a selection is made, the configuration file is read to determine what .tex files will be used in the document and to what GQCS sub-queues the constituent parts will be deposited when saved. Then each configured .tex file is scanned recursively via the T<sub>E</sub>Xmerge API to determine what merge variables are declared and it displays a frame as in fig. 7. With the frame displayed, T<sub>E</sub>Xmerge’s work is done and it’s the user’s turn to work. The task of entering data into the merge fields may be shortened in many cases. The T<sub>E</sub>Xmerge application has been programmed to recognize the names of many merge variables and it assigns special meaning to them. For example, ONAME represents a policy owner’s name, INAME is the insured’s name, SDATE is the system date, etc. By typing a policy number into the POLNUM field and pressing enter T<sub>E</sub>Xmerge

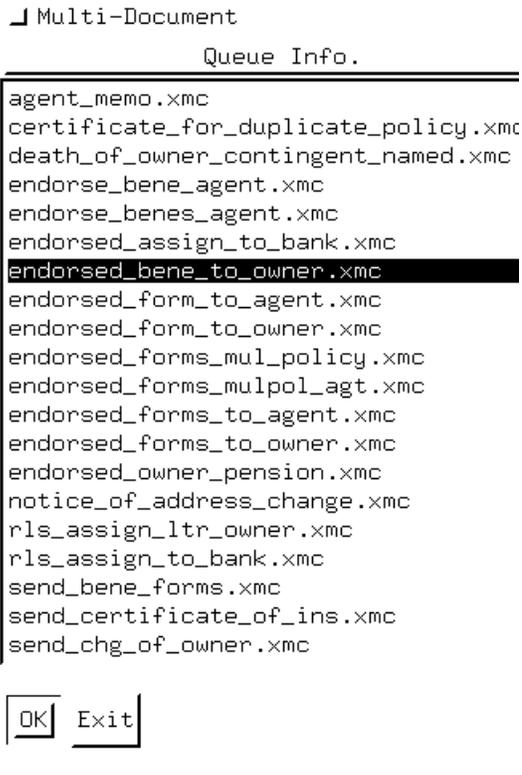


**Figure 5:** TeXmerge multi-part document configuration.

will access the policy administration system using the entered value. If it finds data for the matching number then all merge variable names found to have special meaning will be filled with data from the system. Once the merge fields are populated the document can be saved via GQCS, printed, or viewed via `xdvi`. The printing and viewing operations are accomplished via functions in the TeXmerge API.

**Workflow Management** Using a 2-of-5 barcode font with TeX has enabled a simple workflow environment to be established. A number of documents are intended to be completed by clients externally and returned to the company for further processing. The TeXmerge API has been extended in the following way: When a document is printed via the `TeXmerge_Print()` API call the form document is scanned for the sequence `\def\WRKBARCODESTR{...}`. If this macro is in the document two things happen.

- A transaction identification number will be assigned to the document and stored in the policy administration system.
- The transaction identification number will be encoded as a barcode and printed in the footer of the document. When the document is re-



**Figure 6:** Top-level window of the TeXmerge application.

turned and it is scanned, the transaction identification number will be recognized by the document scanning equipment.

These events enable the document to be automatically archived into the DIS without operator intervention. Future enhancements will also automatically route and image of the document back to the clerk who initiated the transaction for final processing.

**Integrating TeX into the Document Imaging System**

Finally we focus on the document imaging system in use at Texas Life, and the storage of documents produced via the TeXmerge API. A full description of the DIS is outside the scope of this paper; however a brief overview is appropriate here.

The DIS was originally designed to store two types of documents.

1. Reports normally printed on an old-style IBM line printer. These reports range from one to 10,000 pages or more, each page consisting of a

**Figure 7:** Merge field input window of a T<sub>E</sub>Xmerge document.

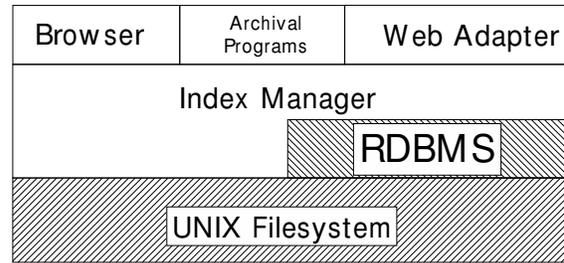
fixed number of rows and columns of printable characters from the ASCII codeset.

2. Scanned images. Usually stored in the CCITT Group-3 fax format.

Over time document types based on Hewlett-Packard’s PCL page formatting language (i.e., PCL print streams normally destined to a LaserJet printer), and Adobe’s PostScript page description language were added, as well as several standard graphics file formats such as GIF, JPEG, and TIFF.

The screen shots in figures 10 and 11 are of the browser used to view documents stored in the DIS. They show both the index panel used for reviewing the document titles that are stored in the system, and the document panel which actually displays the documents. Folder tabs near the bottom of the window allow quick switching between the index and document displays and eliminate the need for manipulating multiple windows.

From a storage perspective the DIS is split into two components (see fig. 8). Contents of documents are stored in files residing within a UNIX filesystem (*host files*). Document titles are indexed into a hierarchical system having a familiar “document/folder” paradigm and is provided by a Relational Database which associates each title with a host file containing



**Figure 8:** Layer diagram of the document imaging system.

the document’s content. Further, the index tracks what folders own what documents.

Host files have a simple internal format. Each host file contains a line-oriented *metadata section*, which stores data about the document. The metadata section contains index information and can be used to rebuild the DIS database index should it become damaged. Format of the host file following the metadata section is document type dependent. ASCII- and PCL-based documents store multiple pages per host file. Scanned documents in CCITT Group-3 fax/JPEG/GIFF/etc. store each page in a separate host file. The storage mechanism for T<sub>E</sub>Xmerge and PostScript documents will be discussed in the next section.

### Storage Strategy for T<sub>E</sub>Xmerge Documents

The storage mechanism for archiving T<sub>E</sub>Xmerge-based documents provides several options which result in files sizes from only a few kilobytes (depending on the amount of variable data to be merged into the document) up to the size required to hold a full PostScript version of the document.<sup>4</sup> The different options for storing T<sub>E</sub>Xmerge-based documents are rooted in the idea that this type of document can exist in at least three forms:

1. A simple associative array of tagged data. Special tags in the data specify what `.tex` file the data should be merged with (TEXTFILE) and in what directory the `.tex` file is to be found (TEXTBASEDIR). This is the canonical form of a T<sub>E</sub>Xmerge document (at least from a storage perspective) and consumes the least amount of disk space.
2. A file of T<sub>E</sub>X code ready to be compiled. A T<sub>E</sub>Xmerge document in this form would appear as in fig. 2.

<sup>4</sup> Most correspondence produced by Texas Life results in PostScript file sizes of 70 kilobytes on average.

3. A PostScript file from `dvips` and is obtained from processing the `.dvi` file which result when compiling the file from item 2 above. From a display/ rendering perspective PostScript is the canonical form for `TeXmerge` documents since they must be in this form before they can be displayed.

There are benefits and disadvantages to storing `TeXmerge` documents in each of these forms (a classic *speed vs. space* dilemma), although the first and last forms seem most useful. Operationally, storing the PostScript is simplest and is least demanding in CPU cycles. In the present DIS Ghostscript is used to rasterize PostScript code for display on an X-Windows terminal, and already having the document in PostScript yields the least amount of work to get the image displayed. Depending on complexity of the document, number of fonts used, etc. the size of a PostScript file produced by `dvips` can be considerable, so storing `TeXmerge` documents in this form does not minimize disk space usage.<sup>5</sup> Option 1 above minimizes disk space usage at the cost of CPU cycles and other complications. The majority of documents will be stored and seldom, if ever, viewed. Therefore, the constraint minimizing disk space is the controlling consideration when choosing a storage strategy.

**The Minimalist Storage Strategy** Saving disk space costs in other areas. Each time the document is displayed, it must be compiled by `TeX` and then converted to PostScript via `dvips` which costs in CPU cycles. Both of these operations can be accomplished via the `TeXmerge` API. The more subtle problem is how to store the `.tex` form files.

**TeX-Freeze and Revision Management** Saving the tagged data array in the document along with the name of its associated `.tex` form file is not sufficient to enable the document to be reproduced. The form file (and any `.tex` files that it `\input`'s) must somehow also be stored internal to the DIS.<sup>6</sup> A complicating factor is that the original form files themselves will be modified at various times and new forms will be added. So while the form files are living, evolving entities, the documents stored in the DIS must be static and always appear exactly as they did the day they were produced.

The mechanism which allows `TeXmerge` documents to appear static in the DIS “freezes” the

<sup>5</sup> The current DIS stores over 500,000 `TeXmerge` documents. (And growing daily.)

<sup>6</sup> Form files used by the `TeXmerge` application and other `TeXmerge`-enabled applications are stored in UNIX filesystems and are independent (but related) to the forms stored in the DIS.

form files when `TeXmerge` documents are archived into the DIS. The place where frozen form files are stored is called the “`TeXfreeze`”, and it is simply the mount-point for a UNIX filesystem. Pathnames of form files in the `TeXfreeze` are derived by concatenating the `TeXfreeze` pathname and the form’s original pathname. For example, if we have a form `/x/y/z/tex/form1.tex` and the `TeXfreeze` directory is `/texfreeze`, then the pathname of the frozen version of `form1.tex` would be `/texfreeze/x/y/z/tex/form1.tex`.

Having a storage mechanism is only half the picture; we still need a method to allow for multiple versions of the same form. We accomplish this by inserting a timestamp into the filename of the form document. So the frozen pathname in the example above would become something like `/texfreeze/x/y/z/tex/form1.20010618.tex`. The most recent version of each form is pointed to by a softlink. The name of the softlink is the form’s un-timestamped name. Continuing the example; in `/texfreeze/x/y/z/tex` we would have a directory entry that looks like:

```
form1.tex —> form1_20010618.tex.
```

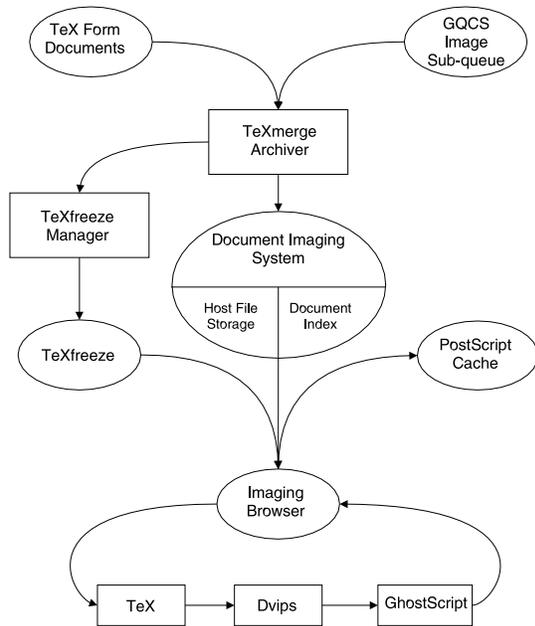
Freezing of form files is the task of the `TeXfreeze` manager. It detects changes between a form file and its current frozen version by calculating an MD5 cryptographic hash of the two files. If the hash values differ the new version of the form is stored in the `TeXfreeze` as discussed above and the softlink is updated to point to the new version.<sup>7</sup> The `TeXfreeze` manager also recursively follows `\input` macros and freezes those files also<sup>8</sup>.

Within the document’s data there is a tag (`TEXMFFORMAT`) which controls the format `TeX` should use when processing the input source. Having laid out the scheme for tracking revisions of `TeXmerge` forms, it should be pointed out that the exact same process is carried out for freezing format files.

With `TeXfreeze` providing archival and revision management facilities for form files, we turn to the process of archiving tagged data for a `TeXmerge` document. The document’s data is stored in a host file as discussed in the section “[Storage Strategy for TeXmerge Documents](#)” with one slight modification. The value of `TEXFILE` (the document’s associated form filename) is adjusted to have the proper timestamp value (obtained from a `readlink(2)` system call on the softlink) inserted into the filename. Also,

<sup>7</sup> The MD5 hashes are also saved in the `TeXfreeze` for efficiency purposes.

<sup>8</sup> Using the environment variable `TEXINPUTS` to search for non-explicit pathnames.

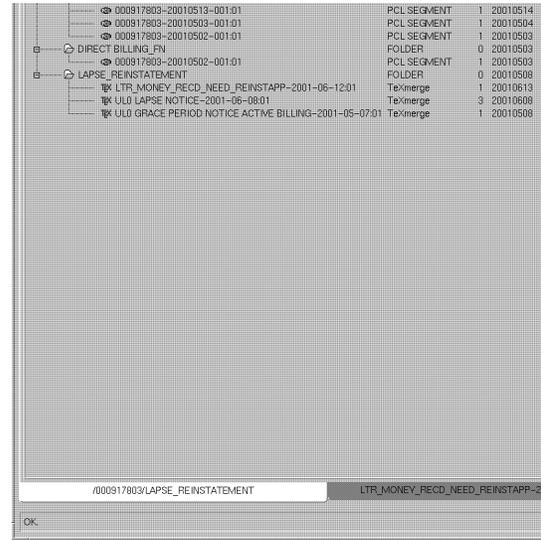


**Figure 9:** Storage strategy for T<sub>E</sub>Xmerge documents.

the TEXMFFORMAT tag will be adjusted with its proper timestamp value.

**“Just-in-Time” Compile** With form files frozen in the T<sub>E</sub>Xfreeze and the document’s data archived in the DIS the task of displaying the document is almost trivial. The T<sub>E</sub>Xmerge API is used to convert the tagged data into a `.tex` file which is compiled by T<sub>E</sub>X. The `.dvi` is passed through `dvips` and the resulting PostScript is rasterized and displayed. One “trick” here is getting T<sub>E</sub>X to find the form file(s). There is a tag in the document’s data, `TEXMBASEDIR` which originally pointed to the directory in which the document’s form resided. That value must now be prepended with the directory of the T<sub>E</sub>Xfreeze. We accomplish this by setting an environment variable, `TEXFREEZE`, to point to the T<sub>E</sub>Xfreeze directory. Then, just before `exec()`’ing the T<sub>E</sub>X compiler, the environment variable `TEXINPUTS` is prepended with `$TEXFREEZE/$TEXMBASEDIR/tex`.

The pipeline sequence of tagged-data to `.tex` to `.dvi` to `.ps` to pixmap is expensive computationally. An easy way to limit this series of operations each time a T<sub>E</sub>Xmerge document is displayed is to insert a cache manager at the beginning of the sequence. The final PostScript output is saved in the cache. Each time before a T<sub>E</sub>Xmerge document is displayed a check is made to see if the PostScript is already in



**Figure 10:** Index panel of imaging browser application.

the cache. If it is found there the whole compile sequence is skipped. Files are dropped from the cache after reaching an age of two weeks to keep the size of the cache directory bounded.

### Future Developments

**T<sub>E</sub>Xmerge-2** Development is complete for the T<sub>E</sub>Xmerge API and no further work on it is planned. T<sub>E</sub>Xmerge-2, a follow-on API, is being developed. The monolithic `.tex` form files will be replaced with a mechanism which assembles the `.tex` file from a series of small text segments or paragraphs that can be reused in various documents. The simple `\texmergevar` variable declaration will be expanded to include other types of variables such as toggle buttons and option boxes to make documents with conditional elements easier to construct. All configuration information, text segments, and variable declarations will be stored in RDBMS tables. A major reason for eliminating the `.tex` form files is resistance from non computer-literate users to embrace the T<sub>E</sub>X philosophy and learn a new skill. Documents based on the T<sub>E</sub>Xmerge-2 API will not have the space efficiency that T<sub>E</sub>Xmerge documents have since all text for T<sub>E</sub>Xmerge-2 documents will be stored directly in the document’s data tags, and

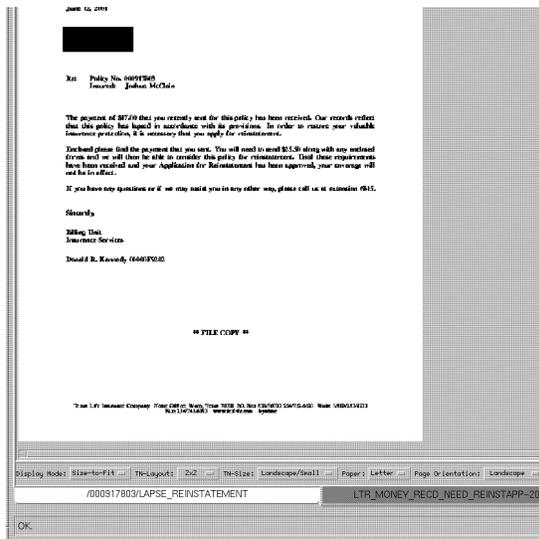


Figure 11: Document panel of imaging browser application.

preparation of the finished document will be less efficient due to the large number of database accesses that will be required to build the  $\text{\TeX}$  code for the document. It is the opinion of the author that  $\text{\TeX}$ merge-2 is not the best track that new development efforts should take; it is management and end-users desire.

**Web Interface to DIS** As shown in fig. 4, work is currently under way to serve documents from the DIS to web browsers. A set of library routines is being developed to render any document type in the DIS to Adobe’s Portable Document Format (PDF).

This library will be used in the planned DIS web server adapter. Security issues are a major consideration that make this an extensive development effort.

**Integrate METAPOST into DIS** Other authors have integrated METAPOST into applications. METAPOST is anticipated to be useful in designing forms where lots of graphics primitives (lines, curves, etc.) are required. An API similar in style to the  $\text{\TeX}$ -merge API is planned.

**Utilize Advanced  $\text{\TeX}$  Formats** To date all form documents are based essentially on the plain  $\text{\TeX}$  format. The  $\text{\TeX}$ merge API allows any  $\text{\TeX}$  format to be used via the  $\text{\TeX}$ MFORMAT environment variable. The  $\text{\LaTeX}$  and  $\text{\ConTeXt}$  formats are anticipated to be useful tools in developing new policy contract forms and for other documents needing a consistent “look-and-feel.”

**Utilize Other Typesetters** Other typesetting engines such as  $\text{\pdfTeX}$ ,  $\text{\xm\TeX}$ , and  $\text{\mathcal{T}\mathcal{S}}$  should be investigated to see what benefits they might bring to the document production and archival environment.

### Conclusion

$\text{\TeX}$  is an excellent engine for integration into a production environment where large volumes of documents need to be produced and archived efficiently. The  $\text{\TeX}$ merge API allows that integration to occur in many applications and for many purposes. Efficiency is gained by moving data between systems instead of fully formatted documents. Finally, using  $\text{\TeX}$ merge in the DIS has provided many benefits, the most important being the space savings derived from the method of storing  $\text{\TeX}$ merge documents, and the ability to interactively produce documents and archive them using the same formatting engine.

**Appendix A T<sub>E</sub>Xmerge API**

`TeXmergeName_t *TeXmerge_AllocNames(int count)`

Variable data is passed to T<sub>E</sub>Xmerge as an associative array. Each entry in the array is a name/value pair. This function allocates an array of such name/value pairs containing `count` entries.

`TeXmergeName_t *TeXmerge_FreeNames(TeXmergeName_t *array, int count)`

This function frees a name/value array. Obviously this one is not listed in order of use!

`int TeXmerge_SetArrayEntry(char *name, char *value, TeXmergeName_t *entry)`

This function sets `entry` to the passed `name` and `value` strings. The next two functions are for convenience and their use is not mandatory. In this and the next function, `value` may be passed as `NULL` to indicate 'no value'.

`int TeXmerge_SetName(char *name, char *value, TeXmergeName_t *array, int count)`

This function searches `array` for an entry whose name value matches `name`, and then sets the entry's value to `value`.

`char *TeXmerge_GetName(char *name, TeXmergeName_t *array, int count)`

This function searches `array` for an entry whose name value matches `name`, and then returns the entry's value string. The function returns `NULL` if `name` is not found in `array`.

`char *TeXmerge_GetNames(char *pathname, TeXmergeName_t **array, int *count)`

This function searches the `.tex` file specified in `pathname` for occurrences of lines having the format:

```
\texmergevar NAME
```

Lines of this form enumerate the merge variables that the document will use. The function returns a list of the names in the pointer variable pointed to by `array`. The number of elements in the array is returned in the integer pointed to by `count`. `array` should be freed when no longer needed with a call to `TeXmerge_FreeNames()`.

`int TeXmerge_OpenOutput(char *pathname, FILE **outFP, char *preamble)`

This function creates an output file named `pathname`. An opened `FILE` pointer is returned in `outFP`. `preamble` is an optional "snippet" of T<sub>E</sub>X code that should be written at the beginning of the file. If `preamble` is passed as `null`, then no preamble code is generated.

`int TeXmerge(char *pathname, TeXmergeName_t *array, int count, FILE *fp, int options)`

This function is the heart of the API. `pathname` is the name of a T<sub>E</sub>X form file containing invocations of macros whose names are the name values set in the passed `array`. `fp` is the `FILE` pointer returned by `TeXmerge_OpenOutput()`. `options` controls the merge operation. Currently the only option is whether or not to draw a frame around the merged variables (`TXM_FRAMEVARS`).

`int TeXmerge_CloseOutput(FILE *fp)`

After all invocations of the above functions are complete, this function should be called to close the output file. `fp` is the `FILE` pointer returned from `TeXmerge_OpenOutput()`.

`int TeXmerge_Process(char *pathname, char *dvidrv_name)`

Once the output file has been closed, it is ready for backend processing by T<sub>E</sub>X. This function invokes T<sub>E</sub>X and then the dvi driver named in `dvidrv_name`. All temporary `.log` and `.dvi` files are removed after use.

`int TeXmerge_View(char *pathname, int waitOption)`

A convenience function to run T<sub>E</sub>X and then run `xdvi`. `waitOption` is one of `TXM_WAIT` or `TXM_NOWAIT`. If `TXM_NOWAIT` is passed, then the current process is forked and then `xdvi` is run in a child process.

`int TeXmerge_Print(char *pathname, char *lpargs, char *output_pathname)`

A convenience function to run T<sub>E</sub>X and then run `dvilj`. If `lpargs` is non-`NULL` then it is used as switches for the `lp` command and the resulting `.lj` file will be queued for printing via the `lp` system. The pathname of the resulting `.lj` is returned in the character array pointed to by `output_pathname`.

```
char *TeXmerge_MakeBarcodeStr(char *composite, TeXmergeName_t *array, int count);
```

Encodes the comma-separated list of data names in `composite` and corresponding values in a character string. It does the encoding by converting the name/value pairs to a Python dictionary object and passing it to the Python `barcodeUtil` module's `encode` function.

```
TeXmergeName_t *TeXmerge_DecodeBarcodeStr(char *str, int *count, int *ret);
```

Decode a barcode string and return its contents as an array of `TeXmergeName_t` structures. Caller is responsible for disposing of the array with `TeXmerge_FreeNames()`. If an error occurs, this function returns `NULL` and returns an error code in the integer pointed to by `ret`. On successful return a pointer to the array is returned, the number of elements in the array is returned in the integer pointed to by `count` and 0 is returned in the integer pointed to by `ret`.

```
char *TeXmerge_MakeTeXBarcodeMacro(char *name, char *composite, TeXmergeName_t *array, int count);
```

Build a snippet of `TEX` code encoding the values from `array` for names listed in comma-separated list passed in `composite` appropriate for printing with a Code 2-of-5 barcode font. The name of the macro will be the string value pointed to by `name`. The returned value is in a static `char` array; it must not be freed and will be overwritten on subsequent calls.

```
char *TeXmerge_GetErrorString(int)
```

This functions returns a character string description corresponding to the passed integer value.

# REVTeX version 4.0, an authoring package by the American Physical Society

Arthur Ogawa

TeX Consultants, Three Rivers CA 93271, USA

[ogawa@teleport.com](mailto:ogawa@teleport.com)

## Abstract

The American Physical Society has just released a new version of their REVTeX authoring package, REVTeX version 4.0. The `revtex4` document class for L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> is completely new code, not a rewrite of REVTeX version 3.1. Those preparing electronic submissions to APS journals, like Physical Review, to American Institute of Physics journals, or to Optical Society of America journals will use `revtex4`, available via <http://publish.aps.org/revtex4> as well as on CTAN.

Other journals and societies may create their own plug-in customizations of REVTeX, as has been done by, e.g., the Optical Society of America.

## Introduction

The `revtex4` document class allows you to prepare a L<sup>A</sup>T<sub>E</sub>X document suitable for electronic submission to any journal of the American Physical Society, or any journal of a participating society.

The formatting details connected with a particular target journal are entirely taken care of by a single document class option, the *journal substyle*. For example, a submission to Physical Review Letters would contain the simple statement:

```
\documentclass[prl]{revtex4}
```

If instead targetting the Reviews of Modern Physics, `prl` is changed to `rmp`, without further alteration. Within the document itself, the syntax is unaltered.

REVTeX therefore establishes a single set of markup that can be used for any of a large and growing number of journals in their electronic submissions programs. In the APS, REVTeX documents are destined for conversion to SGML, a purely descriptive markup scheme, therefore their emphasis is on L<sup>A</sup>T<sub>E</sub>X documents that avoid procedural markup.

The `revtex4` document class is modeled after L<sup>A</sup>T<sub>E</sub>X's `classes.dtx`-based document classes, thus writers familiar with L<sup>A</sup>T<sub>E</sub>X's `article` class will find it easy to adopt REVTeX.

However, REVTeX has additional descriptive markup tools allowing for better structuring of the information on the title page, in the document body, and within the bibliography. It also has options to conveniently compose the material in a format similar to that of the target journal.

## Availability and installation

REVTeX is distributed in ready-to-install form, has complete source with prebuilt documentation, and includes with user documentation and sample files. Its distribution point is <http://publish.aps.org/revtex4> and it also appears on the Comprehensive T<sub>E</sub>X Archive Network (<http://tug.org>) in the directory [tex-archive/macros/latex/contrib/supported/revtex](http://tug.org/tex-archive/macros/latex/contrib/supported/revtex). It is available under the L<sup>A</sup>T<sub>E</sub>X Project Public License (LPPL).

Installation is simple if your T<sub>E</sub>X system uses the T<sub>E</sub>X Directory Structure: REVTeX files are installed in just two locations:

- Macros in `texmf/tex/latex/revtex4`, and
- BIB<sub>T</sub>E<sub>X</sub> styles in `texmf/bibtex/bst/revtex4`

## Relationship to REVTeX 3

The `revtex4` document class for L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> carries almost the entire feature set of the outdated REVTeX 3.1 (which was a L<sup>A</sup>T<sub>E</sub>X 2.09 style), with greater convenience and considerably more powerful formatting. All of the special characters accessible with version 3 have been retained, along with special features, like tables that can break over pages.

## Compatibility with L<sup>A</sup>T<sub>E</sub>X packages

REVTeX is compatible with the American Mathematical Society 2.0 packages `amsfonts`, `amssymb`, and `amsmath`.

REVTeX is compatible with the L<sup>A</sup>T<sub>E</sub>X required packages `array`, `dcolumn`, `graphics`, and `graphicx`,

as well as such popular L<sup>A</sup>T<sub>E</sub>X extensions as `url` and `hyperref`.

Other well-behaved L<sup>A</sup>T<sub>E</sub>X packages are likely to be compatible with `revtex4`. However, compatibility will be problematic with packages that extend L<sup>A</sup>T<sub>E</sub>X in ways already provided for by REV<sub>T</sub>E<sub>X</sub>.

REV<sub>T</sub>E<sub>X</sub> is *not* compatible with the packages `multicol`, `cite`, `endfloat`, and `float`.

REV<sub>T</sub>E<sub>X</sub> is compatible with the `longtable` package, and besides repairing some of that venerable package's bugs, extends it to work properly in a two-column page layout: you can create tables that break over columns if need be.

### Extra convenience in formatting

Title page information, in the past restricted to, e.g., `\title`, `\author`, and `\thanks`, now accommodate in a descriptive way all of the required features of academic journals, such as `\collaboration`, `\email`, `\altaffiliation`, `\homepage`, and `\affiliation`. This novel feature originated with David P. Carlisle.

The `revtex4` class is capable of all of the features of L<sup>A</sup>T<sub>E</sub>X's `twocolumn` option, but additionally allows switching to- or from a one-column page layout anywhere on the page. There is an environment, `\begin{widetext}`, expressly for the purpose of presenting extra-wide math displays that interrupt a two-column layout.

A number of bugs in L<sup>A</sup>T<sub>E</sub>X have been fixed, among them the infamous `\begin{eqnarray}` spacing problem.

Patrick Daly's `natbib` package is always loaded by `revtex4`, and his `custom-bib` package has been used in the preparation of BIB<sub>T</sub>E<sub>X</sub> styles for APS journals. If you have been using BIB<sub>T</sub>E<sub>X</sub> or `natbib`, you will find REV<sub>T</sub>E<sub>X</sub> a familiar environment.

### Power under the hood

The `revtex4` class incorporates two new extensions to the L<sup>A</sup>T<sub>E</sub>X kernel, `ltxutil` and `ltxgrid`, which are available separately for public use under the LPPL. The latter package, based on the work of William E. Baxter, provides a completely rewritten output routine for L<sup>A</sup>T<sub>E</sub>X, one that has far fewer limitations, and which fixes many of L<sup>A</sup>T<sub>E</sub>X's bugs.

### Adaptability

The `revtex4` class's use extends far beyond APS and OSA journals; participants in our beta testing program have employed it for submissions to other journals and for the production of monographs, conference proceedings, and more. Also, the `revtex4` document class provides an extensible architecture for other societies and journals to use: all of its APS-specific features are collected into a

“sub-package”, called `aps.rtx`. Anyone wishing to customize `revtex4` to their own journal is encouraged to write their own `.rtx` file.

### Class options

The document class options of REV<sub>T</sub>E<sub>X</sub> are organized into three categories.

**Society and Journal substyles** An option such as `prl` specifies the target journal of your document. For most purposes, specifying the journal is sufficient, with no other options needed. At present `revtex4` has about a dozen journal substyles.

**Processing options** REV<sub>T</sub>E<sub>X</sub>'s option `preprint` declares that the document should be formatted appropriate to copyediting, with generous leading, larger fonts, and wide margins. An alternative, `twocolumn`, implies closer adherence to the journal's typeset appearance. Another, `lengthcheck`, attempts an even closer adherence, to the point of allowing you to make a good prediction of the number of pages for the published article.

**Simple options** The option `bibnotes` specifies that title block notes (otherwise formatted as footnotes on the title page) are to appear within the bibliography, a choice common in APS journals. This option is *simple* in that it controls a single aspect of document processing.

Other simple options include `footinbib`, which directs footnotes into the bibliography, `endfloats`, which collects figures and tables at the end of the document (a format used in older preprint styles),

A large set of options exerts control over the way the title block is formatted. For instance, the `groupedaddress` option collects authors above a common affiliation set; alternatively `superscript-address` lists each individual affiliation just once, and ties an author to her related set of affiliations with a set of numerical superscripts, a format common in APS journals. There are several dozen simple options in `revtex4`.

In summary, a processing option will effectively invoke multiple simple options, and a journal substyle will invoke a set of such options appropriate to the target journal.

### Specifying the title page

After the `\begin{document}` statement, a simple document would have a set of title block statements like:

```
\title{My paper}
\author{Me}
\affiliation{%
  Domain University\}
```

```
Somewhere, US
}
```

A more complex author list would have one or more `\author` statements followed by one or more `\affiliation` statements:

```
\title{My paper}
\author{You}
\author{Me}
\author{Her}
\affiliation{Here}
\affiliation{There}
\affiliation{Everywhere}
```

This can be thought of as an author group: an affiliation set with a related author list. (An author lacking an affiliation would be followed by a `\noaffiliation` statement.)

An even more complex author list would have multiple author groups, and it is not uncommon for a particular affiliation to appear in more than a single author group. (In `revtex4` it is essential that each `\affiliation` statement for a particular institution to be the same in each instance!)

Furthermore, an author, title, or the first instance of an `\affiliation` may be followed by an arbitrary set of `\homepage`, `\email`, `\thanks`, or `\altaffiliation` statements, e.g.,

```
\author{Me}
\altaffiliation{Here}
\email{Me@hereur.edu}%
```

This markup scheme promises to encompass any sort of author list a journal might practically encounter, while still allowing the title block to be formatted using either the `groupedaddress` or the `superscriptaddress` form. It will be interesting to see if this promise is fulfilled.

### Access to special features

**More robust handling of floats** An aggravating problem not uncommonly encountered by L<sup>A</sup>T<sub>E</sub>X users is the situation where an overly tall figure or table fails to be placed until the end of the document, effectively carrying with it all following floats of the same kind. I call such a situation a *stuck float*. Thanks to the low-level procedures of the `ltxgrid` component of `revtex4`, this state can be detected and emergency action taken. The `floatfix` class option enables this emergency processing.

A second, related disaster is heralded by the L<sup>A</sup>T<sub>E</sub>X message “Too many unprocessed floats”. This situation readily occurs when a document is thick with figures or tables: L<sup>A</sup>T<sub>E</sub>X’s queue of float registers fills up before any can be placed. Here, too, `floatfix` enables emergency processing that, at the

very least, allows T<sub>E</sub>X to complete the typesetting of your document without halting.

Both of these features are available to any document class that incorporates `ltxgrid`, as `revtex4` does.

**Switching between page layouts** The `ltxgrid` package allows one to interrupt a two-column page with full-page-width material, without starting a new page. Likewise, a two-column page layout may be begun on a page with existing full-page-width material. (These features, not available in L<sup>A</sup>T<sub>E</sub>X, will be familiar to users of the `multicol` package.)

Markup that invokes this process is not at all descriptive, so REVT<sub>E</sub>X avoids the mistake of packaging this capability up as a L<sup>A</sup>T<sub>E</sub>X environment. Instead, the low-level tools are invoked by descriptive markup elements, for instance, `\begin{bibliography}`. Nonetheless, two commands are made available to the user, to allow access to the functionality in case of need. The `\twocolumngrid` command begins a two-column page layout, the `\onecolumngrid` command returns to a full-page-width layout.

### Provenance

The `revtex4` document class is the responsibility of the American Physical Society’s Mark Doyle. Under contract to the APS, the first draft was written by David P. Carlisle, with continued development by the author. The APS also supports the work of Patrick Daly, depending as it does on `natbib` and `custom-bib`.

The APS is firmly committed to `revtex4`’s utility as a convenient way to compose electronic submissions for its journals and those of other societies. In addition, `revtex4` is intended to serve as an effective vehicle for other projects that users may conceive, with an emphasis on compatibility with L<sup>A</sup>T<sub>E</sub>X and extensions thereto, and on support for descriptive markup, hypertext, and multipurposing.

Bugs and problems can be reported to [revtex@aps.org](mailto:revtex@aps.org). Any incompatibility with a package declared to be compatible with `revtex4` will receive priority attention.

### The future

Plans for `revtex4` include further modularization, splitting out its author/affiliation procedures as a L<sup>A</sup>T<sub>E</sub>X package, and extending its ability to format journal pages so that it can be used to prepare APS journal articles for delivery over the internet.

I hope that `revtex4` fits into your future plans, either as an authoring tool, or as a platform for your society’s journals.

# The T<sub>E</sub>X History Panel

Anita Z. Schwartz (chair)

User Services, Information Technologies, University of Delaware  
[anita@udel.edu](mailto:anita@udel.edu)

**Anita Schwartz** chaired the T<sub>E</sub>X History Panel.

She introduced herself as a member of TUG for 15 years and has been involved in supporting T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X at the University of Delaware for past 15 years. She has developed a number of macros specifically for professors writing books and styles/classes for theses and dissertations. As part of the history of T<sub>E</sub>X, she provided old issues of *TUGboat* (1980), some special issues, *T<sub>E</sub>Xniques*[4], which led to the current conference Proceedings issues of *TUGboat*, EuroT<sub>E</sub>X '92 (the only hard bound issue) and t-shirts to view from over the years. Also three framed posters by Professor Alban Grimm (see Poster Exhibition article in *TUGboat*, Volume 20 (1999)) were on display.

**Barbara Beeton** introduced herself as one of the first users of T<sub>E</sub>X. She is a true T<sub>E</sub>X Historian. She provided two original T<sub>E</sub>X books that she used to learn the T<sub>E</sub>X system and spoke about her early experiences with T<sub>E</sub>X. She talked about her communications with Don Knuth over the years. Also Barbara modeled one of the original light blue t-shirts with black lettering `{\TeX\ User Group}`.

**Nelson Beebe** spoke about his original interest and involvement with T<sub>E</sub>X and how he got to know about it. His involvement with T<sub>E</sub>X began in the spring of 1979, when during a trip to the Bay Area, he was invited to Xerox PARC Laboratory to hear a talk given by Don Knuth on the new typesetting system, called T<sub>E</sub>X, that Don was working on. This was of immediate interest to him, because he had long been interested in software portability, and had long wanted to have portable documentation for software. In the mid-1970s, Nelson developed a prototype document formatting system, called DOCUMENT, with many similarities to UNIX `nroff`, although he had not heard of `nroff` at the time, since it too was under development. A check of Gehani's book[2] shows a report dated 1977 by Joseph Ossana "*nroff/troff User's Manual*", and a 1975 paper on `eqn`[3], so evidently

`nroff/troff` goes back to at least about 1975. These were more ambitious than DOCUMENT, having the advantage of a real typesetter, and the implementation language C, which did not become widely available outside Bell Labs until the early 1980s. [For portability reasons at the time, Nelson was restricted to Portable Fortran, and later, translated the code to a very nice structured Fortran preprocessor language, SF-TRAN3, developed at the Jet Propulsion Laboratory, Pasadena, CA.] When Nelson saw what Don had done, he was very excited, because here was a vastly better system than anything he'd seen before, and furthermore, it ran on the same hardware that he used at Utah (a DEC-20 running TOPS-20). Importantly, it could do mathematics! It wasn't long before Nelson fetched over Don's early T<sub>E</sub>X '78 distribution from Stanford and made early experiments with dot matrix printers (the Florida Data and the Printronix) to see whether cheap draft output might be possible, because he could not find US\$20K to buy an early model laser printer from Imagen (a Stanford spin-off). He based his work on what Mark Senn had done (at Purdue) for the BBN BitGraph terminal, because he happened to have one. This work was later chronicled in a *TUGboat* paper [1]. It wasn't until 1984 that Nelson finally was able to get a real laser printer that could produce decent output. Nelson has been able to attend all but two of the annual TUG meetings, including the first one, and today, T<sub>E</sub>X is very much a part of his everyday life.

**Hans Hagen** mentioned he felt more like a newcomer, but was willing to note predictions about the future history of T<sub>E</sub>X. He noted that before the World-wide Web was widely used, it was not trivial to get T<sub>E</sub>X up to speed and up to date when you were not part of the "university scene." He had to buy T<sub>E</sub>X programs, buy patterns, tweak English tuned files that he didn't understand, etc. It was one of the things that drove him to develop his own macros: he was

simply unaware of the things happening and the T<sub>E</sub>X community, because everyone expected everyone to be online. He noted later, post-conference, that he even bought back issues of *TUGboat*, but most went unread since he lacked any reference point and understanding of T<sub>E</sub>X cum suis; it's only recently that he has an idea how things were back in the T<sub>E</sub>X 80's.

**Martin Schroeder** discussed the real history of T<sub>E</sub>X and requested that if you had knowledge of important T<sub>E</sub>X dates to submit them to be placed in the T<sub>E</sub>X calendar. He also mentioned important recent and upcoming birthdays of Don Knuth and Leslie Lamport.

## References

- [1] Nelson H. F. Beebe. Low-cost downloadable font devices. *TUGboat*, 4(1):11–12, April 1983.
- [2] Narain Gehani. *Document Formatting and Typesetting on the UNIX System*. Silicon Press, 25 Beverly Road, Summit, NJ 07901, USA, 1986. ISBN:0-9615336-0-9.
- [3] Brian W. Kernighan and Lorinda L. Cherry. System for typesetting mathematics. *Communications of the ACM*, 18(3):151–157, March 1975.
- [4] T<sub>E</sub>Xniques, Publications for the T<sub>E</sub>X community. T<sub>E</sub>X User Group, <http://www.math.utah.edu/pub/tex/bib/index-table-t.html#texnique>, 1988–1990.

# Using T<sub>E</sub>X for High-end Typesetting

Hans Hagen

Pragma ADE, GH Hasselt, The Netherlands

[pragma@wxs.nl](mailto:pragma@wxs.nl)

<http://www.pragma-ade.com>

## Abstract

The fact that T<sub>E</sub>X is already quite old does not change the fact that it can still pretty well compete with high-end desktop publishing programs. The fact that XML has simplified coding, if only by posing limitations, gives T<sub>E</sub>X an even stronger position in today's typesetting arena. In this presentation I will discuss the challenges that come to us, present a couple of tools, and demonstrate what we may expect from T<sub>E</sub>X today. I will also demonstrate that a modern T<sub>E</sub>X-like pdfT<sub>E</sub>X can give today's T<sub>E</sub>X users a pretty strong position.

# TeX — a mass-market product? Or just an image in need of a makeover?

Peter Flynn  
Silmaril Consultants  
[peter@silmaril.ie](mailto:peter@silmaril.ie)

## Abstract

Despite widespread acceptance in the scientific field and with some publishers, and recent advances in the humanities, TeX-based systems have largely been publicised by word of mouth, and no-one can tell how many users there are. The commercial versions advertise where they deem fit but generic publicity for TeX is not common.

Users can be TeX's best advocates, but formal training is rare. Users learn mostly from colleagues — themselves often ill-taught — and acquire bad habits which are hard to overcome. The results are often responsible for the poor image TeX has had among most printers and publishers. If TeX systems are so good at typography, the question is often asked, why does the output look so poor? Although TUG runs courses, it is hard to cover such a geographically dispersed user population.

Support for TeX via the Internet is excellent, often far superior to that of other products, but there is always a need for more introductory documentation aimed at the beginner and the non-scientific user. Some installation help is also still needed, especially for the first-timer: the assumption that everyone is already skilled in the principles of computing no longer holds.

This paper argues that the biggest need is for distributable publicity targeted at identifiable markets backed up by presentable documentation. More of the power of L<sup>A</sup>TeX should be made use of in creating these documents if it is to regain its market share.

## Bad habits die hard

One of the early joys of L<sup>A</sup>TeX was the speed with which a new user — assuming some competence in using a text editor — could learn half a dozen commands and create a paper which looked infinitely better than anything a wordprocessor of the time could produce. Passing on the information that L<sup>A</sup>TeX could do this, even with mathematics, was probably the prime factor in its widespread adoption in the scientific community. Questions could be answered online in the newsgroups and mailing lists, and the TeX Users Group had its annual conference and thousands of members. The academic and research community at that time formed a self-contained mass market.

Synchronous typographic editing, popularised by early DTP systems and graphical wordprocessors did not initially affect TeX-based systems, as the formatting and mathematical capabilities of what were coming to be called 'graphical' systems were primitive by comparison with TeX. Faster proces-

sors enabled more sophisticated interfaces, however, and encouraged the perception that what you saw was all there was to it.

L<sup>A</sup>TeX remains the most sophisticated programmable non-proprietary system, but the prevalent text-mode interface and the perceived difficulty of learning textual commands have long been recognised as major discriminants in a user's choice of application, despite at least half a dozen systems over the years using TeX in a synchronous graphical mode.

**Training and learning** Notwithstanding the continuing availability of training courses in L<sup>A</sup>TeX (both publicly by TUG and privately within user institutions), many users still acquire their knowledge of how to use the system from colleagues in laboratories, classrooms, libraries, and offices. Misconceptions persist, as anyone involved in institutional or newsgroup support of TeX can verify from experience. Numerous well-written documents exist on the CTAN servers which can be used in self-study

to learn how to use  $\LaTeX$  properly, but many of the answers to frequently-asked questions have to be repeated regularly on the newsgroup [comp.text.tex](http://comp.text.tex) as well as appearing in the comprehensive FAQs, and the training technique known as ‘sitting by Daisy’<sup>1</sup> continues to predominate.

This comes to a head not in the production of private or internal documents but in the submission of  $\LaTeX$  files to publishers. Although many of them have used  $\LaTeX$  for years, most would claim that it is only because of pressure from authors who insist on supplying it. Few publishers now employ  $\LaTeX$  experts to undertake the syntactical correction of  $\LaTeX$  files necessitated by the authors’ misunderstandings, and some even find it more profitable to print out an author’s document with errors and have the entire thing retyped from scratch in *Word* by keyboarding companies in the Far East.

Most of the publishers’ misconceptions (‘ $\LaTeX$  has only one font’, ‘ $\LaTeX$  can’t do graphics’, ‘ $\LaTeX$  is only for mathematics’, et cetera ad nauseam) stem from their experience of authors’ own misconceptions and lack of training. Given the unfortunate look and feel of the default formats (book, article, report, and letter), and the early (pre- $\LaTeX 2_\epsilon$ ) difficulty of using anything other than Computer Modern, the publishers’ view that  $\LaTeX$  was typographically inadequate is perhaps understandable.

Authors claim, with some validity, that they are not to blame for this, as it is their job — for example — to perform particle physics experiments, write business reports, or analyse the linguistic aspects of a manuscript, rather than to become typesetters. However, it remains true that one of the major advantages of using  $\LaTeX$ , with its wide range of packages, is that the authors precisely do *not* have to become typesetters, any more than they would have to using a word processor (although in both cases there is an assumption that the author is familiar with the standard requirements of publishers for accuracy, consistency, and familiarity with the conventions of publication).

**Documentation** We can see, therefore, that some of the negative publicity encountered by proponents of  $\LaTeX$  derives from the exposure of the user (whether author or publisher) to examples of  $\LaTeX$  which abuse or misuse the language because of a lack of understanding engendered by inadequate training. The same claim, however, can also be made of any sophisticated system used by untrained

operators. The scientific, business, or humanities author who tries to pick up the rudiments of Quark *XPress* or *Framemaker* from similarly untrained colleagues is likely to encounter similar difficulties. The problem of training is a large one, given that the first task is to persuade sufficient potential users even to consider using the system in the face of the negative image so often encountered.

More and better documentation can probably help here, especially if sufficient attention is paid to its formatting so that it creates a superior image to that presented by other systems, and to its content so that it does not presume an unwarranted familiarity with any particular discipline.

Using  $\LaTeX$  it is perfectly possible to create styles which reflect current trends in user documentation. There is a tendency or desire, however, on the part of some documentation authors to stick with the default formats, and it is unclear if this stems from a lack of experience or a reluctance to introduce what others may see as unneeded dependencies (on PostScript, for example). There are of course occasions when the defaults are desirable, such as the documentation for packages, but there must surely be many more when a different look and feel would be more conducive to persuading the potential user in favour of  $\LaTeX$ .

It is this initial task of *persuading* which seems to offer the best hope of countering the current antipathy or apathy, by way of giving  $\LaTeX$ ’s image the makeover referred to in the subtitle.

## Publicity

The meetings of the TUG Publicity Committee have from time to time considered the production of large-scale generic marketing material, but so far as I am aware (and as a member the fault is as much mine as anyone’s), previous attempts have foundered due to lack of personal time and the problems of agreeing on the content. The TUG office has produced brochures and leaflets, but these have tended to be targeted at specific TUG-related objectives such as increasing membership or conference attendance, rather than simply broadening the appeal of  $\TeX$  itself.

The hard work which the many people involved have put into efforts to date has not gone unheeded, however. The assorted minutes of meetings and the present author’s own notes have been used to provide a framework for an experimental document<sup>2</sup>

<sup>1</sup> Informal learning from a co-worker, derived from the method of training of company telephone operators in the 1930s

<sup>2</sup> It should be noted that the current experiment is not a part of any TUG activity, as it was originally designed as an internal project within the author’s institution.

in the form of a brochure aimed at publicising L<sup>A</sup>T<sub>E</sub>X with specific reference to:

- portability, persistence, and durability;
- ease of use and flexibility in application;
- widespread support;
- commercial and free versions;
- examples of real-life applications.

The objective of this experimental leaflet is to test its effectiveness at generating interest in L<sup>A</sup>T<sub>E</sub>X among non-users and re-generating interest among those with misconceptions. As formal in-depth market research at the level normally conducted by full-scale academic or commercial research projects is out of the question for financial reasons, the participation of individual L<sup>A</sup>T<sub>E</sub>X consultants as well as TUG and the online TeX community will be sought when editing of this first draft is finished in October 2001.

The current draft is implemented as a 4–page brochure which will print on A3 or A4 paper folded once to A4 or A5 respectively. It includes the following content:

- a brief explanation of what L<sup>A</sup>T<sub>E</sub>X is and why the user might need it;
- a list of the principal features;
- quotes from commercial and academic users about its usefulness;
- samples of different kinds of output with brief comments about them, specifically
  - a reset fragment of the 42–line Bible;
  - font samples;
  - mathematics;
  - tabular setting;
  - vector and bitmap graphics;
  - automated cross-referencing.
- information about where to obtain a copy;
- details of vendors, platform support, and technical requirements;
- information about networked support and the TeX Users Group;
- space for the contact details of a local distributor, user group, consultant, vendor, etc.

The quotes and availability information are currently sourced from the present author’s institution, where the leaflet has so far been distributed on a pilot basis at meetings on academic publishing and the reproduction of study texts, but other more

globally representative quotes and details can of course easily be used to replace these, within the space available.

As a first step in extending the pilot phase outside the author’s institution, criticisms, suggestions, and replacement text are actively sought for the four main categories of content:

- the quotations from users;
- the illustrations;
- the font samples;
- the marketing and descriptive text.

The effect of using the current draft as an internal pilot has been marked. The leaflet was distributed to about 200 academics at a variety of document-related meetings within the author’s institution and some 20 individuals were sufficiently interested to ask for more information and the installation of the software to test (the TeX Live 5 CD-ROM was made available). A further exposure by the author’s consultancy to four clients with specific typesetting requirements has resulted in two of them installing the software for evaluation.

The current version can be found in <http://www.silmaril.ie/downloads/documents/leaflet.pdf> and <http://www.silmaril.ie/downloads/documents/leaflet.ps.gz>, and copies will be available for inspection at the TUG annual conference in the University of Delaware in August 2001.

## Conclusions

It is not possible to say from the limited pilot information at this stage if this approach will be successful, as the individuals involved were to some extent a self-selected group with an existing express desire to seek alternative solutions to their current systems.

As explained earlier, the prime objectives are to generate interest in L<sup>A</sup>T<sub>E</sub>X and overcome misapprehensions about it. A more widespread test is needed against groups who have *a*) no previous knowledge of the existence of L<sup>A</sup>T<sub>E</sub>X, or *b*) previous (poor) experiences of using L<sup>A</sup>T<sub>E</sub>X. It should be noted that the author is not a professional designer, so the current implementation should not be taken as indicative of any future version.

Further development of this concept therefore rests on there being sufficient interest among the wider community.

# L<sup>A</sup>T<sub>E</sub>X for Windows – A User’s Perspective

David M. Tulett

Faculty of Business Administration  
Memorial University of Newfoundland  
[dtulett@mum.ca](mailto:dtulett@mum.ca)

## Abstract

L<sup>A</sup>T<sub>E</sub>X is a technically brilliant package for typesetting, but in the Windows world Microsoft Word continues to be widely used.<sup>1</sup> From the perspective of an end-user of these products, the relative strengths and weaknesses of L<sup>A</sup>T<sub>E</sub>X and Word are examined.

## Introduction

Among operating systems for personal computers, the Microsoft Windows family (95/98/ME/NT/2000) has over 90% of the market, and it is on this market on which this paper focuses. For printing text on to paper, most users of Windows use Microsoft Word. Word can be bought as a standalone package, or it can be bought as part of the Office suite package, which contains many popular programs such as Excel and PowerPoint. There are other word processing packages for Windows users, such as Corel WordPerfect, but this package mostly appeals to those who have always used it and have remained loyal to it, or to those who are very price-sensitive. There are also desktop publishing systems such as Corel Ventura, Quark Express, and Adobe PageMaker. These latter programs are quite expensive, and the technical advantages over Word have diminished as Word continues to add new features. These word-processing and desktop publishing systems are all WYSIWYG (what you see is what you get). While to a certain extent all these packages compete with L<sup>A</sup>T<sub>E</sub>X, in this paper the scope is restricted to a comparison between the hugely-successful Word and L<sup>A</sup>T<sub>E</sub>X, which is by comparison a niche product.

One major advantage of L<sup>A</sup>T<sub>E</sub>X is that it does a very good job of typesetting mathematics. Even something as simple as  $x + y = z$  looks better in L<sup>A</sup>T<sub>E</sub>X than it does in Word. The difference between the two approaches widens considerably when typesetting something more complex, such as:

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix}^{-1} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

<sup>1</sup> There are many references in this paper to trademarks or registered trademarks. These have been capitalized.

The second major advantage (though it may seem like a disadvantage at first) of L<sup>A</sup>T<sub>E</sub>X over Word is its different paradigm for document creation. In contrast to the WYSIWYG approach, the user of L<sup>A</sup>T<sub>E</sub>X specifies the structure and lets L<sup>A</sup>T<sub>E</sub>X handle the design of the document. The advantage of this approach over WYSIWYG has been extensively described – see, for example, Love [10] for a discussion of L<sup>A</sup>T<sub>E</sub>X versus Word, and Taylor [12] for a critique of WYSIWYG in general.

This paper addresses the issue of why L<sup>A</sup>T<sub>E</sub>X has nowhere near the numbers of users that Word has. I know of many people with technical backgrounds who have never tried L<sup>A</sup>T<sub>E</sub>X, and indeed I know of some who have given up on L<sup>A</sup>T<sub>E</sub>X and have switched to Word. This paper is written from the perspective of an ordinary user of L<sup>A</sup>T<sub>E</sub>X, not someone who is a computer programmer. The rest of the paper is organized as follows. First, a personal historical background of how I became interested in L<sup>A</sup>T<sub>E</sub>X is presented. Secondly, a comparison of L<sup>A</sup>T<sub>E</sub>X and Word is made. Thirdly, we look at other options. Finally, recommendations are given.

## Historical Background

**The Dark Ages (Before L<sup>A</sup>T<sub>E</sub>X)** In this section I wish to explain how I became interested in L<sup>A</sup>T<sub>E</sub>X. During my engineering undergraduate education (1971-75), all assignments involving mathematics (except my thesis) could be hand-written, and were. For prose essays, I used an electric typewriter. In 1975, I wrote my undergraduate thesis on a typewriter, using a special ball to handle mathematical symbols. I was glad that this was the only requirement for this technology. After working for several years, I entered a doctoral program in management science, and wrote my dissertation in 1985. While there were developments in the T<sub>E</sub>X world at that time, they were not available to me.

What was available (on the university’s mainframe) was a program for word-processing which had limited ability to write mathematical text. Writing something like  $\alpha = \beta + \gamma$  was easy to do, but creating a display equation was not. For example, the L<sup>A</sup>T<sub>E</sub>X expression `\[ y = \sum_{i=1}^n x_i \]` creates

$$y = \sum_{i=1}^n x_i$$

Making this equation back then required writing an  $n$  on one line,  $y = \sum x_i$  on a second line, and finally writing  $i = 1$  on a third line. This would have to all be set in a non-proportional font and then trial-and-error would have to be used to make sure that all the symbols lined up correctly. It goes without saying that putting the three lines together produced an equation which looked terrible. At the time I thought that this word processor was a tremendous improvement over my 1975 experience, but of course hoped that something better would become available.

In 1985 I began working for my current employer. In the previous year the Faculty of Business Administration had acquired a large number of Macintosh computers for faculty and staff, and I began to use the Mac for everyday things like making up tests. At first MacWrite was the only word processing program, but then Microsoft Word was released, long before it came to Windows. In 1988 a colleague and I decided to make up a set of notes for a course which we taught. I decided to have a look at what was available.

**The Middle Ages (L<sup>A</sup>T<sub>E</sub>X 2.09)** In searching for a software package to help us write our book, I did some reading on the WYSIWYG packages, and saw references to L<sup>A</sup>T<sub>E</sub>X, which was then in version 2.09. We decided to adopt it, mostly because of its ability to create nice-looking equations. The other advantage of L<sup>A</sup>T<sub>E</sub>X, that it used logical rather than visual design, seemed at the time to be a disadvantage – after three years of using a Mac it was hard to leave an established paradigm. This was especially true given how we printed the dvi files. We had terminals to the mainframe, which of course let us view the ASCII-based tex file, but there was no way to preview the dvi file. The 300 dpi laser printer was located in another building about 300 metres away.

To learn L<sup>A</sup>T<sub>E</sub>X, I purchased the first edition of Lamport’s book [8]. I read it over a weekend to get the general idea of what it was all about, and then read it again trying to learn the content. Even with this preparation, I needed to have the book next to my side for the first several months. (Even

today, I often need to consult a book for a particular problem.) By contrast, I learnt how to use Word without ever reading the manual.

During this period with a terminal on my desk, my 512K Mac had become obsolete, and I obtained an IBM PC mostly to use a lot of common business software such as the then-popular Lotus 1-2-3. The university had a site license for a commercially-made version of L<sup>A</sup>T<sub>E</sub>X, and for about<sup>2</sup> C\$50 I had it installed on my machine. This was still 2.09, but now I could view dvi output on my monitor.

One nagging problem at the time was what to use as an editor for creating the tex file. I tried several things, but eventually settled on using WordPerfect, simply saving the file in ASCII format. (I needed WordPerfect anyway, for communication with non-T<sub>E</sub>X users.) Compared with what I had had only a few years earlier, this setup seemed to be the cat’s meow.

I had become a proficient user of L<sup>A</sup>T<sub>E</sub>X, and now saw limitations in what had once been a package with so many new things. The biggest limitation was graphics. I can remember trying to approximate a parabola by drawing a sequence of short straight lines. Even straight lines had a small finite set of angles from which to choose. For making a problem involving two-dimensional linear optimization, I would make the objective function and the constraints so that when drawn they would be at angles which L<sup>A</sup>T<sub>E</sub>X could handle, which is surely the tail wagging the dog!

**The Renaissance (L<sup>A</sup>T<sub>E</sub>X 2 $\epsilon$ )** Walking through a bookstore in 1995, I came across the second edition of Lamport’s book [9]. After reading it, I followed Lamport’s recommendation to obtain *The L<sup>A</sup>T<sub>E</sub>X Companion* [3], and I inquired at the university about upgrading the software. The company from which the site license had been obtained was no longer selling L<sup>A</sup>T<sub>E</sub>X, and we were left to fend for ourselves. We obtained a two-CD set of the CTAN archive, and my colleague managed to figure out how to use it despite the lack of instructions. The effort to upgrade L<sup>A</sup>T<sub>E</sub>X was in my opinion justified by the new `\qbezier` command, but we soon found other useful features, such as the ability to: print on legal size paper; use colour; import graphics; and import new packages. In particular, I was glad to see the **times** and **mathptm** packages had been made for creating words and mathematical characters in Times-Roman fonts. The Computer Modern fonts

<sup>2</sup> Prices are in US dollars unless indicated as C\$ to mean Canadian dollars. These figures are approximate and may have changed after March, 2001.

which I had used up to this point never looked as nice as Times-Roman.

When I obtained Internet access, I discovered a new world, soon finding the web page for TUG, from which many other resources could be accessed. Joining TUG provided among other things annual editions of the T<sub>E</sub>X Live CD. A colleague in the mathematics department recommended the Professional File Editor (PFE) to me, and this was a substantial improvement over editing files in WordPerfect. Later I found out about WinEdt, and for only \$40 I then had something even better for creating tex files.

One problem that remained was that I was a L<sup>A</sup>T<sub>E</sub>X user in a predominantly non-L<sup>A</sup>T<sub>E</sub>X environment. I could not expect, for example, my students (business majors) to be able to read dvi files. However, on the web page for the commercial vendor PCT<sub>E</sub>X, I found a free dvi viewer called *DVIscope*, which I set up on my computer. This viewer had all sorts of new features, such as magnification and the ability to print only selected pages, and I soon recommended this to my students. However, installing the viewer was not easy. Then I discovered Portable Document Format, and now instead of having the students install the dvi viewer, I recommend the Adobe Acrobat reader, which is useful for other courses and indeed other purposes. Of course, the reader only became useful because of the creation of the *pdftex* program and other ways of creating pdf files. Related to the use of *pdftex* are the *hyperref* and *url* packages for creating internal and external references.

To learn how to use some of features described here, I added three more books to my collection: *The L<sup>A</sup>T<sub>E</sub>X Graphics Companion* [5]; *A Guide to L<sup>A</sup>T<sub>E</sub>X* [7] for a more recent general-purpose book; and *The L<sup>A</sup>T<sub>E</sub>X Web Companion* [4]. Along with *L<sup>A</sup>T<sub>E</sub>X: A Document Preparation System* [9] and *The L<sup>A</sup>T<sub>E</sub>X Companion* [3], these five books comprise my current L<sup>A</sup>T<sub>E</sub>X library. An additional resource for the basics of L<sup>A</sup>T<sub>E</sub>X is the *Not So Short Introduction to L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>* available on the Web from [11].

## L<sup>A</sup>T<sub>E</sub>X versus Word

**Are There Problems with L<sup>A</sup>T<sub>E</sub>X?** Much has been said about the long learning curve for L<sup>A</sup>T<sub>E</sub>X when compared with Word. However, this is not all that fair when the two packages are asked to do different things. If all that one wanted to do was write prose, a half-hour spent on learning L<sup>A</sup>T<sub>E</sub>X would be sufficient. To write equations, more time is needed to learn L<sup>A</sup>T<sub>E</sub>X, but then more time would be needed to learn the Equation Editor in Word. Specialized

packages for L<sup>A</sup>T<sub>E</sub>X have their own learning curves, but their equivalents in Word (if they exist) require learning too.

Another complaint about L<sup>A</sup>T<sub>E</sub>X concerns the lack of variety of fonts. This was a very valid concern when all L<sup>A</sup>T<sub>E</sub>X had was Computer Modern. Now, the set of available fonts is adequate for most purposes. Indeed, when using Word with its very large number of fonts, I only use two of them: Times New Roman (for most things); and Courier (when a non-proportional font must be used). For L<sup>A</sup>T<sub>E</sub>X, the free *times* and *mathptm* fonts are adequate for my purposes.

Every other perceived inadequacy of L<sup>A</sup>T<sub>E</sub>X has in my opinion been addressed. The T<sub>E</sub>X live CD and new editors such as WinEdt have improved the user friendliness, and all the new packages have greatly improved the functionality. Being able to write to pdf has improved the accessibility of completed L<sup>A</sup>T<sub>E</sub>X documents to non-L<sup>A</sup>T<sub>E</sub>X users.

**Word – The Ubiquitous (Sub)-Standard** At my office Microsoft Windows 2000 is the “standard” operating system, and Microsoft Office 2000 is the standard applications software. These standard products are provided at no charge to the user, and training and other help is available. Non-standard products like Linux and L<sup>A</sup>T<sub>E</sub>X are permitted, but at the user’s own expense,<sup>3</sup> and with no training or help provided. I understand why some level of standardization is necessary – at one time we had Mac-OS, DOS, and Windows 95, and even within one operating system there would be both WordPerfect and Word. In an environment where documents need to be shared, standardization helps bring order out of chaos. At the same time, it’s hard to see how L<sup>A</sup>T<sub>E</sub>X could flourish in this environment. Among forty-five members of faculty, three use L<sup>A</sup>T<sub>E</sub>X for Windows, and one uses L<sup>A</sup>T<sub>E</sub>X for Linux. On a positive note, Adobe Acrobat Reader has recently become a standard, enabling users of L<sup>A</sup>T<sub>E</sub>X to share files by using pdf format.

At least at the office I am *allowed* to use L<sup>A</sup>T<sub>E</sub>X. The same cannot be said for some professional societies. The Administrative Sciences Association of Canada (ASAC) *requires* that all articles for its annual conference or for publication in the *ASAC Bulletin* be submitted in either Word or WordPerfect. I recently received a call for papers for the ANZAM/IFSAM VIth World Congress (Australia, 2002) with the same requirement.

<sup>3</sup> To be fair, we are given C\$400 per annum which could be used to buy non-standard software, but this money could also be used to buy books or computer peripherals.

Another example of Word being imposed as a standard was when I developed some materials for an on-line course. The details of creating the web pages were being handled by a private company, which needed to have all input in Word. Wanting to be cooperative, I did everything in Word, learning how to use its Equation Editor for the first time. Just as I was finishing this document, everything started to freeze. It seems that I had reached a limit for the number of display equations in a document. I had to convert a couple of display equations to in-text equations to finally finish the document. In a way I’m glad that I had to make this document in Word rather than L<sup>A</sup>T<sub>E</sub>X, because I had a chance to truly evaluate Word as an alternative to L<sup>A</sup>T<sub>E</sub>X. To be fair, there are some good points about Word:

1. Word is very easy to learn. Whenever I’ve needed to learn something specific, such as how to make a footnote, all I’ve had to do is use the pull-down help menu.
2. Fonts are plentiful, and can be switched at will.

Overall, however, I believe that L<sup>A</sup>T<sub>E</sub>X is a superior product for document creation because:

1. L<sup>A</sup>T<sub>E</sub>X handles mathematics well, both in creation of the document and in printing the final product.
2. For everything but very short documents, logical design beats visual design.

**Obtaining Word and L<sup>A</sup>T<sub>E</sub>X** Though there are many retailers of Word, only Microsoft makes it. Word (as part of Office) ships with many high-end personal computers. Many companies have site licenses for Word or Office. For those who need to buy the program, it’s expensive for business use, but very inexpensive for academic use. Where I work, Word 2000 can be purchased for C\$117 (about \$75).

L<sup>A</sup>T<sub>E</sub>X for Windows can be obtained both as freeware and commercial packages. Every member of TUG (currently \$65 per annum for an individual) obtains the latest version of L<sup>A</sup>T<sub>E</sub>X from the annual T<sub>E</sub>X live CD. In addition to obtaining L<sup>A</sup>T<sub>E</sub>X, a text editor is needed (such as the shareware program WinEdt for \$40), and a book such as *A Guide to L<sup>A</sup>T<sub>E</sub>X* [7] (about \$40) should be obtained. In total, the cost to get started is about \$145.

This cost can be reduced, because one could obtain L<sup>A</sup>T<sub>E</sub>X from CTAN for free, but I believe that those who benefit from all the work that goes into the development of L<sup>A</sup>T<sub>E</sub>X should pay something for it, and this something is the annual \$65 cost of belonging to TUG.

At the other end of the scale, one can pay far more than \$65 plus \$40 to obtain a commer-

cial version of L<sup>A</sup>T<sub>E</sub>X with its own editor. The commercial vendors of L<sup>A</sup>T<sub>E</sub>X for Windows as listed on the TUG website are (in alphabetical order): **MicroPress V<sub>T</sub>E<sub>X</sub>**; **PCT<sub>E</sub>X**; **True<sub>T</sub>E<sub>X</sub>**; and **Y&Y**.

The premium packages from these suppliers sell for several hundreds of dollars (though some offer older or more basic packages for much less). I do not own any of these packages, and being puzzled as to why someone would buy one, I wrote to all four companies. The most detailed response was from Y&Y [13], in which the following points for buying a commercial system (particularly theirs) were made: ease of setup; access to support; additional features (e.g., cut and paste to PowerPoint); and better fonts. It would be useful if someone were to properly evaluate all four of these commercially available packages and compare them with each other and with what is on the T<sub>E</sub>X Live CD, but that is beyond the scope of this paper.

**Finding out About Word and L<sup>A</sup>T<sub>E</sub>X** It is probably true to say that every user of L<sup>A</sup>T<sub>E</sub>X has heard about Word. How many Word users have heard about L<sup>A</sup>T<sub>E</sub>X? In my experience, most people have never heard of L<sup>A</sup>T<sub>E</sub>X, let alone have any knowledge about it. This situation may upset us, but it shouldn’t surprise us. Bookstores offer a multiplicity of books about Word, but only a few high-end bookstores carry anything about L<sup>A</sup>T<sub>E</sub>X. Schools will have almost always have either Word or WordPerfect, but rarely have L<sup>A</sup>T<sub>E</sub>X. A new computer may have a word processing package bundled with it, but it won’t have L<sup>A</sup>T<sub>E</sub>X. A word search made in March 2001 on *PC Magazine* shows that the last five mentions of L<sup>A</sup>T<sub>E</sub>X go back to 1997; the last five mentions of Word go back only to the last two issues. Clearly, it is easy to never have been exposed to L<sup>A</sup>T<sub>E</sub>X, and this problem must be addressed.

Even among those who have heard of L<sup>A</sup>T<sub>E</sub>X, I would offer the conjecture that most have never tried it, and I know of some who have tried L<sup>A</sup>T<sub>E</sub>X only to later abandon it.

### Other Options

**LyX** LyX is a program which tries to combine the typesetting ability of L<sup>A</sup>T<sub>E</sub>X with the WYSIWYG feel of Word, though LyX call this WYSIWYM (what you see is what you mean). LyX began on the Unix operating system, but has been ported to other operating systems, and in particular it has been ported to Windows by Claus Hentschel (based on previous work by Steven van Dijk). LyX requires that L<sup>A</sup>T<sub>E</sub>X be installed on the user’s computer, and at the present time there is a laborious process to

get LyX installed and running. If those who use Word are doing so in part because setup is easy, then I don't believe that they will experiment with LyX. As for the established base of L<sup>A</sup>T<sub>E</sub>X users, not having WYSIWY(G or M) on the screen is not a serious disadvantage, as one can always use `pdftex` before the tex file has been completed to see how it looks so far. If the day comes when LyX with all the necessary ancillary programs for Windows comes on a CD with an automatic install feature, then it may well improve the use of L<sup>A</sup>T<sub>E</sub>X by Windows users, but we're certainly not there yet.

**Converters** Another approach is to use a “converter”. Such a program translates the output from a package such as Word into L<sup>A</sup>T<sub>E</sub>X, or *vice versa* [6]. Going from L<sup>A</sup>T<sub>E</sub>X to Word might be useful, for example, for someone who has written a paper in L<sup>A</sup>T<sub>E</sub>X and now wishes to submit it to a journal or conference which requires submission in Word. Nevertheless, to me it seems like changing wine into water, because all the elegant mathematical typesetting is lost. Going in the Word to L<sup>A</sup>T<sub>E</sub>X direction, however, could be useful to someone who wants the ease of Word combined with the functionality of L<sup>A</sup>T<sub>E</sub>X. The Word2T<sub>E</sub>X \$45 shareware program [2] performs such conversions, but not always flawlessly. The examples provided on the website of files in .doc, .tex, and .pdf formats are quite impressive. However, when I used the program to convert a Word file that I had created, the converter made incorrect guesses about the `\section` and `\subsection` commands. I would make the conjecture that the Word2T<sub>E</sub>X program works well when the original Word document is well structured (perhaps by using styles), but flounders when the original document has been made completely in the WYSIWYG paradigm in which most Word users operate. The program merely creates tex files; one still needs a L<sup>A</sup>T<sub>E</sub>X system to create a dvi or pdf file.

**XML and Epic** There has been much attention paid to the subject of how to write mathematics on the web. This subject, which is extensively described in [4], is one area where both L<sup>A</sup>T<sub>E</sub>X and Word have problems. My opinion is that for short discussions, the ability to see equations on a browser is useful, but for anything longer than a couple of pages the natural tendency is to want to print the document. This being the case, simply using pdf files (which are easily produced using `pdfTEX`) gives far better quality.

Arbortext [1] claims that its Epic E-content Engine is able to translate a wide variety of what they call “legacy” formats (including Word and

L<sup>A</sup>T<sub>E</sub>X) into XML. Also, the Epic Editor creates XML documents from scratch. Epic, the company claims, enables the user to create a single source XML document from which versions for print, Web, and wireless can be made. Since XML will eventually replace HTML, this may be a company to watch. I know of one major corporation which once used L<sup>A</sup>T<sub>E</sub>X for technical documents, but has switched to Epic.

### What Needs to be Done

Do we care if people use alternatives to L<sup>A</sup>T<sub>E</sub>X? Those who use products like LyX or Epic are using products which have tried to move beyond L<sup>A</sup>T<sub>E</sub>X. Any improvement to L<sup>A</sup>T<sub>E</sub>X will probably help these other products too. However, if someone is using Word instead of L<sup>A</sup>T<sub>E</sub>X, then they have something which is deficient in several ways when compared with L<sup>A</sup>T<sub>E</sub>X. Nevertheless, if that's what they choose to do, knowing that they could switch to L<sup>A</sup>T<sub>E</sub>X, then we have to respect that choice. However, I believe that L<sup>A</sup>T<sub>E</sub>X has few adherents when compared with Word because most people have never heard of it, and those that have may have overestimated its complexity. For these people, I think that we should proclaim what we know to be a better product. Unlike the commercial vendors of L<sup>A</sup>T<sub>E</sub>X, whose profitability would improve if L<sup>A</sup>T<sub>E</sub>X were more prominent, the rest of us have nothing to gain financially by encouraging the use of L<sup>A</sup>T<sub>E</sub>X. However, more users might improve the development of L<sup>A</sup>T<sub>E</sub>X, in particular the L<sup>A</sup>T<sub>E</sub>X3 Project.

The T<sub>E</sub>X Users' Group has greatly helped the technical improvement of L<sup>A</sup>T<sub>E</sub>X. Perhaps TUG needs to focus more of its efforts on the promotion of L<sup>A</sup>T<sub>E</sub>X. With TUG's blessing, perhaps the T<sub>E</sub>X Live CD could be bundled with all books about L<sup>A</sup>T<sub>E</sub>X. Indeed, this has already happened with the German edition of *A Guide to L<sup>A</sup>T<sub>E</sub>X* [7]. We could go even further than this – the CD could be bundled with new computer systems. An editor would be required as well – perhaps PFE would be sufficient at the outset. A pdf file (or even a plain ASCII text file) could contain more information such as how (and why) to join TUG, how to obtain WinEdt or something similar, and a bibliography of books about L<sup>A</sup>T<sub>E</sub>X.

In summary, I believe that L<sup>A</sup>T<sub>E</sub>X is superior to Word, especially for documents which contain mathematics. However, for a variety of reasons, Word is many times more popular than L<sup>A</sup>T<sub>E</sub>X. To increase the number of L<sup>A</sup>T<sub>E</sub>X users, we need to make it very easy for people to obtain the L<sup>A</sup>T<sub>E</sub>X software,

possibly by widespread distribution of the T<sub>E</sub>X Live CD.

## References

- [1] Arbortext. Epic, 2001. <http://www.arbortext.com:80/>.
- [2] Kirill A. Chikrii and Anna V. Chikrii, 2001. <http://www.word2tex.com/>.
- [3] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The L<sup>A</sup>T<sub>E</sub>X Companion*. Addison-Wesley, Reading, Massachusetts, 1994.
- [4] Michel Goossens, Sebastian Rahtz, et al. *The L<sup>A</sup>T<sub>E</sub>X Web Companion*. Addison-Wesley, Reading, Massachusetts, 1999.
- [5] Michel Goossens, Sebastian Rahtz, and Frank Mittelbach. *The L<sup>A</sup>T<sub>E</sub>X Graphics Companion*. Addison-Wesley, Reading, Massachusetts, 1997.
- [6] Wilfried Hennings. Converters between L<sup>A</sup>T<sub>E</sub>X and PC Textprocessors, 2001. <http://www.tug.org/utilities/texconv/index.html>.
- [7] Helmut Kopka and Patrick W. Daly. *A Guide to L<sup>A</sup>T<sub>E</sub>X*. Addison-Wesley, Reading, Massachusetts, third edition, 1999.
- [8] Leslie Lamport. *L<sup>A</sup>T<sub>E</sub>X: A Document Preparation System*. Addison-Wesley, Reading, Massachusetts, first edition, 1985.
- [9] Leslie Lamport. *L<sup>A</sup>T<sub>E</sub>X: A Document Preparation System*. Addison-Wesley, Reading, Massachusetts, second edition, 1994.
- [10] Tom Love. Why L<sup>A</sup>T<sub>E</sub>X?, 2000. [http://www-h.eng.cam.ac.uk/help/tp1/textprocessing/latex\\_advocacy.html](http://www-h.eng.cam.ac.uk/help/tp1/textprocessing/latex_advocacy.html).
- [11] Tobias Oetiker et al. The not so short introduction to L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>, November 2000. <http://ctan.tug.org/tex-archive/info/lshort/english/lshort.pdf>.
- [12] Conrad Taylor. What has WYSIWYG Done to Us?, 1996. <http://www.ideography.co.uk/library/seybold/WYS-ante.html>.
- [13] Y & Y. Personal communication, 2001.

# Margin Kerning and Font Expansion with pdfTeX

Hàn Th   Thành

## Introduction

pdfTeX has some micro-typographic extensions that are not so widely used, for the lack of documentation and quite complicated setup. In this paper I would like to describe their use in a step-by-step manner so the reader can give a try afterwards. Two extensions will be introduced: margin kerning and font expansion.

Margin kerning is the technique to move the characters slightly out to the margins of a text block in order to make the margins look straight. Without margin kerning, certain characters when ending up at the margins can cause the optical illusion that the margins look rather ragged. Margin kerning is similar to hanging punctuation, but it can also be applied to other characters as well. When used with appropriate settings, this extension can help to considerably improve appearance of a text block.

Font expansion is the technique to use a slightly wider or narrower variant of a font to make interword spacing more even. A font that can be expanded thus has some "stretchability" and "shrinkability". The potentiality to make a font wider or narrow then can be used by the line-breaking engine to choose better breakpoints.

I will describe the use of those two extensions by examples, as I find it a very good way to explain how to use something in practice. People interested in the concepts and further details will find more in my thesis.

## Margin kerning

The simplest use of margin kerning can look like:

```
\input protcode.tex
\font\f=cmr10
\rpcode\f'\-=700
\adjustprotcode\f
\f
\pdfprotrudechars=1
Some text...
```

`\rpcode` stands for "right protruding code". The first parameter is a font identifier, and the second parameter is a character code. The third parameter specifies the amount how much the character will be moved to the right margin. The above example says that if the hyphen character from font

`\f` ends up at the right margin, it should be moved out to the margin by 700 thousandths of its width (i.e., 70%).

It is convenient to specify the protruding factor for individual characters in thousandths of character width. This is also the way how `\rpcode` was implemented in versions up to 0.14h. However, this method cannot be used for characters with zero width ("faked" characters that can be used to protrude other elements than normal characters), so in version 0.14h and later, the protruding amount is specified in thousandths of an *em* of the font. A macro called `\adjustprotcode` (defined in file `\protrude.tex`) is used here to check whether the used version is older than 0.14h and if so it will convert the settings for versions before 0.14h (i.e., in thousandths of character width) to the corresponding settings for later versions (i.e., in thousandths of an *em*).

By default, all characters have their `\rpcode` set to zero, so no characters will be protruded unless we explicitly set `\rpcode`.

The primitive `\pdfprotrudechars` is used to control margin kerning at global level.

- $\leq 0$ : no margin kerning.
- 1: level 1 margin kerning, which does not have effect to line breaking. This setting is handy if you want to keep line-breaking to be compatible with TeX.
- $\geq 2$ : level 2 margin kerning, which usually gives different result of line breaking. This setting causes the line-breaking engine to take the amount needed to protrude the characters at the margins into account. As the result, interword spacing is better.

Now we go for another example:

```
\input protcode.tex
\font\f=cmr10
\lpcode\f'\='=700
\rpcode\f'\'=700
\adjustprotcode\f
\f
\pdfprotrudechars=2
Some text...
```

This example introduced another primitive, `\lpcode`. It is the counterpart of `\rpcode`, used for protruding at the left margin. The above example

thus sets the left quote to be protruded at left margin and the right quote at the right margin. We set `\pdfprotrudechars` to 2, which may lead to different line breaking.

Fortunately one does not have to set `\lcode` and `\rcode` for each character. A common set of protruding factors works quite reasonably for most of body fonts, so one can use them as in the following example:

```
\input protcode.tex
\font\f=cmr10
\setprotcode\f
\f
\pdfprotrudechars=2
Some text...
```

The macro `\setprotcode` is also defined in the file `protcode.tex` and it will call `\adjustprotcode` after settings the common values for protruding, that is why we do not have `\adjustprotcode` in this example.

Assignment to `\lcode` and `\rcode` is always global. In a L<sup>A</sup>T<sub>E</sub>X document, setting up margin kerning can look like:

```
\documentclass{report}
...
\input protcode.tex
\begin{document}
\setprotcode\font
{\it \setprotcode\font}
{\bf \setprotcode\font}
{\bf\it \setprotcode\font}
Some text...
\end{document}
```

In case the settings in `protcode.tex` do not look for a particular font, you can always change it to your taste.

## Font expansion

Use of font expansion is more complicated due to the need to create expanded versions of a font. This task can be done on-the-fly; however, its setup is system-dependent so I will describe rather how to create these fonts manually.

**Creating expanded fonts** Font expansion can be used with

1. fonts based on Computer Modern fonts (i.e., Computer Modern fonts and variants for Czech, Vietnamese, etc.);
2. Type 1 fonts;
3. Multiple Master fonts with a width axis.

Expanded fonts have the name from the base font, followed by the expansion amount in thou-

sandths. For example, `cmr10` expanded by 10 thousandths (1%) will be named `cmr10+10`. Expansion value can be negative, which stands for condensing.

Putting the created fonts into their correct location is another matter. I suggest simply putting all expanded fonts into a single directory to play with.

**Computer Modern fonts** Computer Modern fonts can be expanded by altering the *unit width* in the source. For example, `cmr12+10.mf` is created by copying the source of the base font (`cmr12.mf`), with a line appended after the place where the unit width is defined as:

```
u#:=20/36pt#;      % unit width
u#:=u#+10/1000u#;
```

Then the changed source is used to generate the expanded TFM as well as the bitmap font needed for rendering the output. While using expanded Computer Modern fonts, the corresponding entries in used the map files must be commented out, otherwise pdfTeX will treat them as Type 1 fonts.

**Type 1 fonts** Type 1 fonts are expanded by altering the *FontMatrix* of Type 1 fonts. pdfTeX will do that for you, the only task is to create the expanded TFM files. Suppose that we have an AFM file `putr8a.afm` (Adobe Utopia Regular). Then creating 8y-encoded TFM expanded by 10 thousandths (1%) can look like:

```
afm2tfm putr8a.afm -e 1.010
-T texnansi.enc putr8y+10.tfm
```

You also need an entry in the map files that looks like:

```
putr8y Utopia-Regular "TeXnANSIEncoding
ReEncodeFont" <texnansi.enc <putr8a.pfb
```

Only the entry for the base font (non-expanded) is needed. The expanded versions will be embedded according to the entry for the base font.

**Multiple Master fonts** Only Multiple Master with a width axis can be expanded. The idea is to create a new instance with the width value increased by the expansion amount. The following example shows how to create an Multiple Master instance and a variant expanded by 20 thousandths:

```
mmafm --weight=400 --optical-size=12 -
-width=535 -
-output pmmr8a12.afm MinionMM.afm
```

```
mmafm --weight=400 --optical-size=12
--width=545.7
--output pmmr8a12+20.afm MinionMM.afm
```

The magic number 545.7 comes from the expression  $535 \times (1 + \frac{20}{1000})$ , which means that we

increase the width value of the base instance by 20 thousandths.

The PFB font files are created in a similar way. Afterwards the AFM can be converted to TFM using `afm2tfm`:

```
afm2tfm pmnr8a12.afm -T
      texnansi.enc pmnr8y12.tfm
```

```
afm2tfm pmnr8a12+20.afm -T
      texnansi.enc pmnr8y12+20.tfm
```

Similar to Type1 fonts, only the entry for the base font is needed in map files:

```
pmnr8y12 MinionMM_400_535_12_
<texnansi.enc <pmnr8a12.pfb
```

**Using font expansion** Suppose that given a font, we know how to create expanded versions of that font. Now let us try an example:

```
\font\font=cmr10
\pdffontexpand \font 20 10 5 1000
\efcode\font'\e=1000
\font
\pdfadjustspacing=1
Some text...
```

The primitive `\pdffontexpand` says that font `\font` can be expanded up to 20 thousandths, condensed to 10 thousandths by step 5 thousandths. This means that only variants whose expansion amount is a multiple of 5 are needed. In our example, the following variants are needed (they must be created before running the example): `cmr10+20`, `cmr10+15`, `cmr10+10`, `cmr10+5`, `cmr10-5`, `cmr10-10`.

The last parameter is so-called font expansion factor, and 1000 is the recommended value for most cases. More details in my thesis.

The primitive `\efcode` (character expansion factor) has syntax similar to `\lpcode` and `\rptide`. The third parameter says how much the character ‘e’ is allowed to be expanded in thousands, in this case fully. This parameter can be used to restrict expansion of certain characters that are more sensitive to expansion. The default value of `\efcode` is zero, thus no expansion is allowed unless we explicitly set `\efcode`.

`\pdfadjustspacing` is also similar to `\pdfprotrudechars`, i.e., is used to control font expansion at global level: 0: no expansion; 1: expansion with backward compatibility (unchanged line breaking); and  $\geq 2$ : expansion that may change line breaking.

A macro called `\resetefcode`, available in the file `efcode.tex`, can be used to reset all expansion factors to 1000.

So we can try to put margin kerning and font expansion together:

```
\input protcode.tex
\input efcode.tex
\font\font=cmr10
\setprotcode\font
\resetefcode\font
\pdffontexpand\font 20 20 5 1000
\pdfadjustspacing=2
\pdfprotrudechars=2
\font
Some text...
```

or in a L<sup>A</sup>T<sub>E</sub>X document:

```
\documentclass{report}
...
\input protcode.tex
\input efcode.tex
\def\setupfont#1{
  \setprotcode#1
  \resetefcode#1
  \pdffontexpand#1 20 20 5 1000
}
\begin{document}
\setupfont\font
{\it \setupfont\font}
{\bf \setupfont\font}
{\bf\it \setupfont\font}
Some text...
\end{document}
```

## Conclusions

Margin kerning is a simple extension but quite effective. Its setup and use is very easy, while the gain is considerable. I would recommend it to regular use.

Font expansion must be used with care, as use with too tolerant expansion limits can be dangerous. According to practical experiments, the limit is  $\pm 2\%$ .

## References

1. My thesis (PDF version): <http://www.fi.muni.cz/~thanh/download/thesis.pdf>
2. Various macros for margin kerning and font expansion with pdf<sub>T</sub>E<sub>X</sub>: <ftp://ftp.cstug.cz/pub/tex/local/cstug/thanh/pdfTeX/ext/>

# PDF presentations using the *Marslide* package

Wendy McKay

Control and Dynamical Systems, CalTech, Pasadena, USA

[wgm@cds.caltech.edu](mailto:wgm@cds.caltech.edu)

<http://www.cds.caltech.edu/wgm/>

Ross Moore

Mathematics Department, Macquarie University, Sydney

[ross@maths.mq.edu.au](mailto:ross@maths.mq.edu.au)

<http://www.maths.mq.edu.au/ross/>

## Abstract

The *Marslide* package is useful for creating high quality PDF presentation slides, especially when mathematics is required. It works equally well with both pdf- $\LaTeX$  and  $\LaTeX$ +dvips+distill methods for generating documents in PDF format [2].

The package does not of itself provide a lot of new functionality; rather it combines the use of existing packages in a consistent way, solving problems of compatibility and loading-order. With such problems solved, the full power of packages such as `hyperref`, `texpower`, `geometry` and `everyshi` can be exploited fully, to produce presentation slides that rival, indeed surpass, what can be achieved with other tools.

Some new sub-packages are included to make it easy to use alternative fonts (in particular Lucida for mathematics, and some Adobe or TrueType fonts), and to place background images on every page. An extended option-loading mechanism allows for arbitrary extension of the package, and customised document styles.

## What's in a name?

The collection of  $\LaTeX$  macros that are used within what we now call the *Marslide* package have been developed over many years, in response to a direct need for projecting mathematics for lectures and seminar talks. Professor Jerrold Marsden, at the California Institute of Technology's Control and Dynamical Systems group, is a prolific writer of advanced mathematics texts. Jerry has used  $\TeX$  exclusively for many years, both for book manuscripts and for lectures and seminars. One of us (Wendy) is his administrative assistant.

The other (Ross) has enjoyed several invitations to visit CalTech, for the purposes of helping to debug/develop  $\TeX$ niques for use in the books, and to share  $\TeX$ pertise in web-site construction and on-screen presentations via the 'Cal $\TeX$  Talks' lecture series [4, 5, 6, 7]. Some of these visits have correlated well with annual TUG meetings in the USA; the WaRMreader macros [9] (the subject of another talk at this meeting) were developed to solve a specific

problem concerning the labelling of graphics for some of Jerry's many books.

The *Marslide* package itself was developed so as to have a unified way to prepare slides for two different situations:

- lectures and talks, involving significant amounts of mathematics and written text in a classroom setting;
- copying a particular style, developed initially using PowerPoint, to be used at a particular meeting where several presenters were to be using the same uniform style.<sup>1</sup>

In this latter case, the need to include a significant amount of properly typeset mathematics meant that a good  $\TeX$ -based solution needed to be developed. This was done, to the great satisfaction of all who viewed the presentation. Thus the 'Mars' in the

---

<sup>1</sup> This style was developed by Peter Schröder, for the Multiresolution Simulation & Engineering Design (**mRSED**) project at CalTech.

name ‘Marslide’ refers to Prof. Marsden, not to the heavenly body.<sup>2</sup>

### ***Marslide*, as a meta-package**

Although loaded as a  $\LaTeX$  package, using a command such as

```
\usepackage[options]{marslide}
```

it is better to think of *Marslide* as being a *meta*-package, for the following reasons.

1. There are not many new commands or environments defined within `marslide.sty` itself. Most of the specialised behaviour is achieved by loading other packages, already available with all  $\LaTeX$  distributions; e.g., `geometry`, `graphicx`, `hyperref`, `color`, `verbatim`, `fontenc`, `multicol` and also `amssymb`, `amsmath`, `tabularx`.
2. Extra packages `texpower`, `everyshi` and `eso-pic` are used for specialised effects. These are available at CTAN or elsewhere on the Internet.
3. To adjust font-sizes and page-layout to be suitable for a screen presentation, *Marslide* overrides such aspects of the document-class chosen with the `\documentclass` command. (The reason why *Marslide* was not implemented as a document-class is discussed below.)
4. Extra subsidiary packages `hugefonts`, `bgimages` and `lucrotis` have been written primarily for use with *Marslide*. (These packages are sufficiently self-contained to be usable also in other contexts.)
5. *Marslide* can be customised to include extra features. Indeed it *must* be customised. A subsidiary file `marsdefs.sty` is loaded as the default customisation when no alternative option has been specified.
6. Customisation is achieved using a flexible extension to  $\LaTeX$ 's optional argument mechanism. For example,

```
\usepackage[...mydefs]{marslide}
```

will cause a file `mars-mydefs.sty` to be read at the appropriate time during processing of the document preamble, provided such a file is in the current working directory, or can be found on the usual search paths. (Any string instead of `mydefs` can be used, except for strings that correspond to valid options to *marslide* or the packages listed in items 1, 2 and 4 above or to the `lucidabr` package.)

<sup>2</sup> However, the association is not entirely misplaced, as Jerry's work in many-body problems includes techniques for the calculation of orbits of planets and spacecraft. This includes simulations for actual space-missions to the red planet.

The reason for this structure is because *Marslide* needs to load packages such as `geometry`, `hyperref`, etc. with some specific optional parameters. However, some of these packages can conflict with one another, in the sense that reversing the loading order can lead to unexpected results from the  $\LaTeX$  processing.

Authors may wish to use other options as well with these packages. This can be done using  $\LaTeX$ 's `\PassOptionsToPackage` command, *before* the command to load *marslide*.

Similarly, authors may wish to load other packages as well as those provided automatically as part of the *Marslide* setup. Beware that the use of extra packages increases the possibility of encountering situations where unexpected effects may occur due to incompatibilities, or a dependency on the order in which the packages need to be loaded. To be able to deal with such dependencies it is important that packages may be loaded either before, or after `marslide.sty` is read. Since it is an error in  $\LaTeX$  to use `\usepackage` *before* the `\documentclass` command, it was necessary to *not* write `marslide.cls`, but use `marslide.sty` as a (*meta*-)package.

Note that it is possible that some packages may not give the correct results when loaded *either before or after marslide*, due to conflicts with other packages. Instead, such packages may need to be loaded *as part of Marslide*. This is a secondary purpose of the customisation mechanism, outlined in item 6 above, which causes the customisation file to be read after most of the standard packages in items 1, 2 and 4, but before the `geometry` package has been read.  $\LaTeX$ 's `\RequirePackage` command can be used within the customisation file, to load such packages.

### **Not just for PDF**

In setting-up the package-loading structure (as described above) for *Marslide*, much care was taken to ensure that features worked correctly whatever processor was used. Thus the appearance of the PDF document should be the same, whether it is generated directly using pdf- $\LaTeX$  (i.e., pdf- $\TeX$  [8], with the `pdflatex.fmt` format) or using  $\LaTeX$ , followed by a distillation to PDF of the resulting PostScript job.

Indeed the `.dvi` output from  $\LaTeX$  can be used directly for the presentation, either by (color) printing to transparencies, or directly onscreen with a dvi viewer. Even the sequential page-building features of the `texpower` package are available in the dvi version, and some viewers support the hyperlinking

features produced using `hyperref`. This latter requires that the correct driver files are loaded for the particular  $\text{T}\text{E}\text{X}$  implementation. The WaRM team<sup>3</sup> have successfully used *Marslide* with  $\text{t}\text{E}\text{X}$  under Unix, and *Textures*<sup>4</sup> for the Macintosh, as well as with  $\text{p}\text{d}\text{f}\text{-T}\text{E}\text{X}$ .

### Customisation

As noted above, a customisation file is *required* to use *Marslide*. The main purposes of this customisation are to:

1. establish the size and orientation of the paper, and the area in which typesetting is to occur;
2. select the font-faces and sizes to be used for the typesetting;
3. choose colours for textual and graphic elements in the resulting slides;
4. declare graphics for logos, and define other special features to appear on all pages, or just some of them.

It is also usual to define macros for establishing running headers and/or footers within the customisation file, and to define commands making it easy to layout a distinctive title-page for the presentation. These things could be defined within the document preamble, but it is neater to hide them away within the customisation file, so that the document source is not cluttered with hard-to-read macro definitions.

The default customisation file provides a good starting point for defining your own. Firstly you should copy `marsdefs.sty` and rename this copy before making any changes inside. Since it contains a `\ProvidesPackage` command, using `\filedate` and `\fileversion`, please adjust these within the copy.

The default customisation begins by loading the `geometry` package, as follows:

```
\usepackage[landscape, letterpaper, verbose,
%,textheight=5.5 truein %calculated by geometry
%,textwidth=10.0 truein %calculated by geometry
,voffset=-.35pt
,hoffset=0pt
,tmargin=0pt
,bmargin=0pt
,lmargin=36pt
,rmargin=36pt
,headheight=78pt
,headsep=20pt
,footskip=0.20 truein %non-mRSED
,tmargin=.10truein
]{geometry}
\addtolength{\voffset}{-.10truein}
```

<sup>3</sup> Wendy and Ross M(oore or McKay, take your pick!).

<sup>4</sup> produced by Blue Sky Research; see the website <http://www.bluesky.com/>.

The sizes given here work very nicely for US-letter sized paper, oriented as landscape. Consult documentation on the `geometry` for the use of any of these parameters, if it's not clear from the names. In practice, small adjustments to margins or offsets may be needed to remove a single row or column or white pixels at the edge of the paper, when the slide presentation is viewed in a PDF browser. (It is better to do this kind of edit within a customisation file rather than within a larger package.)

The default `marsdefs.sty` also allows for using Lucida fonts, as an alternative to  $\text{T}\text{E}\text{X}$ 's standard Computer Modern fonts. Of course, you'll need to have purchased a set of Lucida fonts, and have correctly installed them for your  $\text{T}\text{E}\text{X}$  application, to be able to use this option. Using the option `lucida`, when loading the `marslide`, causes a file `lucrotis.sty` to be read. This in turn loads the `lucidabr` package, along with suitable options, some of which may have been inherited from options given to `marslide`. Options, such as `callig`, `handw`, `sslucida`, `seriftt`, can be used to determine which of the Lucida fonts is used for the main text font, and the face used with `\texttt`. Other options, such as `T1`, `OT1`, `7bit`, `8bit` determine which font encoding to use, by loading the `fontenc` package with an appropriate option.

The `lucrotis` also allows for the use of Adobe's Rotis font, which has an unusual, but quite pleasing, appearance in its semi-serif form. To use this, you'll need to have `.pfb` (or other PostScript) files for the fonts, and have installed appropriate metrics, virtual fonts and `.fd` files. Metrics and  $\text{T}\text{E}\text{X}/\text{L}\text{A}\text{T}\text{E}\text{X}$ -specific files can be distributed with *Marslide*, but you'll need to purchase the fonts themselves from Adobe, if you wish to use these fonts. Similarly the interface files can be constructed (e.g., using the `fontinst` package) for use with *Marslide*, but that's a whole other story.

### Features & Documentation

Most of the special features of *Marslide*, some using the default `marsdefs` customisation file, are displayed in the PDF documentation that accompanies the package, at its distribution site<sup>5</sup>. Figures 1–5 reproduce this documentation, as slides created using the package itself, then captured as 4-up using the `pdfpages` package.

These slides show how to make use of features such as:

- heading levels: different font-sizes, bullets, colorings and indents;

<sup>5</sup> <http://www.cds.caltech.edu/~wgm/WARM/slides/marslide/>



Figure 1: Marsden Slide Package Documentation; pages 1–4, showing different heading levels with associated bullets, coloring and font-sizes. Also some mathematics is shown.



Figure 2: Marsden Slide Package Documentation; pages 5–8, showing examples of code-listings, color-specifications and some predefined color-commands for partial graphic elements and styles of text.

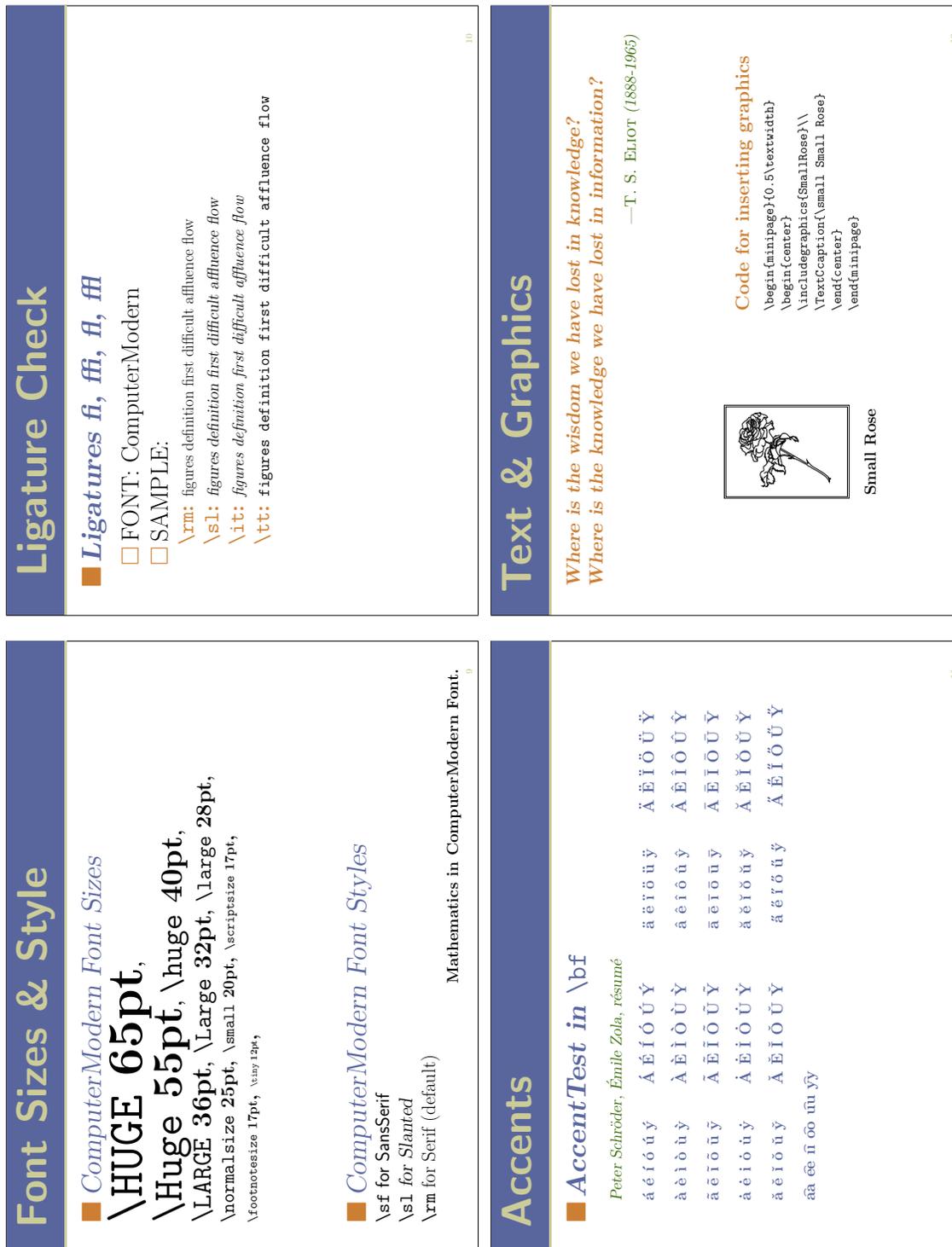


Figure 3: Marsden Slide Package Documentation; pages 9–12, showing the sizes associated with L<sup>A</sup>T<sub>E</sub>X’s size commands, and examples testing that ligatures and accents are correctly formed.



**Figure 4:** Marsden Slide Package Documentation; pages 13–16, showing how to include graphics and colored banners, as well as examples of inserting dingbats either explicitly or implicitly.

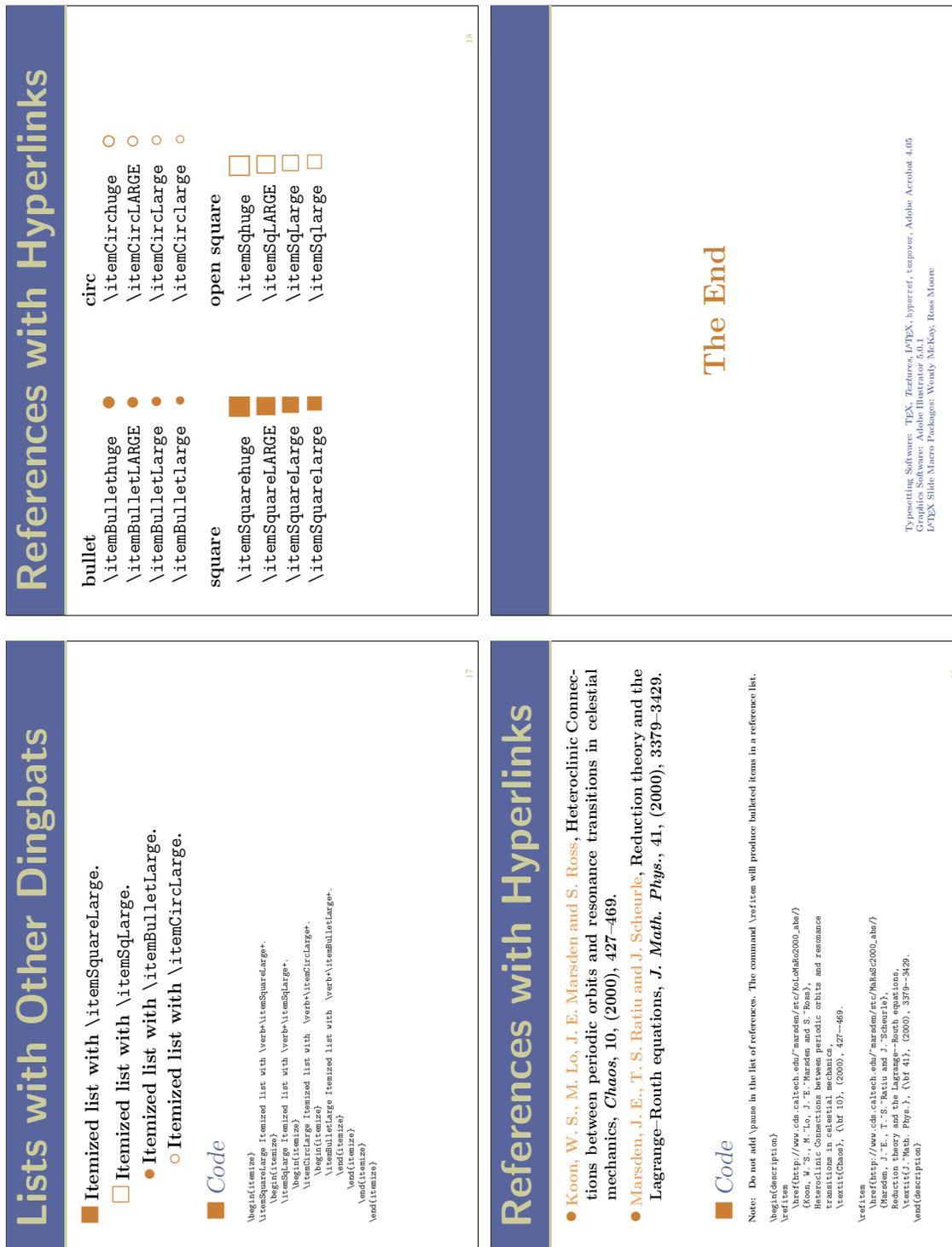
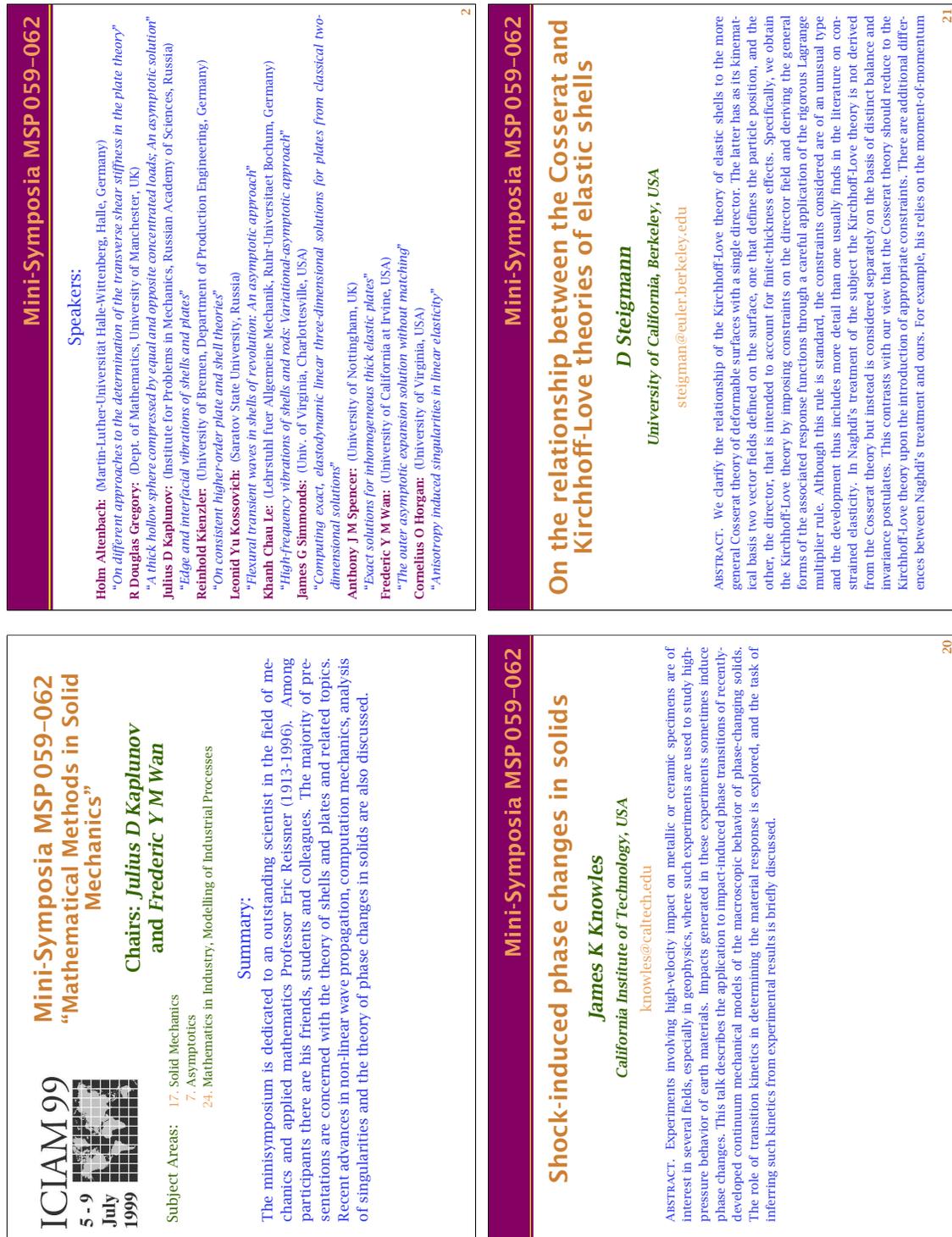


Figure 5: Marsden Slide Package Documentation; pages 17–20, showing the full range of available dingbats at different sizes, and how to use hyperlinks in the bibliography.



**Figure 6:** Introductory slides for a congress mini-symposium, with speaker listing and abstracts of all the talks.

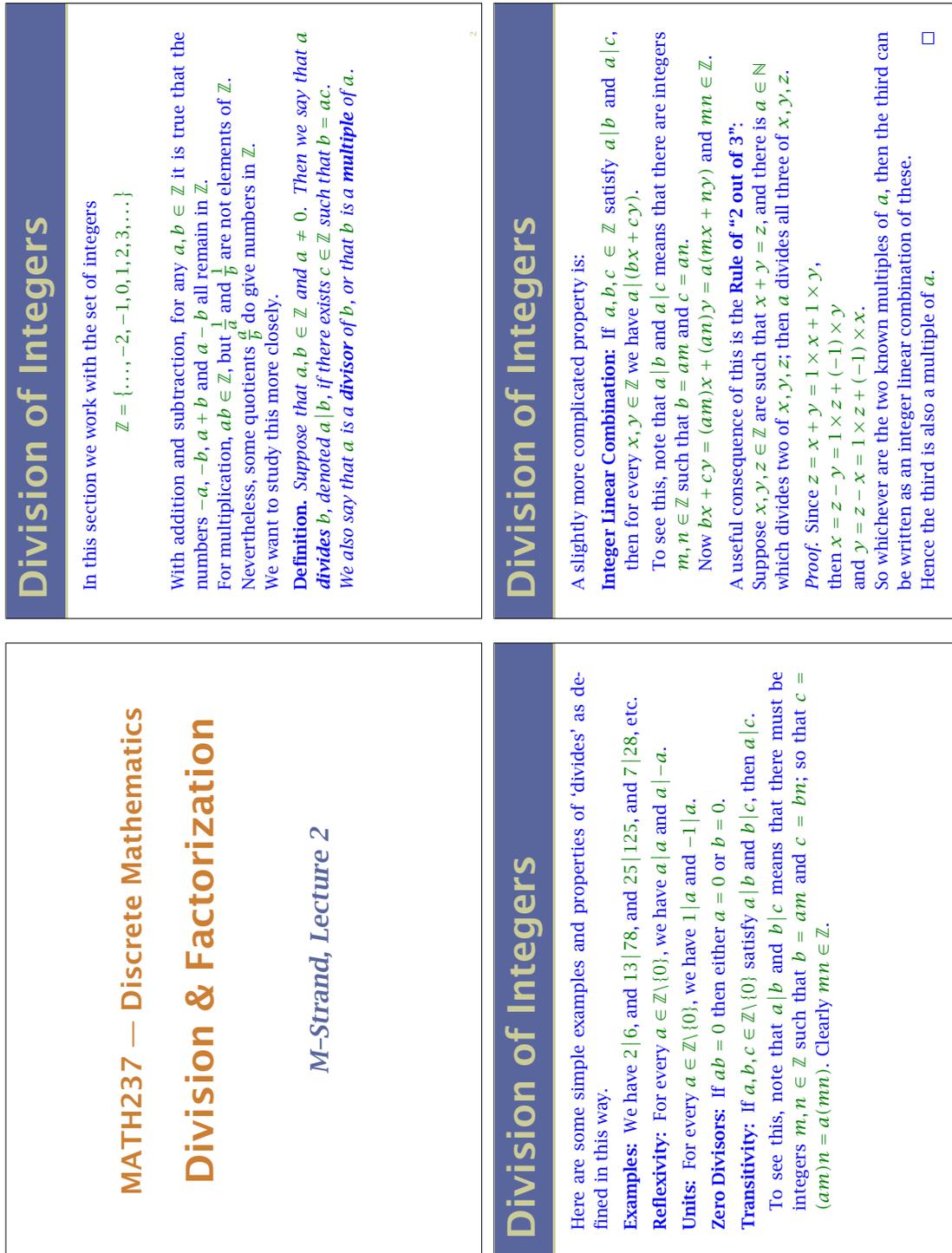


Figure 7: Lecture slides for use in mathematics teaching.

- colors: RGB-codes, color-names and color commands;
- font-sizes, ligatures and accents;
- colored banners: in the header, or part of the body-text;
- dingbats: different shapes and sizes, open or solid;
- graphics: imported using `\includegraphics`;
- hyperlinks: for cross-references, citations, etc.

There are some newer features which are not yet documented there. These include:

- using a background image on every page, or several images on select pages;
- removing the banner in the header, for a larger text-body;
- generation of PDF bookmarks to document sections, or pages where the banner text changes;
- macros to insert hyperlinks to external movie files, with a poster-picture as the hyperlink button to start the movie;
- set a margin to avoid the edges of the paper that cannot be printed, but which does not show when the PDF slides are viewed in a browser.

### More examples

Figure 6 shows a document style that could be used effectively at a large conference or congress. This document presents the speakers and abstracts for just one lecture session or mini-symposium on a specialised topic. Typically there'll be many of these, distributed together on CD-ROM, with speaker and subject indexes for the whole congress.

Figure 7 shows lecture slides that one author has used when teaching a mathematics course. Such documents are available to students from a web-site, so that they can review the lecture presentation. As there are typically many incremental page-builds, these documents are not suitable for printing as lecture notes. Instead, a 4-up version is provided of the fully-built pages, as are presented here in this proceedings volume.

Elsewhere in this volume are other papers by the same author(s) [9, 10]. These include further examples of the use of *Marslide*.

### References

- [1] Adobe Systems Inc.; Acrobat Reader, viewer for PDF format [2] documents, available free of charge from <http://www.adobe.com/>.
- [2] Adobe Systems Inc.; “Portable Document Format, Reference Manual, Version 1.3”; March 11, 1999.
- [3] Adobe Systems Inc.; “pdfmark Reference Manual”; Technical Note #5150; Adobe Developer Relations; Revised: March 4, 1999.
- [4] CalTEXTalk #1, June 1998: “Automatic Generation of Web-sites using L<sup>A</sup>T<sub>E</sub>X2HTML”; on-line abstract: <http://www.cds.caltech.edu/caltex/1998/>.
- [5] CalTEXTalk #2, July 1999: “T<sub>E</sub>X is Alive and Well for Web-based Presentation of Mathematics”; abstract: <http://www.cds.caltech.edu/caltex/1999/>.
- [6] CalTEXTalk #3, May 2000: “High Quality Electronic Presentation of Mathematics; Using PDF, T<sub>E</sub>X/L<sup>A</sup>T<sub>E</sub>X, and More”; abstract: <http://www.cds.caltech.edu/caltex/2000/>.
- [7] CalTEXTalk #4, August 2001: “Advanced uses of pdfL<sup>A</sup>T<sub>E</sub>X”; abstract: <http://www.cds.caltech.edu/caltex/2001/>.
- [8] Hàn, Thé Thành; pdfT<sub>E</sub>X, free software for generating documents in PDF format, based on the T<sub>E</sub>X typesetting system. Available for all computing platforms; see <http://www.tug.org/applications/pdftex/>.
- [9] McKay, Wendy and Moore, Ross and Ruark, Tom; “Adobe plugin for WARMreader” T<sub>E</sub>X Users Group 2001 Proceedings, (elsewhere in this volume).
- [10] Moore, Ross and Griffin, Frances; “MacQ<sub>T</sub>E<sub>X</sub>: Online self-marking Quizzes, using pdfT<sub>E</sub>X and exerquiz” T<sub>E</sub>X Users Group 2001 Proceedings, (elsewhere in this volume).

# Using T<sub>E</sub>X to Enhance Your Presentations

Hans Hagen

Pragma ADE, GH Hasselt, The Netherlands

[pragma@wxs.nl](mailto:pragma@wxs.nl)

<http://www.pragma-ade.com>

## Abstract

The quality of a presentation can be improved by using a decent visualization of the content. For this purpose ConT<sub>E</sub>Xt comes with a bunch (currently there are some 25) of presentation styles. In this talk I will demonstrate how you can use T<sub>E</sub>X to structure your presentation in such a way that it suits the topic and way you think. I will show that the combination of structure and high end typesetting will give your presentations the finishing touch they deserve. Although making presentations can be done with any T<sub>E</sub>X, I will discuss the power of coupled ConT<sub>E</sub>Xt and pdfT<sub>E</sub>X.

# Techniques of Introducing Document-level JavaScript into a PDF file from a $\LaTeX$ Source

D. P. Story

Department of Mathematics and Computer Science, University of Akron, Akron, OH 44325

[dpstory@uakron.edu](mailto:dpstory@uakron.edu)

<http://www.math.uakron.edu/~dpstory/>

## Abstract

The method of introducing Acrobat document-level JavaScript (DLJS) into a PDF depends on the application used: `pdftex` or `dvipdfm`. Until recently, users of Acrobat's distiller did not have the ability to automatically introduce such JavaScript into the document from a  $\LaTeX$  source. For users of Acrobat 5.0, this situation is, at last, rectified. The focus of this paper is to enumerate, illustrate and discuss these various methods.

A new package, `insDLJS`, is also introduced. This package enables both package and document authors to insert document-level JavaScript.

## Introduction

Adobe's Portable Document Format (PDF) has become widely used by some as a document exchange format;  $\LaTeX$  users routinely convert their research papers and technical documents to PDF and post them on the Internet, or e-mail them to a interested colleague. When the  $\LaTeX$  package `hyperref`, written by Sebastian Rathz and Heiko Oberdiek, is used, cross-references created by standard  $\LaTeX$  commands are automatically converted into hyperlinks within the PDF document.

However,  $\LaTeX$  and PDF are capable of more than just producing an electronic copy of a paper document. By utilizing  $\LaTeX$ 's package customization feature ( $\LaTeX$  packages) and Acrobat's form elements and powerful JavaScript language, an author can build a colorful, richly interactive PDF. Such a document can be used as a teaching tool that would quicken the interest of the student.

The focus of this paper is on JavaScript. JavaScript can be attached to a PDF document in many ways, for example, to a form button. JavaScript attached to buttons can be used to initiate actions. Any scripts that are repeatedly used, are general (not form specific), or are rather lengthy, can be placed at what Acrobat calls the *document-level*; they are called document-level JavaScripts (DLJS). A button action, then, can simply call these DLJS to perform various calculations or tasks.

For authors or package writers who really want to tap into the power of JavaScript, document-level scripts need to be written that perform the target task and embedded into the PDF. In the context

of developing an educational document, many revisions to the document are typically made and the JavaScript must be reinserted each time the PDF is built. Ideally, the JavaScripts should appear within the  $\LaTeX$  source file, which makes it easy to access and modify them, and then *automatically* inserted when the PDF is built.

The applications `pdftex` and `dvipdfm` both support the insertion of DLJS from a  $\LaTeX$  source. Historically, it was not possible to insert DLJS from a  $\LaTeX$  source through the distiller method, and sadly, *this is still the case*. However, the Acrobat application, version 5.0, does include some extensions that can be exploited to obtain an automatic insertion of DLJS. For version 5.0, it is the Acrobat application that inserts the DLJS automatically following distillation, *not* the distiller itself.

The discussion that follows—for the case of production of PDF by the distiller method—assumes that the author has the full Acrobat 5.0 product. It is important to note that it is only necessary for the *author* to have full Acrobat 5.0; the user, the one who reads the document, needs only have the Acrobat Reader (version 4.0 or later).

## Inserting DLJS

We begin by discussing the basic techniques for each of the common applications used for producing a PDF document. Suppose we wish to introduce the following JavaScript function at the document level.

```
function HelloWorld()  
{  
    app.alert("Hello World!", 3);  
}
```

A variation on a rather well-known, yet useless function.

In the sections that follow, it is assumed the source document uses the `hyperref` package, with the appropriate driver setting: `dvips`, `dvipsone`, `pdftex` and `dvipdfm`. Some of the commands in the code below have their definitions in the `hyperref` package.

Within a package, or preamble, make the following definitions. This code is common to all cases, `pdftex`, `dvipdfm` and `distiller`.

```
\def\jsR{\string\r} % carriage return in JS
\def\jsT{\string\t} % tab in JS
\begingroup\obeyspaces\obeylines%
\global\let^^M=\jsR%
\gdef\DLJS{%
function HelloWorld()
{
    app.alert("Hello World!", 3);
}
}%
\endgroup
```

This defines a macro that expands to the code of the desired JavaScript function. Note that `\obeyspaces` and `\obeylines` are used; `\global\let^^M=\jsR` will insert a `\r` at the end of each line of the JavaScript code, this nicely formats the script within the PDF file itself.

**pdftex and dvipdfm** The approach to DLJS insertion is similar in the case of these two applications, both of which fully support, through primitives or specials, DLJS insertion.

In each case, insertion is a two step process: (1) create an PDF object dictionary containing the necessary key-value pairs

```
<< /S /JavaScript /JS (\DLJS) >>
```

and (2) enter the script name and reference offset into the `Names` array of the `JavaScript` dictionary.

The implementation of these two steps is different for these two applications.

We wrap the code in the next subsections in a `\AtBeginDocument` command. For private macros, this is not necessary, but for public macro packages it is useful to delay the insertion of the code until after the preamble.

**pdftex** For `pdftex`, the following is all the additional code needed to get DLJS insertion:

```
\AtBeginDocument
{%
    \immediate\pdfobj
    { << /S /JavaScript /JS (\DLJS) >> }
    \xdef\objDLJS{\the\pdflastobj\space 0 R}
    \immediate\pdfobj {%
        << /Names [(Doc Level JS) \objDLJS] >> }
```

```
\xdef\objNames{\the\pdflastobj\space 0 R}
\pdfnames {/JavaScript \objNames}
}
```

The control sequences `\pdfobj` and `\pdfnames` are `pdftex` primitives.

**dvipdfm** Similarly, for `dvipdfm`, we need only include the following code to get the DLJS:

```
\AtBeginDocument{%
    \immediate\@pdfm@mark{obj @objDLJS
        << /S /JavaScript /JS (\DLJS) >> }
    \immediate\@pdfm@mark{obj @objnames
        << /Names [(Doc Level JS) @objDLJS] >> }
    \@pdfm@mark{put @names
        << /JavaScript @objnames >> }
}
```

The command `\@pdfm@mark` is defined in terms of a `dvipdfm` `\special`,

```
\def\@pdfm@mark#1{\special{pdf: #1}}
```

as defined in `hyperref`.

**Multiple Functions** More than one function can be grouped together using the obvious approach:

```
\begingroup\obeyspaces\obeylines%
\global\let^^M=\jsR%
\gdef\DLJS{%
function HelloWorld()
{
    app.alert("Hello World!", 3);
}
function GetNumberOfPages()
{
    app.alert("The number of pages is "
        + this.numPages);
}
}%
\endgroup
```

Now, we proceed as described above.

An alternate approach is to have separate definitions for each of you functions, say, `\DLJSi` and `\DLJSii`, and insert each of them separately into the JavaScript dictionary. For example, in the case of `pdftex`, we could include the following code:

```
\AtBeginDocument
{%
    \immediate\pdfobj
    { << /S /JavaScript /JS (\DLJSi) >> }
    \xdef\objDLJSi{\the\pdflastobj\space 0 R}
    \immediate\pdfobj
    { << /S /JavaScript /JS (\DLJSii) >> }
    \xdef\objDLJSii{\the\pdflastobj\space 0 R}
    \immediate\pdfobj {%
        << /Names
            [
                (Doc Level JS) \objDLJSi\space
                (More DLJS) \objDLJSii
            ]
        >> }
```

```
>> }
\edef\objNames{\the\pdfobj\space 0 R}
\pdfnames {/JavaScript \objNames}
}
```

Similarly for dvipdfm.

Acrobat expects the script names of the DLJS (these are ‘Doc Level JS’ and ‘More DLJS’, in the example above) to be *sorted*; otherwise, the Acrobat application does not give editing access to the scripts that are ‘out-of-sorts’.<sup>1</sup> The scripts would be accessible from within Acrobat or Reader, but would not necessarily be available for editing within Acrobat.

**Acrobat 5.0** For authors that use Acrobat 5.0 (those that use `dvips` or `dvipson` to produce a PostScript file and then distill) the problem is a little more complicated. Version 5.0 comes with an extended FDF (Forms Data Format) specification. This new specification allows DLJS to be placed within the FDF file. The FDF file is then imported into the document and the DLJS is inserted.

**The Concepts** Take our basic `HelloWorld` JavaScript function, and place it into the FDF file, which I’ll call `dljs.fdf`, as follows:

```
%FDF-1.2
1 0 obj
<<
  /FDF
  <<
    /JavaScript
    <<
      /Doc 2 0 R
    >>
  >>
endobj
2 0 obj
[ (Doc Level JS) 3 0 R ]
endobj
3 0 obj
<<>>
stream
function HelloWorld()
{
  app.alert("Hello World!",3);
}
endstream
endobj
trailer << /Root 1 0 R >>
%%EOF
```

Once this file has been created, it can be inserted into the PDF file by including the following code in your L<sup>A</sup>T<sub>E</sub>X document:

<sup>1</sup> Neither `pdftex` or `dvipdfm` sorts the `Names` array by script names.

```
\AtBeginDocument{\literalps@out{%
[ {Page1} << /AA << /O << /S /JavaScript /JS
(%
  if(typeof HelloWorld == "undefined")\jsR\jsT
  this.importAnFDF("dljs.fdf");
)
>> >> >>
/PUT pdfmark}}
```

This code creates an open page action for page 1 of the PDF. When the document is first opened in the Acrobat application, usually immediately following distillation, if the function `HelloWorld` is not already defined, the JavaScript method `importAnFDF` will import the specified FDF into the document; otherwise, the open action does nothing. The JavaScript function `HelloWorld` will be placed at the document-level because of the structure of the FDF file.

**A Simple Implementation** The FDF can be created and edited as a separate file; however, our goal was to have all code contained in the L<sup>A</sup>T<sub>E</sub>X source file itself. A simple solution would be to use a verbatim write to write the necessary code to a FDF file.

For example, consider the following code. It is assumed that the `verbatim` package has been loaded.

```
\makeatletter
\newwrite \dljs@FDF

% open a stream
\immediate\openout\dljs@FDF=dljs.fdf
\let\verbatim@out=\dljs@FDF

% verbatimwrite Environment: Writes to current
%\verbatim@out. Based on examples and macros
% of the verbatim package. Set \verbatim@out
% before calling this macro
\newenvironment{verbatimwrite}
{\@bsphack
 \let\do\@makeother\dospecials
 \catcode'\^^M\active
 \def\verbatim@processline{%
 \immediate\write\verbatim@out
 {\the\verbatim@line}}%
 \verbatim@start}%
{\immediate\closeout\verbatim@out\@esphack}

Now write the FDF file from the LATEX source.
\begin{verbatimwrite}
%FDF-1.2
1 0 obj
<<
  /FDF
  <<
    /JavaScript
    <<
```

```

        /Doc 2 0 R
    >>
>>
endobj
2 0 obj
[ (Doc Level JS) 3 0 R ]
endobj
3 0 obj
<<
>>
stream
function HelloWorld()
{
    app.alert("Hello World!", 3);
}
endstream
endobj
trailer
<<
    /Root 1 0 R
>>
%EOF
\end{verbatimwrite}

```

Use `pdfmark` to import the FDF when Acrobat is first opened.

```

\AtBeginDocument{\literalps@out{%
[ {Page1} << /AA << /O << /S /JavaScript /JS
(%
if(typeof HelloWorld == "undefined")\jsR\jsT
this.importAnFDF("dljs.fdf");
)
>> >> >>
/PUT pdfmark}}
\makeatother

```

Once the DLJS has been inserted, save the file (using “Save As”). The DLJS now is saved with the file. When the document is opened in Acrobat (or Reader) the JavaScript is available to be executed. You can delete the open action at this point.

**Multiple Functions** Multiple functions can either be written to the same FDF file, or to separate FDF files. When you import multiple FDF files into Acrobat using `importAnFDF`, their script names are automatically sorted, see the footnote at the end of ‘[Multiple Functions](#)’, page 162.

**Plain TeX Users** The above examples illustrate the basics of DLJS inclusion. Plain TeX users can (easily) adapt these techniques to plain TeX. This is left as an exercise.

### The insDLJS Package

All the code described earlier works very well, and is adequate for someone trying to develop materials for the WWW or for a CD-ROM using a private

macro package. Such a person typically uses only one system—`pdftex`, `dvipdfm`, or the `distiller`—to create the materials. These macros cannot be used conveniently as part of a package, where the user—the one using the package—may want to modify the script and/or define a custom script. A general insert DLJS package is needed. In this section, we discuss a new package called `insDLJS`.

The package `insDLJS` takes the basic techniques already discussed, modifies them so that the insertion will be friendly to package authors who use `insDLJS` to write L<sup>A</sup>T<sub>E</sub>X packages, and to document authors who want to jazz up their documents with DLJS as well.

The package has four driver options: `dvipson`, `dvips`, `pdftex` and `dvipdfm`. The `insDLJS` also defines a new environment called `insDLJS`; the syntax is

```

\begin{insDLJS}[<func>]{<base>}{<name>}
<JavaScript functions or exposed code>
...
...
\end{insDLJS}

```

where,

- #1: This optional parameter, `<func>`, is *required* for the `dvipson` and `dvips` options; otherwise it is ignored. Its value must be the name of one of the functions defined in the environment.
- #2: This parameter, `<base>`, is an alphabetic word with no spaces. It is used to build the names of auxiliary files and to build the names of macros used by the environment.
- #3: The third parameter, `<name>`, is the script name of your JavaScript. This name appears in the “JavaScript Functions” dialog of Acrobat, under the menu

```
Tools > JavaScript > Document JavaScripts..
```

This environment writes one file (`<base>.def`), or possibly two files (`<base>.def` and `<base>.fdf`).

In the case of `pdftex` or `dvipdfm` options, the `<base>.def`, which contains the necessary definitions and code, as discussed in ‘[pdftex](#)’, page 162 and the following section on ‘[dvipdfm](#)’, is read back into the output document to set the DLJS code.

In the `distiller` case, `<base>.def` is read in and second file, `<base>.fdf`, is written. It is the file `<base>.fdf` that is imported into Acrobat following distillation, as outlined on page 163.

The DLJS defined within the `insDLJS` environment in a package and in a preamble of a document will be included in the PDF document automatically. It is important to choose a base name, `<base>`, and a script name, `<name>`, *different* from any that has

already been declared earlier; otherwise, code will be overwritten or simply not appear.

**Example 1.** The following example assumes the driver is either `dvipsone` or `dvips`; in these cases the first parameter is required.

```
\begin{insDLJS}[HelloWorld]{mypkg}{Pkg DLJS}
function HelloWorld()
{
  app.alert("Hello World!", 3);
}
\end{insDLJS}
```

This would result in the automatic insertion of the specified JavaScript at the document-level when the PDF is built.

In this case, the `importAnFDF` method (JavaScript) is used, as discussed on page 163. The value of the optional argument, ‘HelloWorld’, is used to detect whether the script has already been imported. The environment generates the following code:

```
\AtBeginDocument{\literalps@out{%
[ {Page1} << /AA << /O << /S /JavaScript /JS
(%
  if (typeof HelloWorld == "undefined")\jsR\jsT
  this.importAnFDF("mypkg.fdf");
)
>> >> >>
/PUT pdfmark}}
```

The optional argument is ignored and need not even be included when `pdftex` or `dvipdfm` is specified as a package option. □

**Example 2.** A L<sup>A</sup>T<sub>E</sub>X package writer may want to use the `insDLJS` environment to insert DLJS into the package; also, a document author who uses such a package may want to define *additional* DLJS. In this case, just place the code in the preamble, for example,

```
\begin{insDLJS}{dljs}{Private DLJS}
function tugWelcome()
{
  app.alert("Welcome to TUG 2001!", 3);
}
\end{insDLJS}
```

The script will be automatically introduced into the PDF document. □

The `Exerquiz` Package<sup>2</sup> has been rewritten to use the `insDLJS` package. This not only makes the JavaScript code easier to read, modify and maintain, it also allows users of `Exerquiz` to add in their own DLJS. `Exerquiz` uses the base name of `exerquiz`.

**Example 3.** A package author who uses the package `insDLJS` to insert DLJS may wish the package user to modify the scripts in some authorized way. For example, the package author may include

<sup>2</sup> CTAN:macros/latex/contrib/supported/webeq

```
\newcommand\Hello{"Hello World!"}
\begin{insDLJS}
function HelloWorld()
{
  app.alert(@Hello, 3);
}
\end{insDLJS}
```

The script included in the output file (`.dvi` or `.pdf`) using the `\AtBeginDocument` mechanism. The user then has the opportunity to redefine `\Hello` to, say, `\renewcommand\Hello{"Bonjour le Monde\space !"}`

Now the `HelloWorld` function will create an alert dialog that says, “Bonjour le Monde !”. □

**Important.** Note that ‘@’ is used as the escape character within the environment. The `insDLJS` environment eventually leads to a `verbatim` environment. The at-char, ‘@’, has its catcode changed, `\catcode'\@=0`; this will allow macros to expand within the `verbatim` environment. The backslash ‘\’ is needed in JavaScript since it is used as an escape. JavaScript does *not* use the ‘@’ symbol at all, so we are free to use it here.

**A Complete Example** The following listing is a complete example of usage of the `insDLJS` package.

```
\documentclass{article}
\usepackage[pdftex]{hyperref}
\usepackage[pdftex]{insdljs}
\newcommand\tugHello{Welcome to TUG 2001!}
\begin{insDLJS}{mydljs}{Private DLJS}
function HelloWorld()
{
  app.alert("@tugHello", 3);
}
\end{insDLJS}
\begin{document}
\begin{Form} % needed for \PushButton
\section[Test of the \texttt{insDLJS} Package]
% use built-in form button from hyperref
Push \PushButton[name=myButton,
  onclick={HelloWorld();}]{Button}
\end{Form}
\end{document}
```

**A Double-Edged Problem** In the process of extending the `exerquiz` package to include the ability to evaluate objective-style questions using DLJS, one big problem was encountered; consider the following example:

```
\begin{insDLJS}[<func>]{<base>}{<name>}
function TestForLParen(string)
{
  if (/\/(.test(string))
    app.alert("Found Left Parenthesis!",3);
}
\end{insDLJS}
```

**The Problem.** The above script will search the parameter `string` for a left parenthesis, ‘(’. When `dvipson` or `dvips` is used (i.e., the distiller is used), the `<base>.fdf` file is created, imported into the PDF file and the script works correctly. When `pdftex` or `dvipdfm` is used this code does not work! When the file is opened in Acrobat an exception is thrown, and an error message appears:

```
SyntaxError: unterminated parenthetical (
```

In the case of `pdftex` and `dvipdfm`, the JavaScript code is written directly to the PDF file, bypassing distiller. The problem with the above code turns out to be *unbalanced parentheses*. (Acrobat treats ‘(’ as a left parenthesis, not an escaped special character.) All special characters, therefore, need to be escaped; we want

```
\begin{insDLJS}[<func>]{<base>}{<name>}
function TestForLParen(string)
{
  if (/\\(/.test(string))
    app.alert("Found Left Parenthesis!",3);
}
\end{insDLJS}
```

This code now works when the option `pdftex` or `dvipdfm` is used; however, when this code is used with the distiller (with the `dvipson` or `dvips` option), *Acrobat 5.0 crashes!*

**The Solution.** We have two opposing problems, solving one creates another. The solution is to introduce a command, `\e`. (The ‘e’ is for escape.) The macro has two definitions: for the `dvipson` and `dvips` options, we define `\gdef\e{}`; for `pdftex` and `dvipdfm` options, we make the following definitions:

```
\catcode'\@=0 @catcode'\@=12 @gdef\e{\
```

Note that in all cases, we do change the catcode of ‘@’ to `\catcode'\@=0`.

Thus, to get the function `TestForLParen` to be properly defined for all options, we must type:

```
\begin{insDLJS}[<func>]{<base>}{<name>}
function TestForLParen(string)
{
  if (/@e\@e(/.test(string))
    app.alert("Found Left Parenthesis!",3);
}
\end{insDLJS}
```

This problem is not limited to regular expressions. Anywhere there is an escape character ‘\’, there is a potential for problems. For example, if you want to assign the variable `x` a value of ‘\’, you would have to say, within some `insDLJS` environment:

```
var x = "@e\@e\";
app.alert(x, 3);
```

In systems using the `dvipson`/`dvips`, this would expand to `var x = "\\`; in contrast, on systems

using the `pdftex`/`dvipdfm` option, it would expand to `var x = "\\`. Tricky!

Here is a final complete example.

```
\documentclass{article}
\usepackage[dvipdfm]{hyperref}
\usepackage[dvipdfm]{insdljs}

\newcommand\tugHello{Welcome to TUG 2001!}

\begin{insDLJS}{mydljs}{My DLJS}
function HelloWorld()
{
  app.alert("@tugHello", 3);
}
function TestForLParen(string)
{
  if (/@e\@e(/.test(string))
    app.alert("Found Left Parenthesis!",3);
}
\end{insDLJS}

\begin{document}
\begin{Form}

\section[Test of the \texttt{insDLJS} Package]

\begin{itemize}
  \item \PushButton[name=myButton,
    onclick={HelloWorld();}]{Button:}
  \item \TextField
    [name=myText,width=2in,
    keystroke={if(event.willCommit)
      TestForLParen(event.value);}
    ]{Text Field:}
\end{itemize}

\end{Form}
\end{document}
```

A button and a text field are created. Click on the button to get an alert dialog. Enter some text in the field; when the data is committed, the function `TestForLParen` is called, and the field is scanned for a left parenthesis.

**A Note to `pdftex` and `dvipdfm` Users** If you plan to make extensive use of DLJS, and to write complex functions, such as the ones that appear in the `exerquiz` package, the purchase of the full Acrobat suite is *strongly* recommended.

The only way to debug Acrobat JavaScript is through the JavaScript edit window of the Acrobat viewer. If you only use Acrobat Reader, there is very little feedback as to what goes wrong with a particular script; development and debugging of scripts will be a very long and tedious process.

All the JavaScripts in the `exerquiz` package were first written from within the JavaScript editor and

tested. Once the script was operating correctly, it would be copied, and pasted into the `exerquiz` source file for automatic inclusion.

### Final Comments

Acrobat Reader provides a forms capability and a powerful JavaScript engine that can be exploited to create dynamic, interactive PDF documents. The L<sup>A</sup>T<sub>E</sub>X system with its flexible “plug-in” capability (L<sup>A</sup>T<sub>E</sub>X packages) is a solid authoring tool for creating high-quality typeset content. The L<sup>A</sup>T<sub>E</sub>X packages that give access to PDF are

1. The `hyperref`<sup>3</sup> package written by Sebastian Rathz, Heiko Oberdiek, *et al.*, automatically creates bookmarks and cross-reference hyperlinks, allows document information fill-in and provides basic form creation code, to mention a few. This package is fundamental to any author who wants to create PDF documents for distribution over the Web.
2. The `webeq`<sup>4</sup> (D. P. Story) and `pdfscreen`<sup>5</sup> (Radhakrishnan CV) packages are screen ori-

ented packages. With them, an author can develop a document with page size suitable for presentations or on-line viewing. These packages come with other bells and whistles as well.

3. The `exerquiz`<sup>6</sup> package (D. P. Story) defines several environments for creating exercises and quizzes, both multiple choice and fill-in. This package makes extensive use of the form features and (document-level) JavaScript.
4. The `insDLJS` package gives the package or document author an easy and convenient method of including document-level JavaScripts.

There are many other packages (e.g., for graphics insertion, color creation) too numerous to mention.

As a result of the above mentioned packages, as well as the various applications for producing PDF (`pdftex`, `dvipdfm` and the `distiller`), an author now has a fairly complete set of authoring tools for developing such a colorful, visually attractive and highly interactive documents.

---

<sup>3</sup> [CTAN:macros/latex/contrib/supported/hyperref](http://CTAN:macros/latex/contrib/supported/hyperref)

<sup>4</sup> [CTAN:macros/latex/contrib/supported/webeq](http://CTAN:macros/latex/contrib/supported/webeq)

<sup>5</sup> [CTAN:macros/latex/contrib/supported/pdfscreen](http://CTAN:macros/latex/contrib/supported/pdfscreen)

---

<sup>6</sup> [CTAN:macros/latex/contrib/supported/webeq](http://CTAN:macros/latex/contrib/supported/webeq)

# MacQ $\TeX$ : Online self-marking Quizzes, using pdf $\TeX$ and exerquiz

Ross Moore

Mathematics Department, Macquarie University, Sydney

[ross@maths.mq.edu.au](mailto:ross@maths.mq.edu.au)

<http://www.maths.mq.edu.au/~ross/>

Frances Griffin

Mathematics Department, Macquarie University, Sydney

[fgriffin@maths.mq.edu.au](mailto:fgriffin@maths.mq.edu.au)

<http://www.maths.mq.edu.au/~fgriffin/>

## Abstract

The MacQ $\TeX$  quiz system uses JavaScript [1, 9] embedded within PDF format [4] documents to allow students to do multiple-choice style quizzes. The Internet may be used to supply the quiz document, and to record results. But even when not connected, there is immediate feedback as to how many questions were answered correctly and what are the correct answers, as well as providing worked solutions indicating how the correct answers could be deduced.

The highest quality of typesetting is employed in the quizzes by using the  $\TeX$  typesetting software [7], via the pdf $\TeX$  variant [6], to control the generation of the PDF documents [4]. Other software, such as Perl [12] and *Mathematica* [13], can be used to control the production of unique instances of a particular quiz so that each student gets slightly different questions to answer.

## PDF Quizzes

At Macquarie University the Mathematics Department has been developing<sup>1</sup> a web-based system for producing quizzes which allow students to test their knowledge of mathematical ideas commonly used in courses that we teach. Currently these quizzes are used mainly at the most elementary level, for revision of the basic skills which the students should have acquired from courses at high school.

The current version of this quiz facility provides students with a multiple-choice answer quiz, of typically 10–12 questions, as a PDF document [4] downloaded from a web-site (figure 1). This document is an interactive form, controlled using embedded JavaScript [1, 9], which allows a student to read and work with the document, using the Acrobat Reader plug-in [2] to his/her favourite web-browser (figure 2). Figures 2–7 show some views of such a quiz, as it appears to the student before, during and after attempting to answer the questions.

---

<sup>1</sup> This project has received funding via a ‘Targeted Flagship Grant’ from the Center for Flexible Learning, Macquarie University, and the Division of Information and Communication Sciences, Macquarie University as well as an equipment grant from Apple Computer, Australia Pty Ltd, via the Apple Universities Consortium.

In this paper we will concentrate mainly on the  $\TeX$ nical aspects of the MacQ $\TeX$  quiz system. For other aspects of the full system, such as the rationale for using quizzes at all, and features available to an instructor when preparing a set of quizzes for use by students, figures 13–15 show presentation slides prepared<sup>2</sup> for talks at educational meetings.

## pdf $\TeX$ , exerquiz and JavaScript

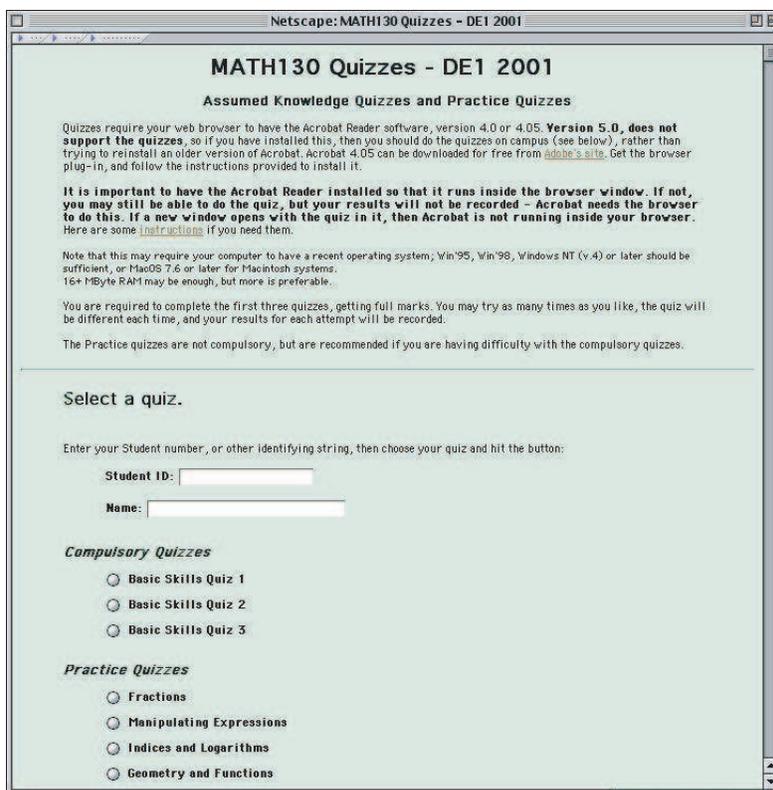
A quiz document is typeset using pdf- $\LaTeX$  [6], with the *exerquiz* [10] macros to handle the embedded JavaScript [1, 9] actions needed to produce appropriate interactivity. In this setting, JavaScript controls

- the appearance of check-boxes, as the student selects his/her answers;
- counting the number of correct choices selected, and displaying an appropriate message;
- showing which of the student’s selections were correct, which were wrong, and which were the correct choices for each question;
- resetting the form, for further attempts at the same set of questions.

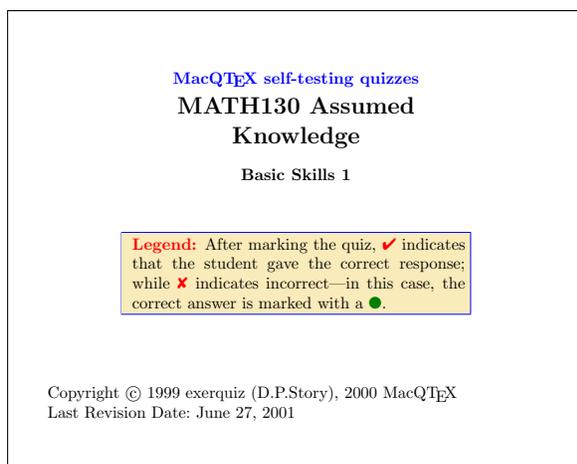
Donald Story [10] has explained some of these methods elsewhere in this volume.

---

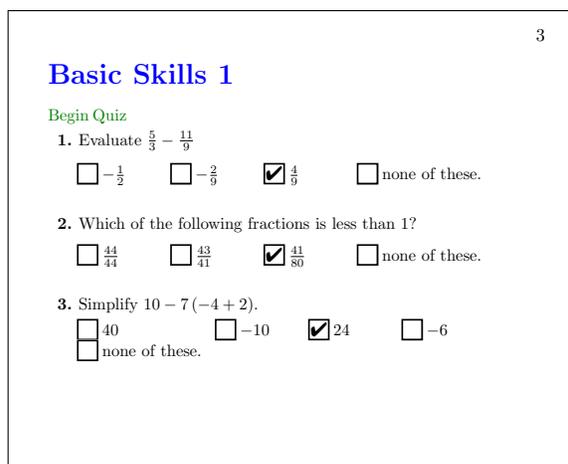
<sup>2</sup> ... using the Marslides package [8].



**Figure 1:** Quiz-site (at <http://www.maths.mq.edu.au/~fgriffin/quizzes/MATH130quizzes.html>) from which students can download the compulsory quiz documents. Username/password are required for recording accesses and results. Also from this site they may download practice quizzes, devoted to a particular mathematical concept. Guest access is also allowed for all quizzes.



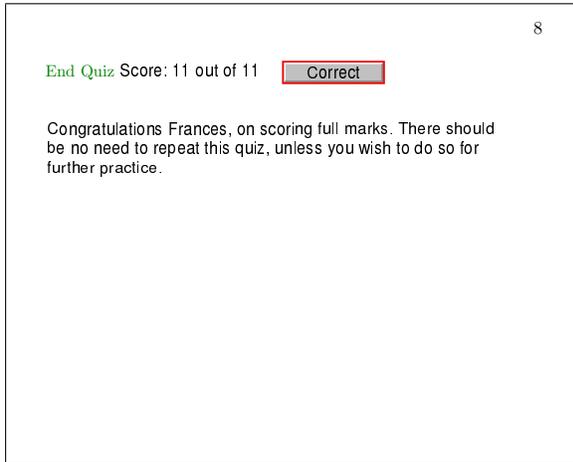
**Figure 2:** An opening page to a typical quiz.



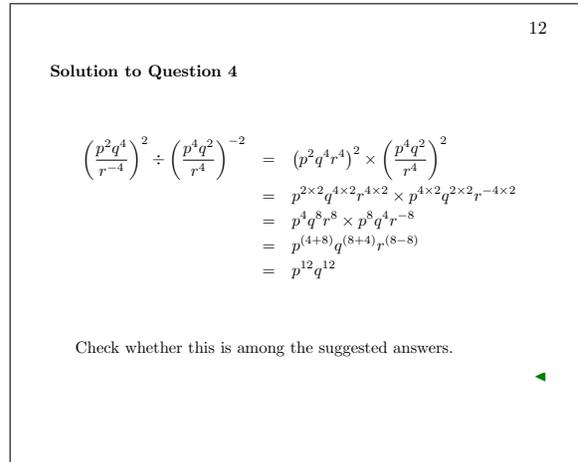
**Figure 3:** First page of questions, with “Begin Quiz” button and user-selections.

In fact we have added some extra features not found in the released versions of exerquiz. Hence the macro file that we actually use is named exerquizX,

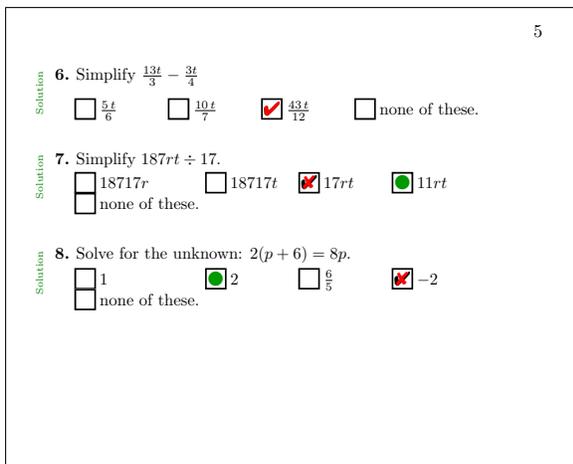
and epdfTeX has the coding specific to the pdfT<sub>E</sub>X driver. These extra features include:



**Figure 4:** Last page of questions, after having selected the “End Quiz” button, showing the total score, and personalised confirmation message.



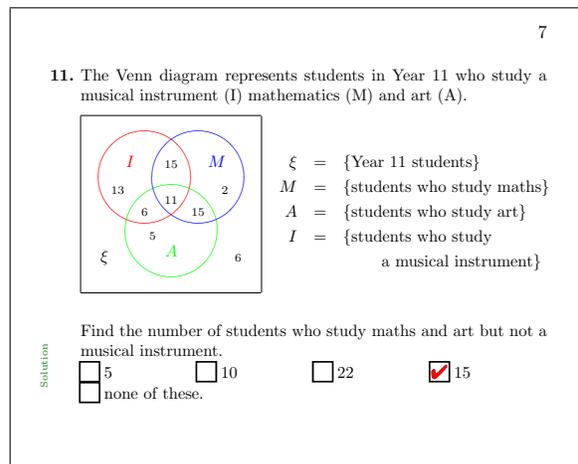
**Figure 6:** Worked solutions use properly typeset mathematics, as do the questions themselves. This one makes substantial use of mathematical symbols and equation alignments.



**Figure 5:** Embedded JavaScript [1, 9] is used to show the correct answer, when the student has made an incorrect choice. Also visible are buttons, previously hidden, which link to worked solutions.

- a quiz variant including both the self-marking feature *and* worked solutions;
- submission of a student’s results to a server, via the Internet;
- return of a personalised message, as feedback to acknowledge receipt of the submitted results.

With these features, a quiz document provides a nicely typeset collection of questions and solutions which can be studied by a student, even when not connected to the Internet. The quiz can be reset for repeated attempts at answering the questions.



**Figure 7:** Elegant mathematical diagrams can also be used, both in the quiz questions and worked solutions.

Only the first attempt can be recorded, and even then only if there is an active network connection.

Concerning the worked solutions, these must remain inaccessible until the quiz has been completed and the results submitted. To do this, the worked solutions are typeset in a separate document, then each is imported as a separate image to be the icon for a button field. This button can be shown or hidden under the control of embedded JavaScript code. As well as allowing the solutions to be viewed, a hyperlink from the question to its solution becomes accessible. This is done by hiding an opaque button

which otherwise screens the active area of the link from receiving mouse-clicks.

Another novel aspect of the MacQ $\TeX$  quiz system is the use of randomness to produce a slightly different quiz for each student access. Currently this is done using the *Mathematica*<sup>3</sup> [13] software, programmed to write a file of  $\TeX$  definitions for each question. Other ways of doing this could be used; e.g., other software could be used, or sets of  $\backslash$ definitions could be read from a file which has been generated in advance specially for this purpose.

### Designing a new quiz

Constructing a new quiz is done at a MacQ $\TeX$  quiz-site, using an HTML form, as shown in figure 8. Here an instructor may choose from pre-prepared question topics; currently there are 14 such types available, covering areas of basic mathematics.

For each topic, there are up to 6 actual question types. Both the number of topics and question types for a topic can be easily extended, though some knowledge of  $\TeX$  or  $\LaTeX$  is needed to do this. When randomisation is required, then some knowledge of programming in *Mathematica* is also useful. For a successful quiz, it is necessary to generate plausible incorrect answers, as well as the correct answer. Generating these in *Mathematica* can be an interesting challenge. Sometimes it is necessary to discard random choices where an answer intended to be wrong actually agrees with the correct answer; that is, obtaining the correct answer by a completely fallacious method.

Figure 9 shows the work-flow for making a new quiz. This includes loops for producing example quizzes, and for editing of  $\LaTeX$  sources for wording and/or layout. Only when the instructor is completely satisfied should a batch (e.g., 50) of quizzes be generated, and made available for student access.

### $\LaTeX$ source code

Figure 10 shows the directory structure at a quiz site. It can be seen that there are many files with  $\text{.tex}$  suffix, which need to be read as part of a typesetting job. Reading all of these files in the correct order is essentially a bootstrap process, in which  $\backslash$ definitions are made as required, before the next file is  $\backslash$ input.

For example, the main job for the quiz which was used for figures 2 to 6 uses a file having the name `MATH130quiz1.tex`, as follows:

```
 $\backslash$ def\recipient{}
```

<sup>3</sup> *Mathematica* is a trademark of Wolfram Research Limited.[13]

```
 $\backslash$ def\defsdire{newquiz50/}
 $\backslash$ def\textdir{../}
 $\backslash$ nonstopmode
 $\backslash$ catcode'\@=11
 $\backslash$ edef\eq@author{\recipient}
 $\backslash$ edef\eq@keywords{version 50}
 $\backslash$ catcode'\@=12
```

```
 $\backslash$ def\loginID{version 50}
 $\backslash$ def\whichquiz{MATH130quiz1}
 $\backslash$ input \textdir user.tex
```

The number 50 that occurs here is because 50 instances of this quiz have been generated. For each instance the number would have been different.

The file `user.tex` is constant, for all quiz instances:

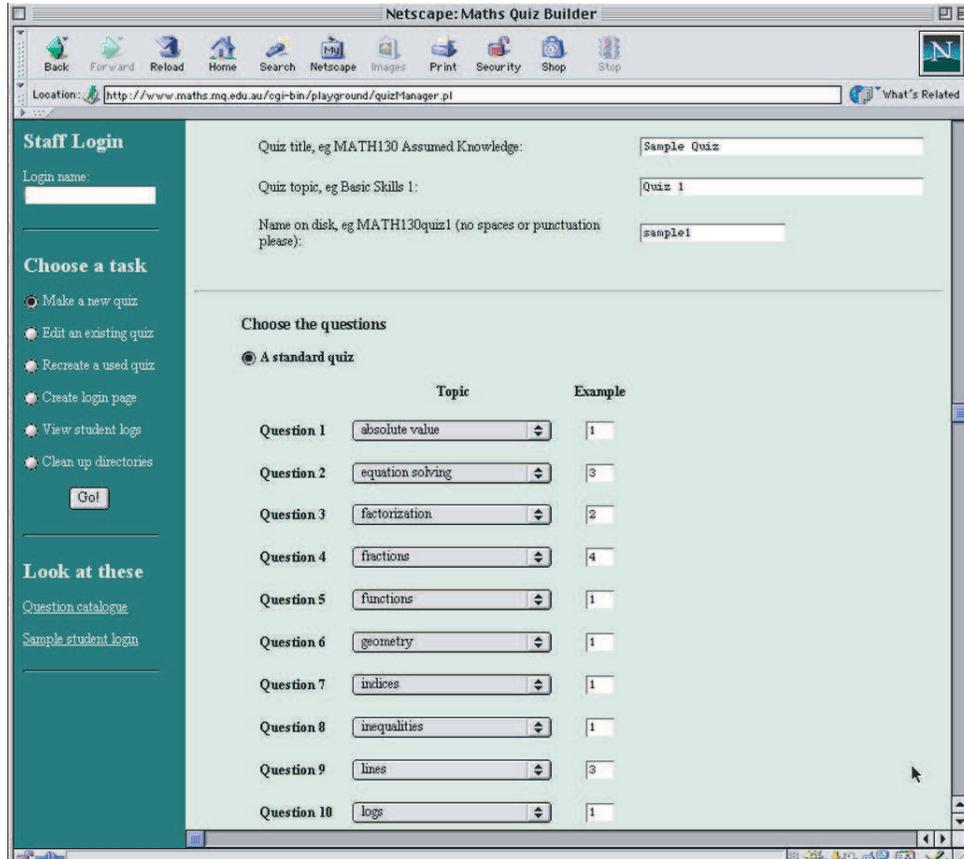
```
 $\backslash$ def\author{Fran}
 $\backslash$ def\imagedir{\textdir}
 $\backslash$ input \textdir a.tex
 $\backslash$ def\quizname{Basic Skills 1}
 $\backslash$ input \textdir b.tex
 $\backslash$ input \textdir bb.tex
 $\backslash$ input \textdir c.tex
 $\backslash$ input \textdir z.tex
```

Those files `a.tex`, `b.tex`, `bb.tex` and `z.tex` are constant for all instances of a particular quiz. Indeed `b.tex` and `c.tex` are created by a Perl [12] script, and hard-code variables such as the title of the quiz and its topic. It is `a.tex` which contains the  $\backslash$ documentclass command, and loads the (modified) `exerquiz` package, as well as other standard  $\LaTeX$  packages. Similarly, the  $\backslash$ end{document} is in `z.tex`. These two files are simply copied from a global storage location.

Information for the opening page of a quiz is contained in `b.tex`. This file could well be edited to alter the instructions, or to convey other information about the quiz. The main information in `bb.tex` is the URL to which results submissions should be sent. Other specialised  $\TeX$  definitions can be added here, when not suitable to be included in other packages. It is thus `c.tex` which controls input of the questions themselves, as follows:

```
% You may edit this file to change the order of
% the questions, or adjust spacing and pagination.
```

```
 $\backslash$ def\answerdir{\defsdire}
 $\backslash$ def\CheckifGiven
{Check whether this is among the suggested answers.}
 $\backslash$ def\bfr{\mathbf{R}}
 $\backslash$ def\bfZ{\mathbf{Z}}
 $\backslash$ def\errOnSend
{Your quiz results have not been received.}
%
 $\backslash$ begin{quiz}*{\whichquiz}
%
 $\backslash$ begin{questions}
```



**Figure 8:** This web-page is used by instructors and course coordinators to design quizzes for the students to use. Questions can be chosen from predefined categories; each category has up to 6 choices of question.

```

\item \input{\texdir question1.tex}
\medskip
\vfil
\goodbreak
\vfilneg
\item \input{\texdir question2.tex}
\medskip
...
...
...
\goodbreak
\vfilneg
\item \input{\texdir question11.tex}
\medskip
\vfil
\goodbreak
\vfilneg
\end{questions}
\end{quiz}
%
%
\TextField[name=\whichquiz,width=1.25in,
  align=0,bordercolor={1 1 1},
  default=Score:,readonly=true]{}%
\raisebox{3.5pt}\quad{\eqButton{\whichquiz}}
\therearequizzesolutionstrue
\medskip

```

```

\TextField[name=progress,height=3cm,width=10cm,
  align=0,bordercolor={1 1 1},default=\errOnSend,
  multiline=true,readonly=true]{%

```

From this it can be seen that the source code for the questions themselves is contained in files named `question1.tex`, ..., `question11.tex`. This also contains the  $\text{\LaTeX}$  source for the worked solution. One of these looks like:

```

\def\setupvalues{%
\def\fracsuma{1}\def\fracsumb{2}\def\fracsumc{5}
\def\fracsumd{9}\def\fracsump{9}\def\fracsumq{2}
\def\fracsumr{18}\def\fracsums{19}
\def\fracsumsign{+}
\def\ok{\frac{\fracsums}{\fracsumr}}
\def\wronga{3}\def\wrongb{\frac{6}{11}}
}

\IfFileExists{\defsdirefs\thequestionno}
{\input{\defsdirefs\thequestionno}}{\setupvalues}

Evaluate

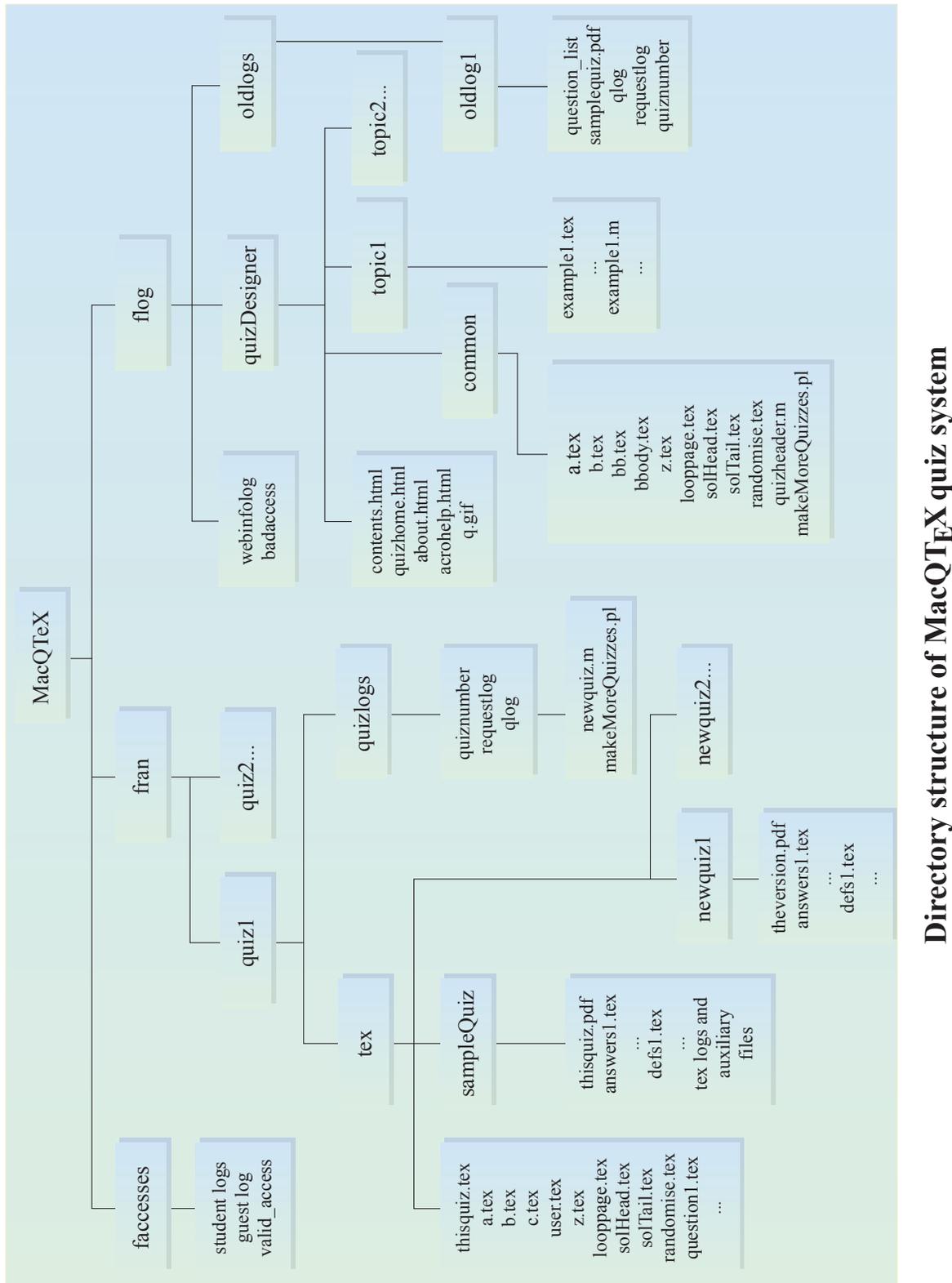
$$\frac{\frac{\frac{suma}{sumb}}{\frac{sumc}{sumd}}}{\frac{sump}{sumq}}$$

\fracsumsign {\frac{\frac{sumc}{sumd}}{\frac{sump}{sumq}}}

\RandomAnswers[123]{\answerdirefs\thequestionno}
\begin{answers}[whichquiz:q\thequestionno]{5}

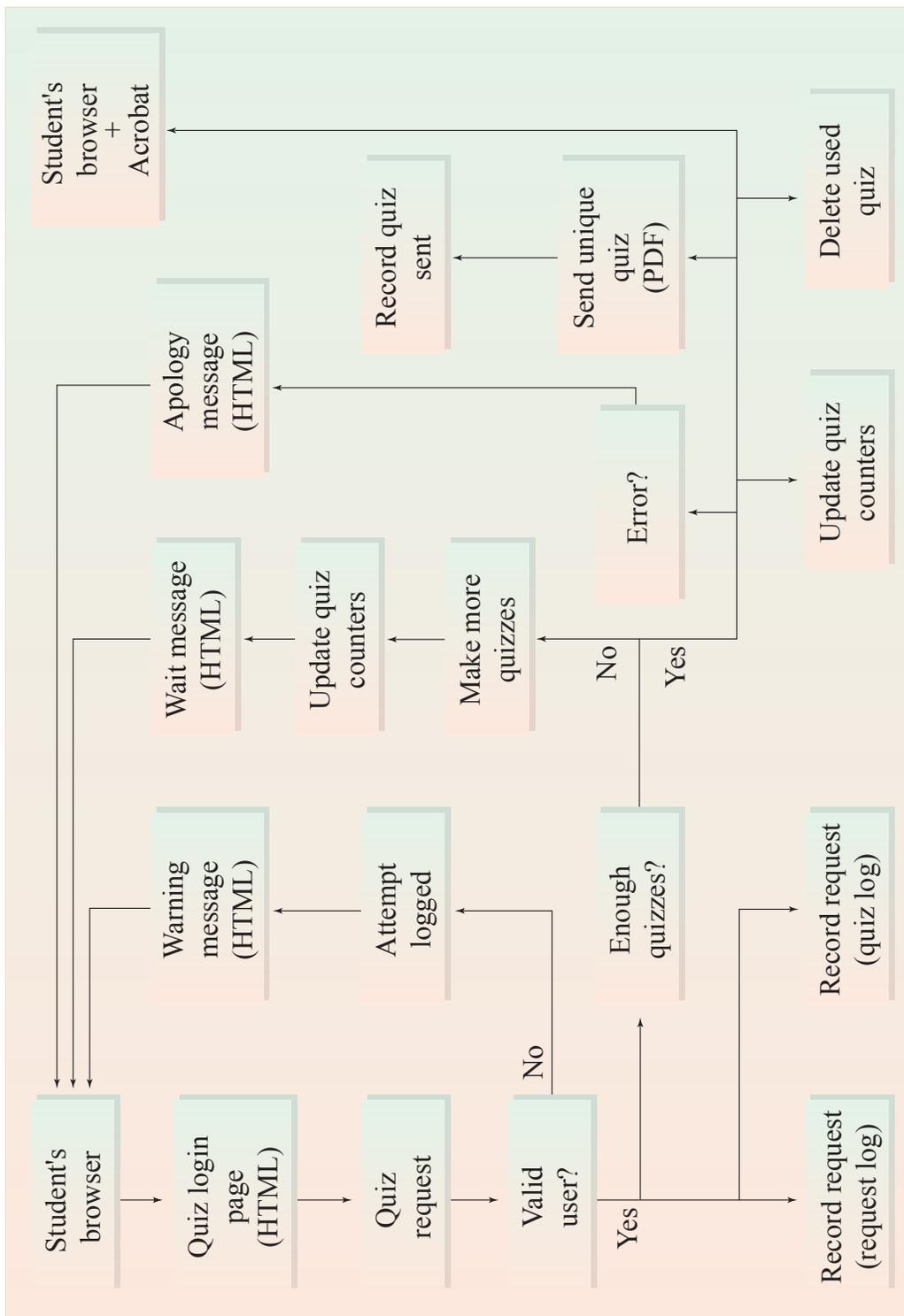
```





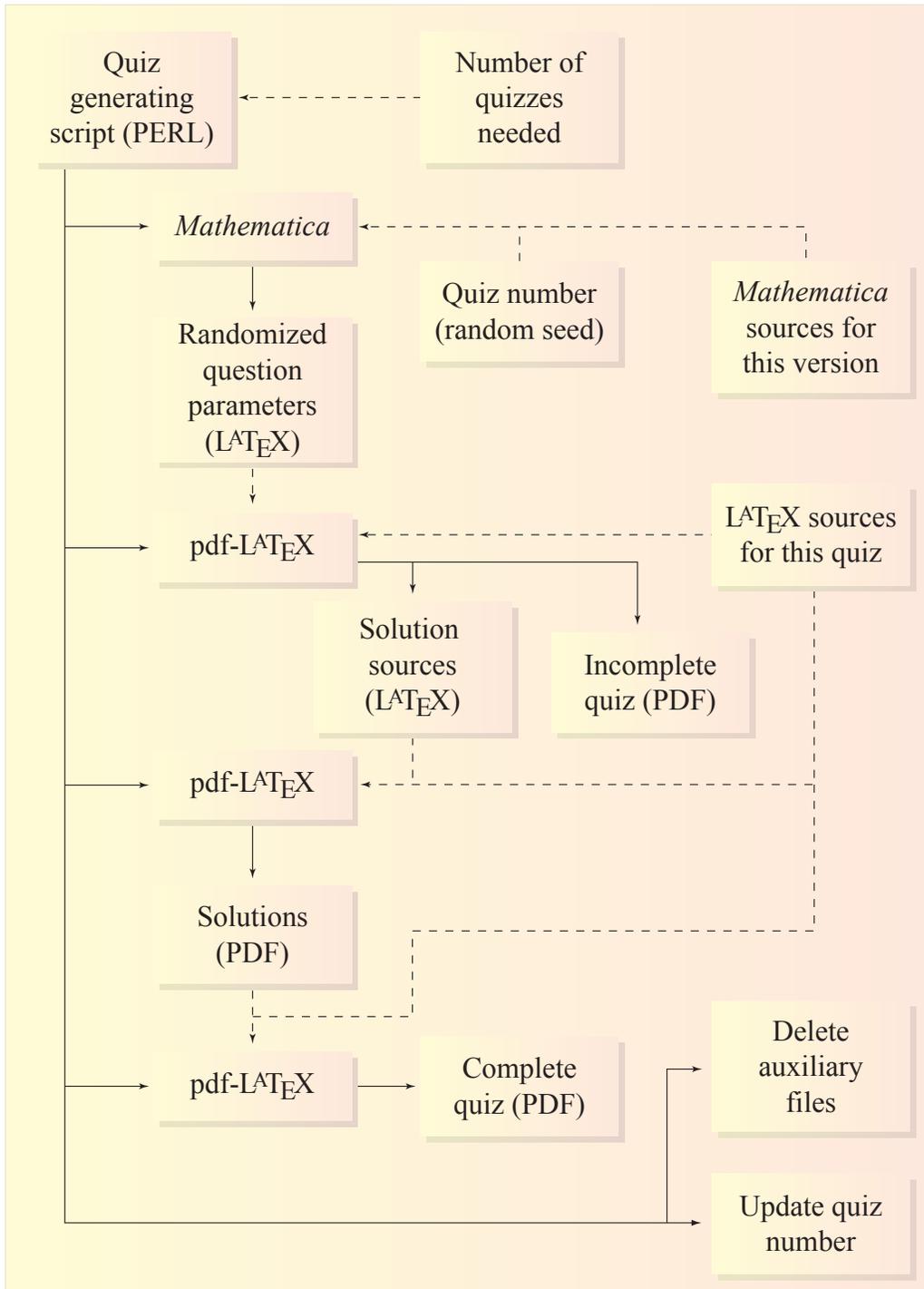
Directory structure of MacQTeX quiz system

**Figure 10:** Directory structure of a quiz site, showing the locations of all the  $(\LaTeX)$  input files for each question and quiz instance.



**Sending a quiz to the student's browser**

**Figure 11:** Detailed work-flow for sending a quiz to the student’s browser. Only at the step “make more quizzes” does pdf-<sub>TEX</sub> come into play within this work-flow, as otherwise sufficiently many quiz documents have been typeset, awaiting requests for downloads. See figure 12 for the work-flow when it is necessary to generate quiz documents.



### Generating new quizzes

**Figure 12:** Details of the sequence of calls to pdf-L<sup>A</sup>T<sub>E</sub>X for generating a quiz instance, along with its worked solutions. The first call causes a new document to be made containing the L<sup>A</sup>T<sub>E</sub>X coding for worked solutions. These are typeset at the 2nd call. Finally the 3rd call constructs a PDF document containing both the quiz questions and each worked solution, imported as a single graphic.



**MacQTEX**  
Randomized Quiz System

**Ross Moore & Frances Griffin**  
Mathematics Department  
Macquarie University, Sydney




### Mathematics Quizzes

At Macquarie University the Mathematics Department has been developing<sup>1</sup> a web-based system for producing quizzes which allow students to test their knowledge of mathematical ideas required in the courses that we teach. Currently these quizzes are used mainly at the most elementary level, for revision of the basic skills which the students should have acquired from mathematics courses at high school.

**Examples**

- **Basic Math Skills:** [MATH130, examples](#)
- **Discrete Math Quizzes, with answers:** [MATH237](#)

<sup>1</sup>This project has received funding via a 'Targeted Flagship Grant' from the Center for Flexible Learning, Macquarie University, and the Division of Information and Communication Sciences, Macquarie University as well as an equipment grant from Apple Computer, Australia Pty Ltd, via the Apple Universities Consortium.

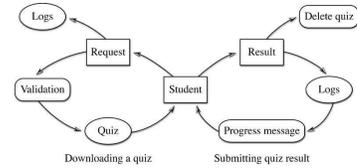
### Training not Testing

The aim of the quizzes is not so much for assessment as for self-testing and practice of material covered previously in lectures or back at school. Why use quizzes?  
For various practical reasons, related to the students ...

- wide range in ability and mathematical background;
- not their main area of study, so need to identify holes in their mathematical knowledge;
- several years since last studied any mathematics;
- maybe only a refresher course may be needed;
- insufficient staff to help every student.

The quizzes can be used by students to **identify for themselves** where they are weak and may need to seek the extra help that *can* be provided.

### Student Interaction



When a student requests a quiz, the identity is first validated; if authorized, a quiz is sent to the student's browser. A record is kept of all request details.

After completing the quiz, submitted results are recorded in the student's log and in the overall quiz log. A personalized message concerning the student's progress is returned as FDF data. This appears in a form field at the end of the PDF quiz document.

**Figure 13:** Presentation slides, mainly for use at education meetings. These give the rationale for using quizzes, as well as giving a rough representation of a student's interaction with the system. Also one slide has active hyperlinks to some actual web pages from whence quizzes can be downloaded.

```

\theAnswersStream
%
\multicolumn{2}{1}{\Ans0 none of these. \hfil}
\end{answers}
%
\begin{solution}
\medskip

\def\setupvalues{%
\def\fracsuma{1}\def\fracsumb{2}\def\fracsumc{5}
\def\fracsumd{9}\def\fracsump{9}\def\fracsumq{2}
\def\fracsumr{18}\def\fracsums{19}
\def\fracsumsign{+}
\def\ok{\frac{\fracsums}{\fracsumr}}
\def\wronga{3}\def\wrongb{\frac{6}{11}}

\addtocounter{solutionno}{1}

\IfFileExists{\defsdirefs\thesolutionno}{
\input{\defsdirefs\thesolutionno}}{\setupvalues}

Using the common denominator \fracsumr
$$\frac{\fracsuma}{\fracsumb}
\fracsumsign \frac{\fracsumc}{\fracsumd}
=\frac{\fracsuma\times\fracsump}{\fracsumr}
\fracsumsign

```

```

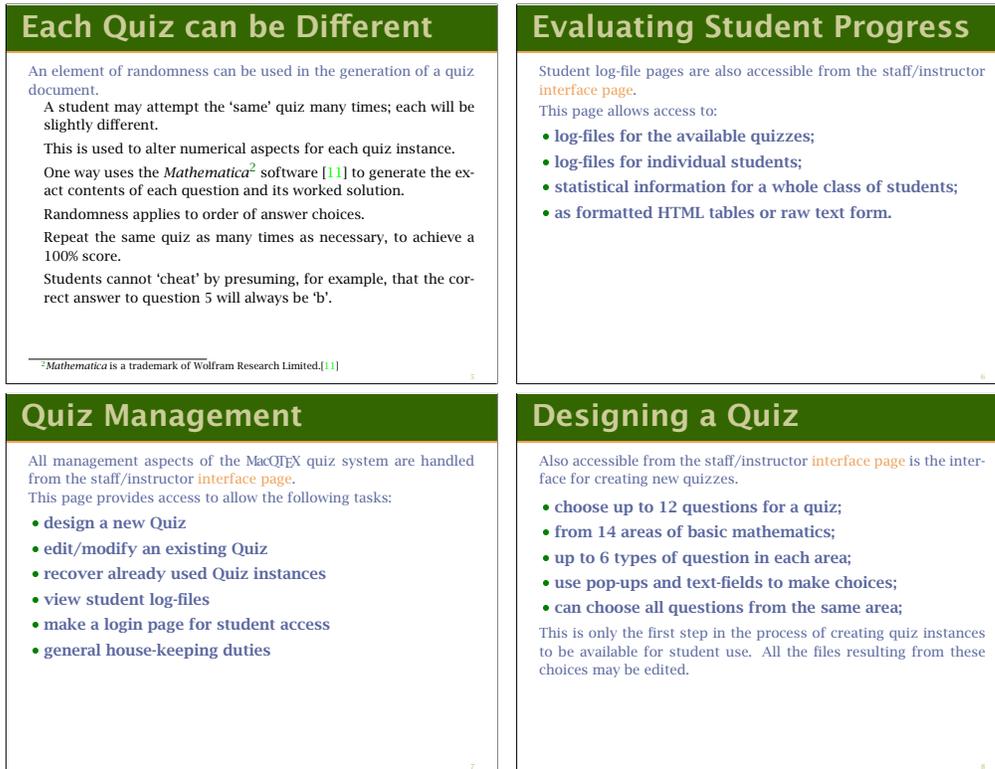
\frac{\fracsumc\times\fracsumq}{\fracsumr}
=\ok.
$$
\medskip

\CheckifGiven
\end{solution}
\medskip

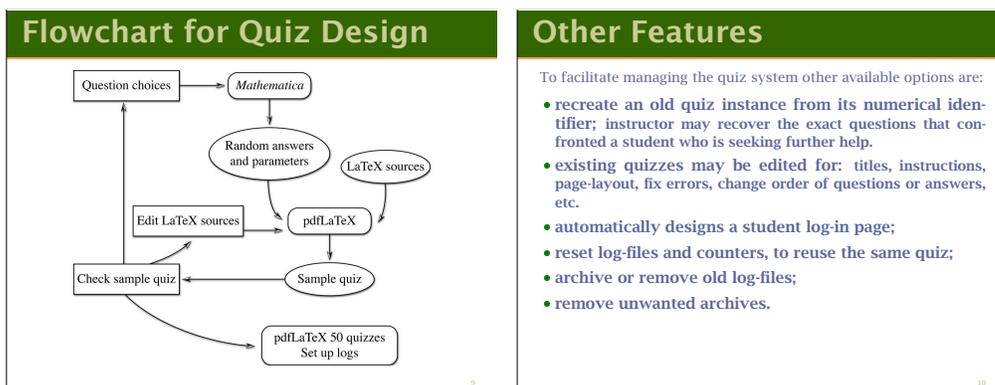
```

In the above T<sub>E</sub>X coding, the role of `\setupvalues` is to provide default definitions, just in case there is no file `defs\thequestionno` (that is `defs1`, `defs2`, ... according to the question number) within the `\defsdirefs` directory on the local file-system. If it does exist, (e.g., having been written by *Mathematica*<sup>4</sup>, after performing calculations employing a random-number seed) then it should contain the necessary `\definitions`. Similarly there can be a file `\answerdirefs\thequestionno` to govern

<sup>4</sup> *Mathematica* is a trademark of Wolfram Research Limited.[13]



**Figure 14:** These presentation slides indicate how a random aspect is incorporated into the quizzes. Also shown are other aspects of generating quizzes, obtaining statistical information from the log-files, and general house-keeping duties that can be performed.



**Figure 15:** One slide here gives a rough view of the work-flow involved when preparing a set of quizzes for student use. A more detailed work-flow is given in figure 11.

the order in which the answers are presented in the quiz question.

## References

- [1] Adobe Systems Inc.; “Acrobat Forms JavaScript Object Specification, Version 4.0”; Technical Note #5186; Revised: January 27, 1999.
- [2] Adobe Systems Inc.; Acrobat Reader, viewer for PDF format [4] documents, available free of charge from <http://www.adobe.com/>.
- [3] Adobe Systems Inc.; “PDF Toolkit Overview”; Technical Note #5194; Revised: August 10, 1999.
- [4] Adobe Systems Inc.; “Portable Document Format, Reference Manual, Version 1.3”; March 11, 1999.
- [5] Adobe Systems Inc.; “pdfmark Reference Manual”; Technical Note #5150; Adobe Developer Relations; Revised: March 4, 1999.
- [6] Hàn, Thê Thành; pdf $\TeX$ , free software for generating documents in PDF format, based on the  $\TeX$  typesetting system. Available for all computing platforms; see <http://www.tug.org/applications/pdftex/>.
- [7] Lamport, Leslie; L $\TeX$ , a Document Preparation System. This is free software available for all computing platforms. Consult the  $\TeX$  Users Group (TUG) web site, at <http://www.tug.org/>.
- [8] McKay, Wendy and Moore, Ross; “PDF presentations using the Marslide package.”  $\TeX$  Users Group 2001 Proceedings, (elsewhere in this volume).
- [9] Netscape Communications Corporation; Netscape JavaScript Reference, 1997; online at: <http://developer.netscape.com/docs/manuals/communicator/jsref/toc.htm>.
- [10] Story, Donald; exerquiz & Acro $\TeX$ , packages for including special effects in PDF documents, using  $\TeX$  and L $\TeX$ . Dept. of Mathematics and Computer Science, University of Akron. Software available online from <http://www.math.uakron.edu/~dpstory/webeq.html>.
- [10] Story, Donald; “Techniques of Introducing Document-level JavaScript into a PDF file from a L $\TeX$  source.”  $\TeX$  Users Group 2001 Proceedings, (elsewhere in this volume).
- [12] Wall, Larry; *Perl*, a general purpose scripting language for all computing platforms. This is Free Software, available from <http://www.perl.com/>.
- [13] Wolfram Research Inc; *Mathematica*, a system for doing Mathematics by computer. Consult the web site at <http://www.wri.com/>.

# MacQ $\TeX$ : Online self-marking Quizzes, using pdf $\TeX$ and *exerquiz*

Ross Moore

Mathematics Department, Macquarie University, Sydney

[ross@maths.mq.edu.au](mailto:ross@maths.mq.edu.au)

<http://www.maths.mq.edu.au/~ross/>

Frances Griffin

Mathematics Department, Macquarie University, Sydney

[fgriffin@maths.mq.edu.au](mailto:fgriffin@maths.mq.edu.au)

<http://www.maths.mq.edu.au/~fgriffin/>

## Abstract

The MacQ $\TeX$  quiz system uses JavaScript [1, 9] embedded within PDF format [4] documents to allow students to do multiple-choice style quizzes. The Internet may be used to supply the quiz document, and to record results. But even when not connected, there is immediate feedback as to how many questions were answered correctly and what are the correct answers, as well as providing worked solutions indicating how the correct answers could be deduced.

The highest quality of typesetting is employed in the quizzes by using the  $\TeX$  typesetting software [7], via the pdf $\TeX$  variant [6], to control the generation of the PDF documents [4]. Other software, such as Perl [12] and *Mathematica* [13], can be used to control the production of unique instances of a particular quiz so that each student gets slightly different questions to answer.

## PDF Quizzes

At Macquarie University the Mathematics Department has been developing<sup>1</sup> a web-based system for producing quizzes which allow students to test their knowledge of mathematical ideas commonly used in courses that we teach. Currently these quizzes are used mainly at the most elementary level, for revision of the basic skills which the students should have acquired from courses at high school.

The current version of this quiz facility provides students with a multiple-choice answer quiz, of typically 10–12 questions, as a PDF document [4] downloaded from a web-site (figure 1). This document is an interactive form, controlled using embedded JavaScript [1, 9], which allows a student to read and work with the document, using the Acrobat Reader plug-in [2] to his/her favourite web-browser (figure 2). Figures 2–7 show some views of such a quiz, as it appears to the student before, during and after attempting to answer the questions.

---

<sup>1</sup> This project has received funding via a ‘Targeted Flagship Grant’ from the Center for Flexible Learning, Macquarie University, and the Division of Information and Communication Sciences, Macquarie University as well as an equipment grant from Apple Computer, Australia Pty Ltd, via the Apple Universities Consortium.

In this paper we will concentrate mainly on the  $\TeX$ ical aspects of the MacQ $\TeX$  quiz system. For other aspects of the full system, such as the rationale for using quizzes at all, and features available to an instructor when preparing a set of quizzes for use by students, figures 13–15 show presentation slides prepared<sup>2</sup> for talks at educational meetings.

## pdf $\TeX$ , *exerquiz* and JavaScript

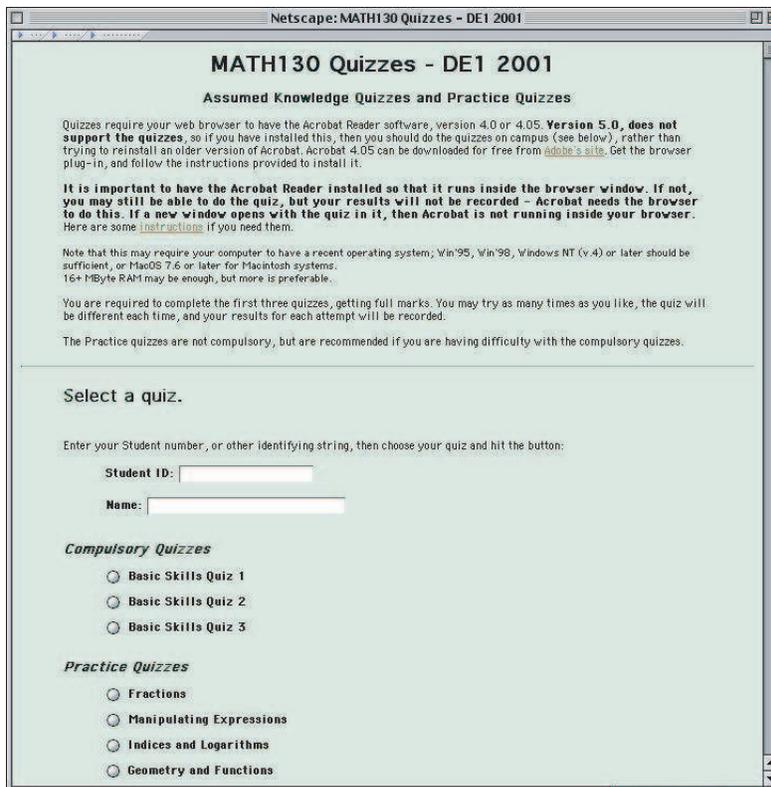
A quiz document is typeset using pdf $\LaTeX$  [6], with the *exerquiz* [10] macros to handle the embedded JavaScript [1, 9] actions needed to produce appropriate interactivity. In this setting, JavaScript controls

- the appearance of check-boxes, as the student selects his/her answers;
- counting the number of correct choices selected, and displaying an appropriate message;
- showing which of the student’s selections were correct, which were wrong, and which were the correct choices for each question;
- resetting the form, for further attempts at the same set of questions.

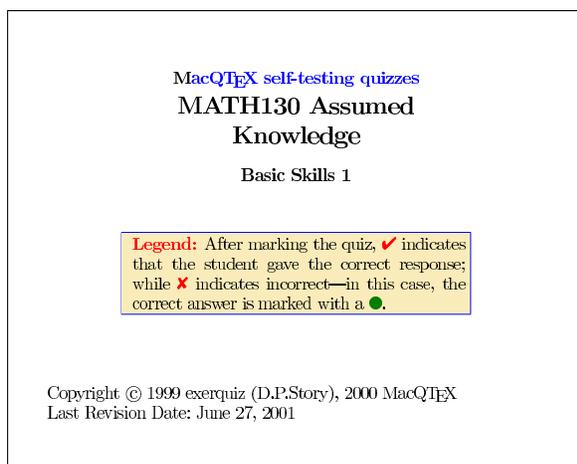
Donald Story [10] has explained some of these methods elsewhere in this volume.

---

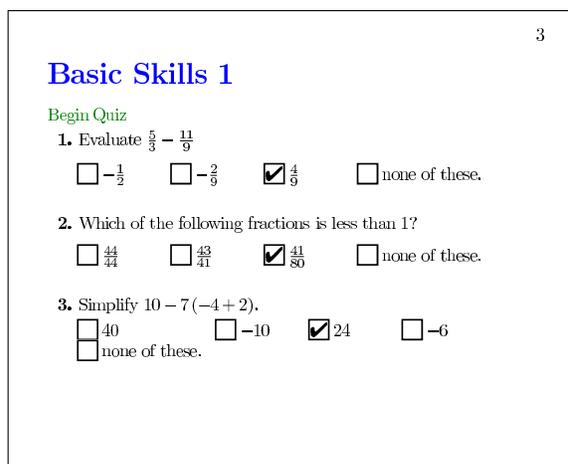
<sup>2</sup> ... using the Marslides package [8].



**Figure 1:** Quiz-site (at <http://www.maths.mq.edu.au/~fgriffin/quizzes/MATH130quizzes.html>) from which students can download the compulsory quiz documents. Username/password are required for recording accesses and results. Also from this site they may download practice quizzes, devoted to a particular mathematical concept. Guest access is also allowed for all quizzes.



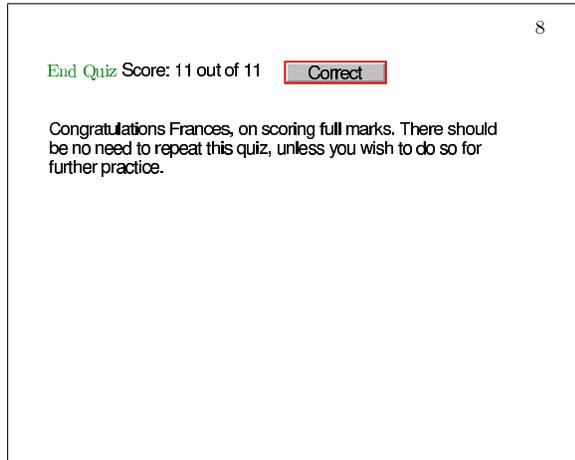
**Figure 2:** An opening page to a typical quiz.



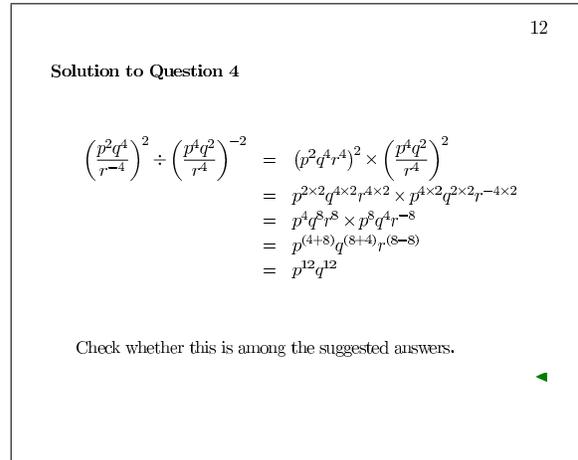
**Figure 3:** First page of questions, with “Begin Quiz” button and user-selections.

In fact we have added some extra features not found in the released versions of *exerquiz*. Hence the macro file that we actually use is named *exerquizX*,

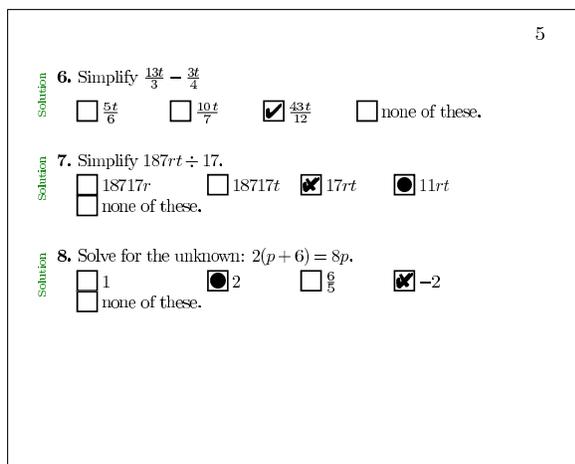
and *epdfTeX* has the coding specific to the pdf<sub>T</sub>E<sub>X</sub> driver. These extra features include:



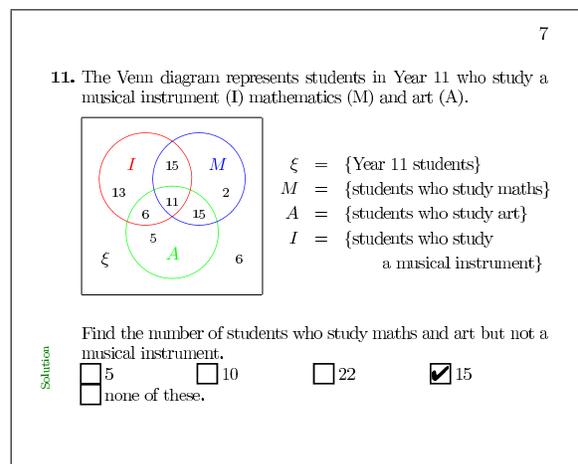
**Figure 4:** Last page of questions, after having selected the “End Quiz” button, showing the total score, and personalised confirmation message.



**Figure 6:** Worked solutions use properly typeset mathematics, as do the questions themselves. This one makes substantial use of mathematical symbols and equation alignments.



**Figure 5:** Embedded JavaScript [1, 9] is used to show the correct answer, when the student has made an incorrect choice. Also visible are buttons, previously hidden, which link to worked solutions.



**Figure 7:** Elegant mathematical diagrams can also be used, both in the quiz questions and worked solutions.

- a quiz variant including both the self-marking feature *and* worked solutions;
- submission of a student’s results to a server, via the Internet;
- return of a personalised message, as feedback to acknowledge receipt of the submitted results.

With these features, a quiz document provides a nicely typeset collection of questions and solutions which can be studied by a student, even when not connected to the Internet. The quiz can be reset for repeated attempts at answering the questions.

Only the first attempt can be recorded, and even then only if there is an active network connection.

Concerning the worked solutions, these must remain inaccessible until the quiz has been completed and the results submitted. To do this, the worked solutions are typeset in a separate document, then each is imported as a separate image to be the icon for a button field. This button can be shown or hidden under the control of embedded JavaScript code. As well as allowing the solutions to be viewed, a hyperlink from the question to its solution becomes accessible. This is done by hiding an opaque button

which otherwise screens the active area of the link from receiving mouse-clicks.

Another novel aspect of the MacQ $\TeX$  quiz system is the use of randomness to produce a slightly different quiz for each student access. Currently this is done using the *Mathematica*<sup>3</sup> [13] software, programmed to write a file of  $\TeX$  definitions for each question. Other ways of doing this could be used; e.g., other software could be used, or sets of  $\TeX$  definitions could be read from a file which has been generated in advance specially for this purpose.

### Designing a new quiz

Constructing a new quiz is done at a MacQ $\TeX$  quiz-site, using an HTML form, as shown in figure 8. Here an instructor may choose from pre-prepared question topics; currently there are 14 such types available, covering areas of basic mathematics.

For each topic, there are up to 6 actual question types. Both the number of topics and question types for a topic can be easily extended, though some knowledge of  $\TeX$  or  $\LaTeX$  is needed to do this. When randomisation is required, then some knowledge of programming in *Mathematica* is also useful. For a successful quiz, it is necessary to generate plausible incorrect answers, as well as the correct answer. Generating these in *Mathematica* can be an interesting challenge. Sometimes it is necessary to discard random choices where an answer intended to be wrong actually agrees with the correct answer; that is, obtaining the correct answer by a completely fallacious method.

Figure 9 shows the work-flow for making a new quiz. This includes loops for producing example quizzes, and for editing of  $\LaTeX$  sources for wording and/or layout. Only when the instructor is completely satisfied should a batch (e.g., 50) of quizzes be generated, and made available for student access.

### $\LaTeX$ source code

Figure 10 shows the directory structure at a quiz site. It can be seen that there are many files with `.tex` suffix, which need to be read as part of a typesetting job. Reading all of these files in the correct order is essentially a bootstrap process, in which  $\TeX$  definitions are made as required, before the next file is `\input`.

For example, the main job for the quiz which was used for figures 2 to 6 uses a file having the name `MATH130quiz1.tex`, as follows:

```
\def\recipient{}
```

<sup>3</sup> *Mathematica* is a trademark of Wolfram Research Limited.[13]

```
\def\defsdire{newquiz50/}
\def\texdir{../}
\nonstopmode
\catcode'\@=11
\edef\eq@author{\recipient}
\edef\eq@keywords{version 50}
\catcode'\@=12
```

```
\def\loginID{version 50}
\def\whichquiz{MATH130quiz1}
\input \texdir user.tex
```

The number 50 that occurs here is because 50 instances of this quiz have been generated. For each instance the number would have been different.

The file `user.tex` is constant, for all quiz instances:

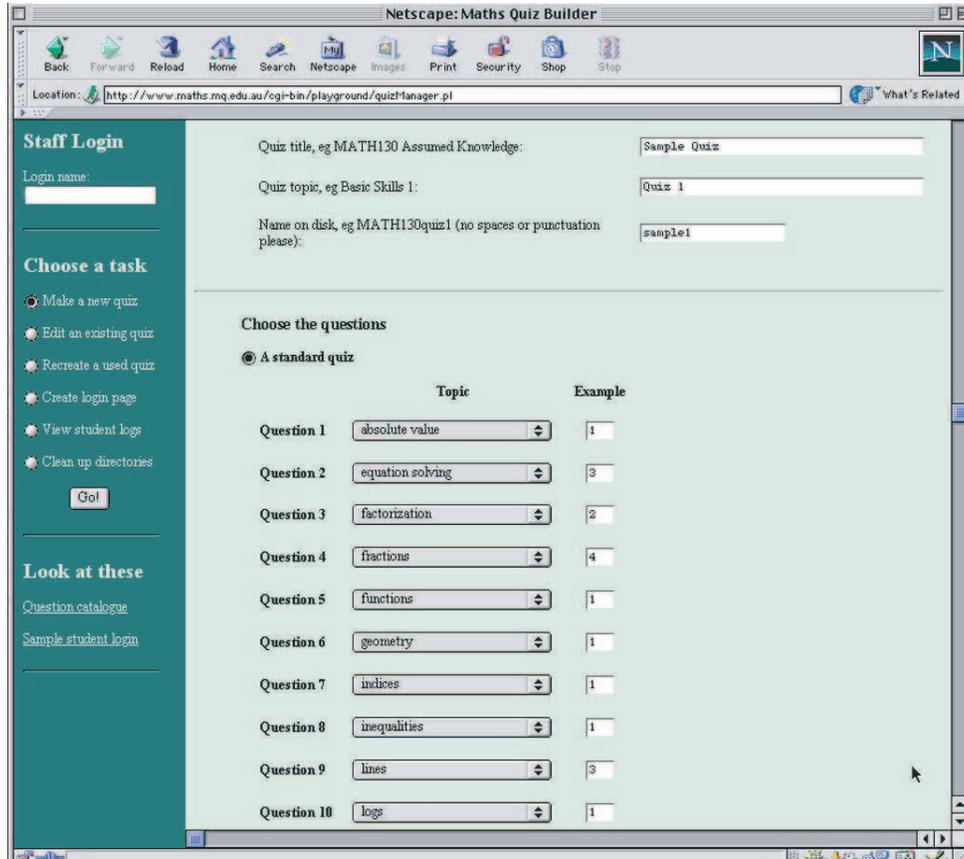
```
\def\author{Fran}
\def\imagedir{\texdir}
\input \texdir a.tex
\def\quizname{Basic Skills 1}
\input \texdir b.tex
\input \texdir bb.tex
\input \texdir c.tex
\input \texdir z.tex
```

Those files `a.tex`, `b.tex`, `bb.tex` and `z.tex` are constant for all instances of a particular quiz. Indeed `b.tex` and `c.tex` are created by a Perl [12] script, and hard-code variables such as the title of the quiz and its topic. It is `a.tex` which contains the `\documentclass` command, and loads the (modified) `exerquiz` package, as well as other standard  $\LaTeX$  packages. Similarly, the `\end{document}` is in `z.tex`. These two files are simply copied from a global storage location.

Information for the opening page of a quiz is contained in `b.tex`. This file could well be edited to alter the instructions, or to convey other information about the quiz. The main information in `bb.tex` is the URL to which results submissions should be sent. Other specialised  $\TeX$  definitions can be added here, when not suitable to be included in other packages. It is thus `c.tex` which controls input of the questions themselves, as follows:

```
%% You may edit this file to change the order of
% the questions, or adjust spacing and pagination.

\def\answerdir{\defsdire}
\def\CheckifGiven
{Check whether this is among the suggested answers.}
\def\bfr{\mathbf{R}}
\def\bfZ{\mathbf{Z}}
\def\errOnSend
{Your quiz results have not been received.}
%
\begin{quiz}*\{whichquiz}
%
\begin{questions}
```



**Figure 8:** This web-page is used by instructors and course coordinators to design quizzes for the students to use. Questions can be chosen from predefined categories; each category has up to 6 choices of question.

```

\item \input{\texdir question1.tex}
\medskip
\vfil
\goodbreak
\vfilneg
\item \input{\texdir question2.tex}
\medskip
...
...
...
\goodbreak
\vfilneg
\item \input{\texdir question11.tex}
\medskip
\vfil
\goodbreak
\vfilneg
\end{questions}
\end{quiz}
%
%
\TextField[name=\whichquiz,width=1.25in,
  align=0,bordercolor={1 1 1},
  default=Score:,readonly=true]{}%
\raisebox{3.5pt}\quad{\eqButton{\whichquiz}}
\therearequizsolutionstrue
\medskip

```

```

\TextField[name=progress,height=3cm,width=10cm,
  align=0,bordercolor={1 1 1},default=\errOnSend,
  multiline=true,readonly=true]{ }

```

From this it can be seen that the source code for the questions themselves is contained in files named question1.tex, ..., question11.tex. This also contains the L<sup>A</sup>T<sub>E</sub>X source for the worked solution. One of these looks like:

```

\def\setupvalues{%
\def\fracsuma{1}\def\fracsumb{2}\def\fracsumc{5}
\def\fracsumd{9}\def\fracsump{9}\def\fracsumq{2}
\def\fracsumr{18}\def\fracsums{19}
\def\fracsumsign{+}
\def\ok{\frac{\fracsums}{\fracsumr}}
\def\wronga{3}\def\wrongb{\frac{6}{11}}
}

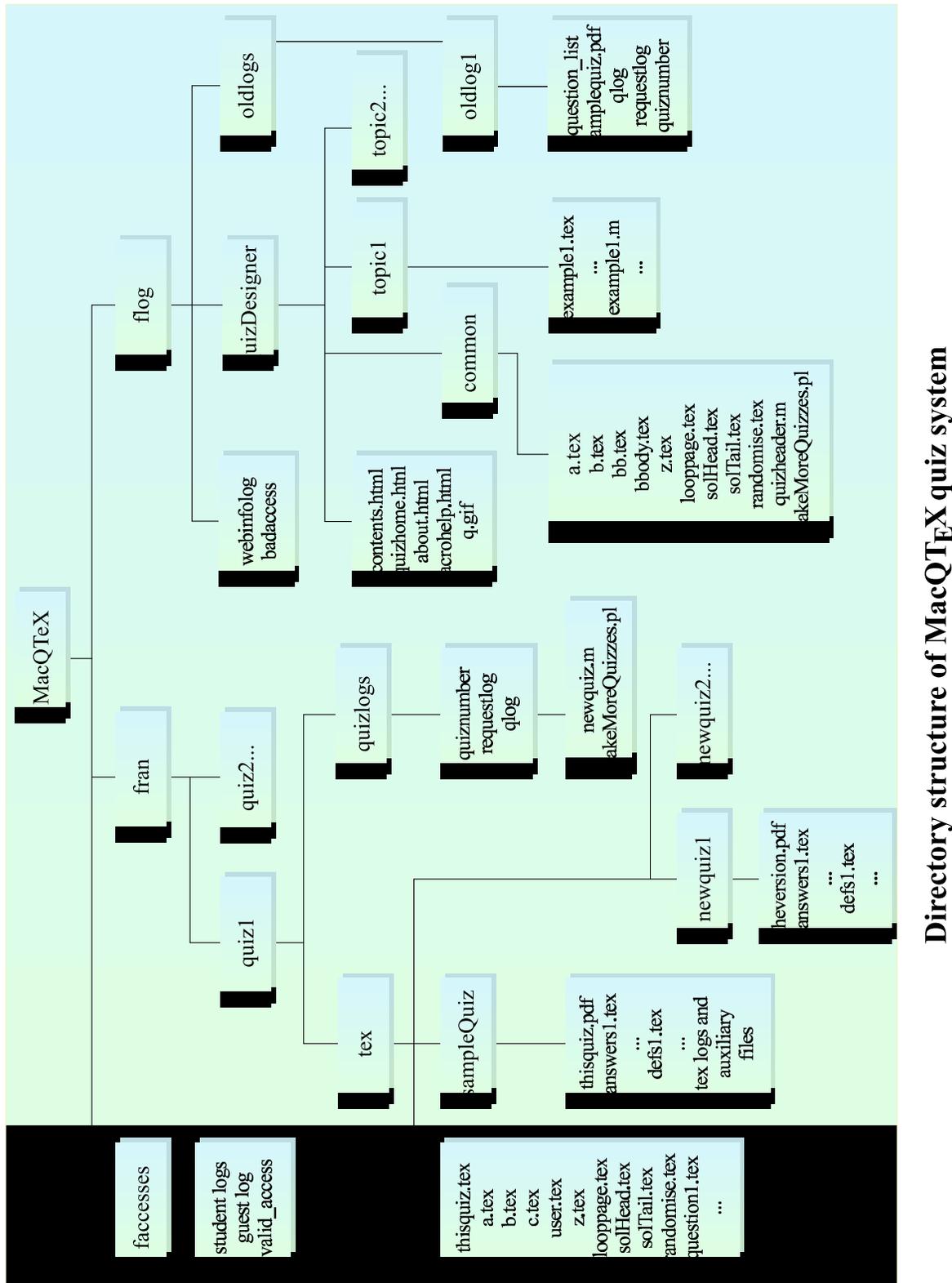
\IfFileExists{\defsdirefs\thequestionno}
{\input{\defsdirefs\thequestionno}}{\setupvalues}

Evaluate
${\frac{\fracsuma}{\fracsumb}}
\fracsumsign {\frac{\fracsumc}{\fracsumd}}$

\RandomAnswers[123]{\answerdirefs\thequestionno}
\begin{answers}[{\whichquiz:q\thequestionno}]{5}

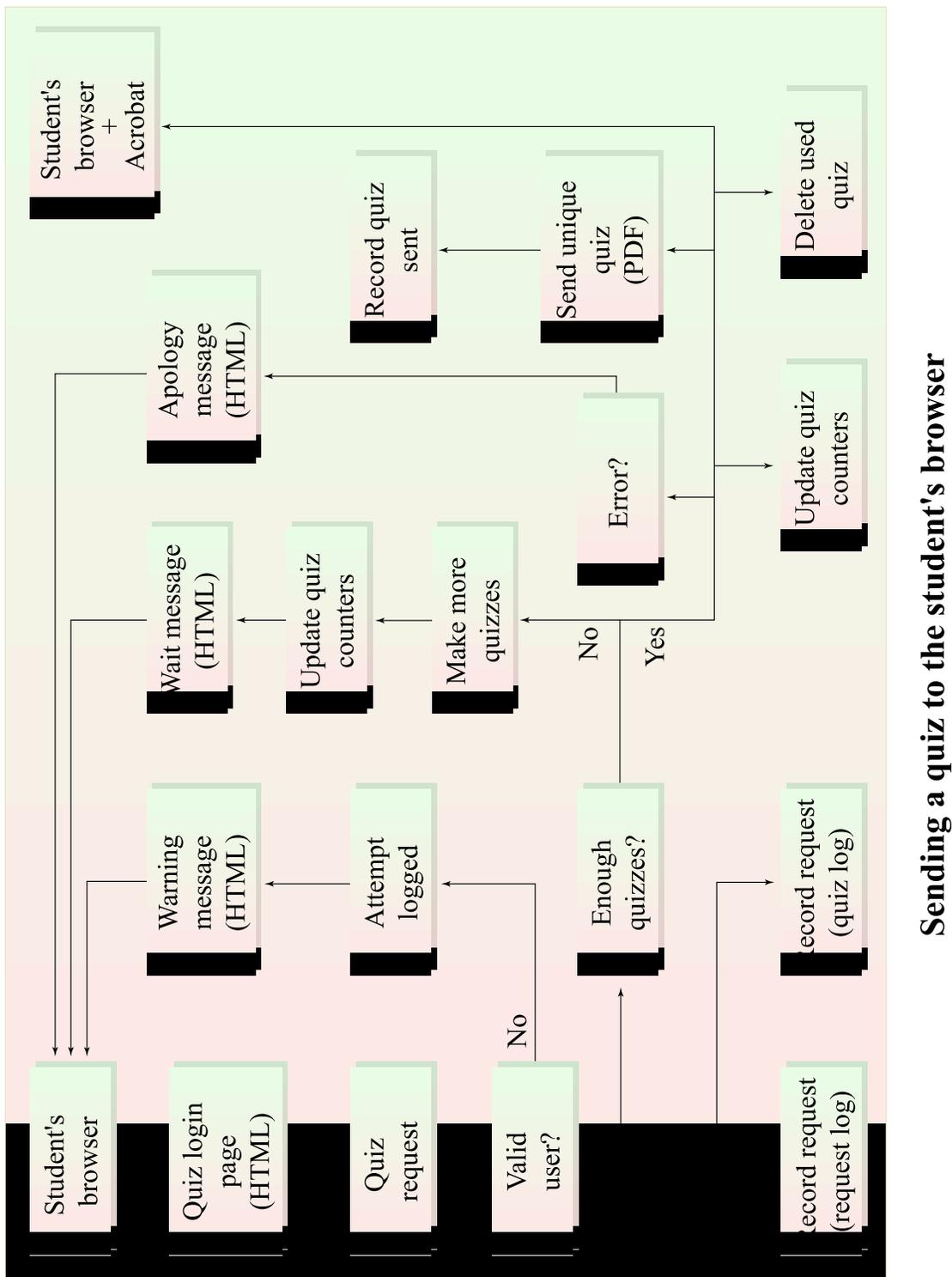
```



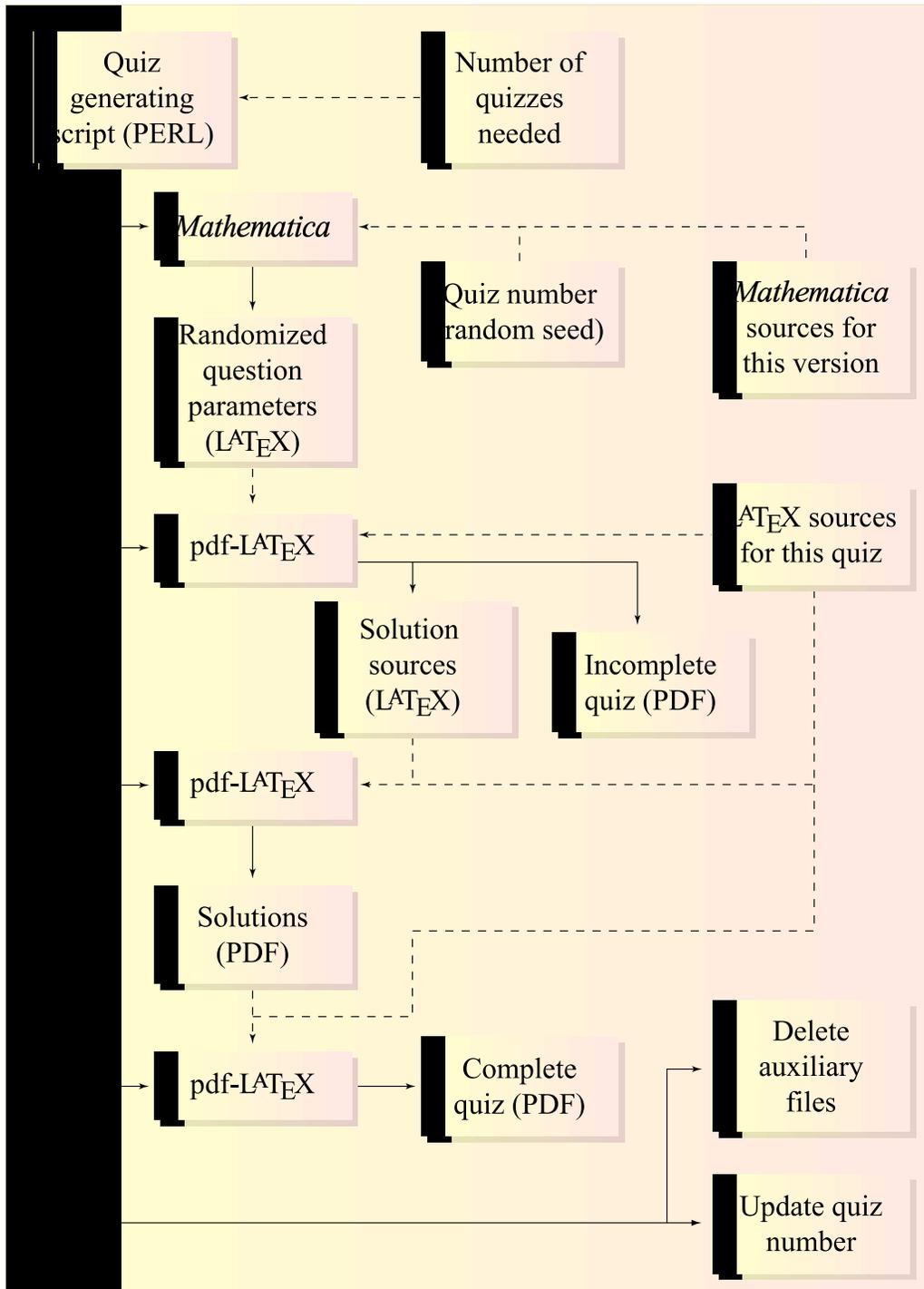


Directory structure of MacQ<sub>TEX</sub> quiz system

**Figure 10:** Directory structure of a quiz site, showing the locations of all the (L<sup>A</sup>)<sub>TEX</sub> input files for each question and quiz instance.

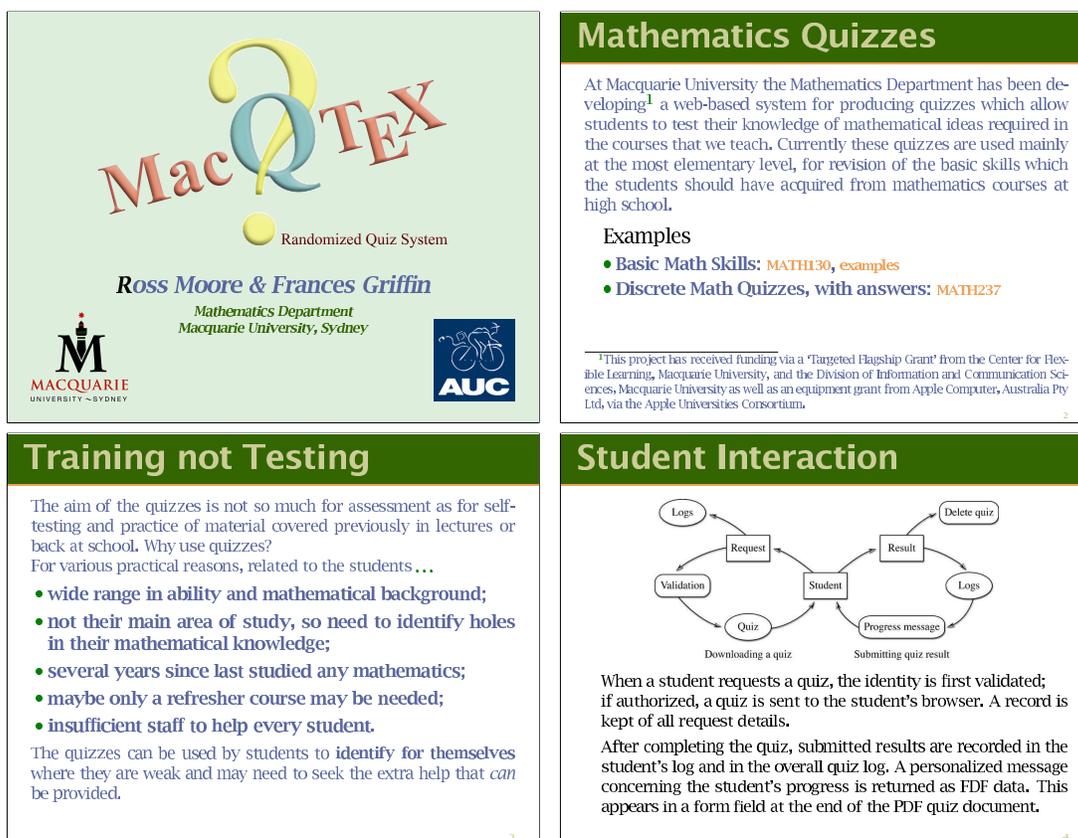


**Figure 11:** Detailed work-flow for sending a quiz to the student’s browser. Only at the step “make more quizzes” does pdf-<sub>TEX</sub> come into play within this work-flow, as otherwise sufficiently many quiz documents have been typeset, awaiting requests for downloads. See figure 12 for the work-flow when it is necessary to generate quiz documents.



### Generating new quizzes

**Figure 12:** Details of the sequence of calls to pdf-LAT<sub>E</sub>X for generating a quiz instance, along with its worked solutions. The first call causes a new document to be made containing the L<sub>A</sub>T<sub>E</sub>X coding for worked solutions. These are typeset at the 2nd call. Finally the 3rd call constructs a PDF document containing both the quiz questions and each worked solution, imported as a single graphic.



**Figure 13:** Presentation slides, mainly for use at education meetings. These give the rationale for using quizzes, as well as giving a rough representation of a student’s interaction with the system. Also one slide has active hyperlinks to some actual web pages from whence quizzes can be downloaded.

```

\theAnswersStream
%
\multicolumn{2}{1}{\Ans0 none of these. \hfil}
\end{answers}
%
\begin{solution}
\medskip

\def\setupvalues{%
\def\fracsuma{1}\def\fracsumb{2}\def\fracsumc{5}
\def\fracsumd{9}\def\fracsump{9}\def\fracsumq{2}
\def\fracsumr{18}\def\fracsums{19}
\def\fracsumsign{+}
\def\ok{\frac{\fracsums}{\fracsumr}}
\def\wronga{3}\def\wrongb{\frac{6}{11}}

\addtocounter{solutionno}{1}

\IfFileExists{\defsdirefs\thesolutionno}
{\input{\defsdirefs\thesolutionno}}{\setupvalues}

Using the common denominator \fracsumr
$$\frac{\fracsuma}{\fracsumb}
\fracsumsign \frac{\fracsumc}{\fracsumd}
=\frac{\fracsuma\times\fracsump}{\fracsumr}
\fracsumsign

```

```

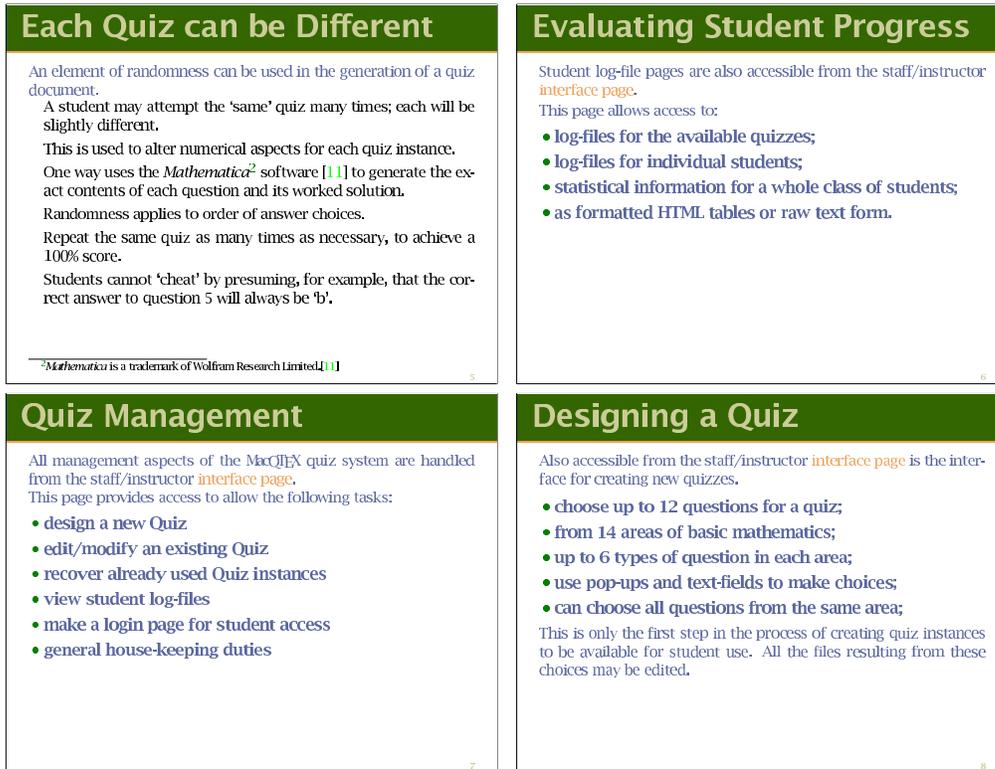
\frac{\fracsumc\times\fracsumq}{\fracsumr}
=\ok.
$$
\medskip

\CheckifGiven
\end{solution}
\medskip

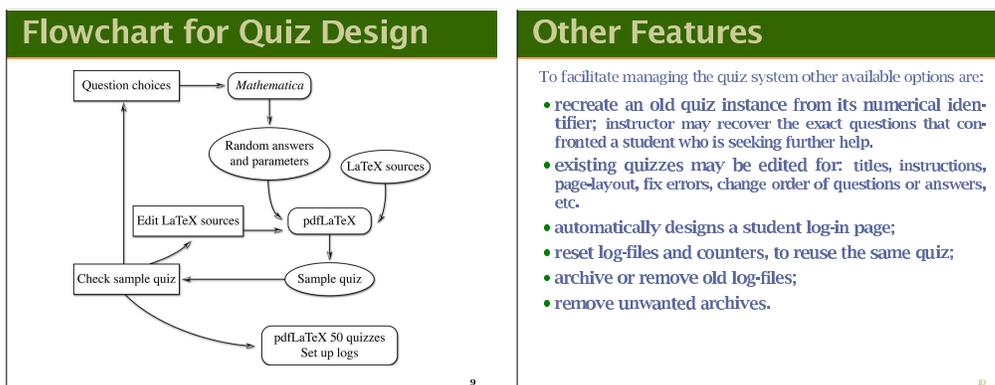
```

In the above TeX coding, the role of `\setupvalues` is to provide default definitions, just in case there is no file `defs\thequestionno` (that is `defs1`, `defs2`, ... according to the question number) within the `\defsdirefs` directory on the local file-system. If it does exist, (e.g., having been written by *Mathematica*<sup>4</sup>, after performing calculations employing a random-number seed) then it should contain the necessary `\definitions`. Similarly there can be a file `\answerdir answers\thequestionno` to govern

<sup>4</sup> *Mathematica* is a trademark of Wolfram Research Limited.[13]



**Figure 14:** These presentation slides indicate how a random aspect is incorporated into the quizzes. Also shown are other aspects of generating quizzes, obtaining statistical information from the log-files, and general house-keeping duties that can be performed.



**Figure 15:** One slide here gives a rough view of the work-flow involved when preparing a set of quizzes for student use. A more detailed work-flow is given in figure 11.

the order in which the answers are presented in the quiz question.

## References

- [1] Adobe Systems Inc.; “Acrobat Forms JavaScript Object Specification, Version 4.0”; Technical Note #5186; Revised: January 27, 1999.
- [2] Adobe Systems Inc.; Acrobat Reader, viewer for PDF format [4] documents, available free of charge from <http://www.adobe.com/>.
- [3] Adobe Systems Inc.; “PDF Toolkit Overview”; Technical Note #5194; Revised: August 10, 1999.
- [4] Adobe Systems Inc.; “Portable Document Format, Reference Manual, Version 1.3”; March 11, 1999.
- [5] Adobe Systems Inc.; “pdfmark Reference Manual”; Technical Note #5150; Adobe Developer Relations; Revised: March 4, 1999.
- [6] Hàn, Thế Thành; pdf $\TeX$ , free software for generating documents in PDF format, based on the  $\TeX$  typesetting system. Available for all computing platforms; see <http://www.tug.org/applications/pdftex/>.
- [7] Lamport, Leslie; L $\TeX$ , a Document Preparation System. This is free software available for all computing platforms. Consult the  $\TeX$  Users Group (TUG) web site, at <http://www.tug.org/>.
- [8] McKay, Wendy and Moore, Ross; “PDF presentations using the Marslide package.”  $\TeX$  Users Group 2001 Proceedings, (elsewhere in this volume).
- [9] Netscape Communications Corporation; Netscape JavaScript Reference, 1997; online at: <http://developer.netscape.com/docs/manuals/communicator/jsref/toc.htm>.
- [10] Story, Donald; exerquiz & Acro $\TeX$ , packages for including special effects in PDF documents, using  $\TeX$  and L $\TeX$ . Dept. of Mathematics and Computer Science, University of Akron. Software available online from <http://www.math.uakron.edu/~dpstory/webeq.html>.
- [10] Story, Donald; “Techniques of Introducing Document-level JavaScript into a PDF file from a L $\TeX$  source.”  $\TeX$  Users Group 2001 Proceedings, (elsewhere in this volume).
- [12] Wall, Larry; *Perl*, a general purpose scripting language for all computing platforms. This is Free Software, available from <http://www.perl.com/>.
- [13] Wolfram Research Inc; *Mathematica*, a system for doing Mathematics by computer. Consult the web site at <http://www.wri.com/>.

# Using pdf $\TeX$ in a PDF-based Imposition Program

Martin Schröder

ArtCom GmbH, Grazer Strae 8, 28359 Bremen, Germany

[ms@artcom-gmbh.de](mailto:ms@artcom-gmbh.de)

<http://www.oneiros.de>

## Abstract

pdf $\TeX$  has been successfully used to build an industrial-strength, PDF-based, imposition program.

We give an introduction to imposition, and describe the system and our reasons for using pdf $\TeX$ . The system stores its information in an internal format that is translated into  $\LaTeX$ . This file is run through pdf $\TeX$  which collects the images and puts them all on a single page. We describe the transformation and our experiences in setting up a  $\TeX$ -installation and present our enhancements and corrections to pdf $\TeX$ .

# The PDF Panel

Nelson H. F. Beebe

Center for Scientific Computing

University of Utah

Department of Mathematics, 322 INSCC

155 S 1400 E RM 233

Salt Lake City, UT 84112-0090

USA

Email: [beebe@math.utah.edu](mailto:beebe@math.utah.edu), [beebe@acm.org](mailto:beebe@acm.org), [beebe@computer.org](mailto:beebe@computer.org), [beebe@ieee.org](mailto:beebe@ieee.org) (Internet)

URL: <http://www.math.utah.edu/~beebe>

Telephone: +1 801 581 5254

FAX: +1 801 585 1640, +1 801 581 4148

## Introduction

The TUG'2001 Portable Document Format (PDF) Panel convened on Tuesday, August 14, 2001, with members Nelson H. F. Beebe, Hans Hagen (chair), Martin Schröder, Don Story, and Hàn Thế Thành, with many comments and questions from the audience.

The list of topics that was projected on the screen makes up the sectional headings in what follows.

Any errors or omissions in this article are solely the fault of this reporter. The text is based on cryptic notes that I typed into a computer file immediately after the panel session, but has been expanded into English, with literature and Web references.

## PDF design goals

Adobe Systems defined the Portable Document Format (PDF) in a specification published in 1993 [7], and updated in 2000 [3] and 2001 [4]. Adobe maintains a developer's Web site for PDF at <http://partners.adobe.com/asn/developer/technotes/acrobatpdf.html> where technical notes and electronic versions of the specifications may be found.

The PDF language shares much of the syntax of PostScript, but unlike PostScript, is not a true programming language. Several technological goals influenced the development of PDF:

- The data format must be identical on all operating system platforms.
- The data format must be public.
- A compact page description is desirable, to reduce data storage and transfer costs. While later versions of PostScript defined additional compression algorithms, their use has primarily been for compressing bitmaps, and program-

mers have to work quite hard to produce compact PostScript. [Tom Rokicki's *dvips* and my *dviaw* have both taken this about as far as is feasible: both produce much more compact output than the majority of PostScript-producing packages.]

- Page independence is needed to support high-speed parallel printing, to speed up screen display, and to permit network transmission of document fragments.
- Random access to individual page descriptions is needed to speed processing.
- Single-pass file generation is required.
- It should be possible to incrementally update page description files, while retaining the complete original contents. Examples include addition of reader comments in overlay notes, replacement of low-resolution figures by high-resolution ones, and repair of minor typographical errors.
- A simplified, nonextensible, page-description language permits simpler and faster implementations of page rendering software.
- Font embedding promotes document portability.
- Font subsetting may reduce file size, and, for some vendors, solves the font copyright issue. For more information about this, see the *TEX Font Panel* article in these proceedings.
- The format should be extensible, so that new object types (e.g., audio and video) can be added in the future, without invalidating software that recognizes only older formats.
- It should be possible to encrypt file contents, and prevent (or at least, strongly discourage) certain actions, such as data extraction and printing.

These goals have all been met in Adobe's implementations of PDF processing software.

Interestingly,  $\TeX$  DVI files, defined fifteen years earlier, fulfill all of these requirements, except for font embedding, encryption, and incremental updating.

### PDF advantages

Publishers and print shops like PDF, because such files are less troublesome to deal with than PostScript files often are. Numerous magazines and newspapers are now printed locally from master PDF files shipped electronically, saving the significant expense and delay of long-distance transportation of printed matter.

Some printer vendors exploit page independence to achieve very high performance: IBM has a PDF printer with 24 CPUs simultaneously rendering PDF page images to print at more than 400 pages/minute.

At least one PDF file viewer is freely available for each of the major platforms, including a hand-held Personal Digital Assistant (PDA), so the vast majority of computer users can view PDF files without cost. Besides Adobe's free Acrobat Reader, and their commercial Capture, Catalog, Distiller, InDesign, Photoshop, and Illustrator tools, there are Ghostscript (<http://sourceforge.net/projects/ghostscript/>), Ghostview, gv, and xpdf (<http://www.foolabs.com/xpdf/>) for viewing and printing, pdf2ps for printing, pdftotext for extracting raw text, and Ghostscript's ps2pdf and Frank Siegert's PStill (<http://www.wizards.de/~frank/pstill.html>) for converting PostScript to PDF.

The availability of multiple independent implementations is critical for demonstrating the sufficiency of the published PDF specification. It also promotes market competition, and gives users alternatives when the inevitable nasty software bug arises.

Apple's MacOS X operating system uses PDF as the native screen description format. There were early attempts to use PostScript for that purpose by Sun, with the *Network extensible Window System*, NeWS [8], in the late 1980s, and by NeXT, with Display PostScript [2, 9], in the early 1990s. Regrettably, processing power at the time was insufficient to make those efforts successful.

Adobe developed a special simplified generic PDF-producing printer driver, PDFWriter, for Microsoft Windows and Apple MacOS. This has made it possible for software vendors on those platforms to add PDF output capability with relatively little

effort. Regrettably, output quality is sometimes inferior to what Distiller can produce, leading to user confusion and dissatisfaction. Adobe Acrobat 5 now installs a PostScript driver instead of PDFWriter.

PDF supports the notion of 'thumbnails': small bitmap images of pages that can be quite helpful in navigating through those documents where pages have recognizably different appearance. It also has bookmarks and hypertext links.

PDF viewers also offer magnification, which can be quite helpful in overcoming low screen resolution, or compensating for vision impairment.

Newer PDF viewers provide for page rotation, which is essential for reading documents with tables in landscape orientation.

Adobe offers a free PDF file creation service on the Web at <https://createpdf.adobe.com> that can be used to convert files from a variety of current desktop publishing and bitmap graphics file formats to PDF.

PDF has been extended to handle *forms*: documents with boxes to be filled out and transmitted electronically. The U.S. Internal Revenue Service provides income tax forms this way.

### PDF and $\TeX$

Hàn Th  Thành's important Ph.D. thesis research that led to pdf $\TeX$  has shown how  $\TeX$  users can directly enjoy the benefits of PDF. The close coupling between typesetter and device driver makes some things possible that would perhaps be impractical in the conventional  $\TeX \rightarrow \text{DVI} \rightarrow \text{PostScript} \rightarrow \text{PDF}$  production path.

Elsewhere in these proceedings, Don Story shows how JavaScript can be used with  $\TeX$  and PDF to create interactive documents, and Hans Hagen's fine work with Con $\TeX$ t and PDF is almost magical.

The hyperref package, written by Sebastian Rahtz, Heiko Oberdiek, and others, modifies  $\LaTeX$  sectional and cross-referencing commands to emit  $\TeX$  `\special` commands to record hypertext links that some DVIware, and pdf $\TeX$ , can deal with. PDF supports such links, so PDF file viewing is automatically enhanced with navigational links. The package is available in the CTAN archives at <ftp://ctan.tug.org/tex-archive/macros/latex/contrib/supported/hyperref/>.

### PDF and document archiving

In my view, the open specification and wide acceptance of PDF is very likely to ensure that it can be used for 'long-term' document storage, something

that cannot be said for *any* of the proprietary desktop publishing formats.

Nevertheless, because PDF is a page description language, rather than a document markup language, it is still best to preserve document input forms, provided those are open and, possibly de-facto, standard.

### PDF disadvantages: availability

Despite the praise of the previous sections, PDF is imperfect.

PDF implementations do not always agree with the specification, and Adobe's software often precedes the specification by months, or even a few years, as happened with PostScript Level 3. Third-party software developers then face the Herculean task of trying to reverse engineer the specification from experiments with Adobe's software. The development of both Ghostscript and pdfTeX has been significantly delayed by such problems.

Adobe's initial support of PDF for Apple MacOS, IBM PC DOS and OS/2, Microsoft Windows, and several flavors of UNIX (Compaq/DEC OSF/1, GNU/Linux on Intel x86, Hewlett-Packard HP-UX, IBM AIX, and Sun SunOS and Solaris) was encouraging. After all, a file format can hardly have the term 'Portable' in its name if it is not usable almost everywhere.

Sadly, Adobe's original commitment to broad support of PDF has been sharply curtailed. While the free Acrobat Reader component is offered for a number of platforms and human languages (see <http://www.adobe.com/products/acrobat/alternate.html>), the Acrobat product family with Distiller and Exchange has been completely dropped on all but MacOS and Windows. This is extremely troublesome, when Adobe markets PDF as a ubiquitous solution for page description.

The Acrobat releases for UNIX systems have an un-UNIX like command line, and lack support for path searching to find needed files. They also ship without any manual pages, a deficiency that I remedied locally. I donated my work back to Adobe for free and unfettered future distribution. Since the UNIX product line was dropped, that did not happen, so I am willing to make that documentation available on request to licensees of the product. For copyright reasons, I cannot place it in a public archive.

Were it not for Aladdin Ghostscript, users on other platforms would be mostly unable to produce PDF files at all. While the *Aladdin Free Public License* is quite generous, it does restrict commercial re-use, which, among other things, means that

Aladdin Ghostscript cannot be included on TUG's annual TeX Live CD-ROM. Instead, TeX Live has to use the approximately one-to-two-years-older GNU release of Ghostscript.

It is never a good idea to rely on *any* software product that has a sole implementation, or runs only on a single platform. Software is complex, and even the yet-to-be-written perfect software package can be crippled by errors in the compiler, or runtime libraries, or operating system, or even hardware. Scientific experiments are never considered reliable until they have been independently reproduced. Software use is, after all, just another kind of experiment, and experience should have taught us to be highly skeptical of the outcome of any change to input data, or to program code.

Thanks to the fine work of Karel Skoupy and the NTS team [16], even TeX now has an independent implementation, although METAFONT still does not.

### PDF disadvantages: complexity

PDF is compact because of data compression, and use of a binary, rather than ASCII, representation. Although the latter is possible, and was originally touted as an advantage of PDF [7], in practice, binary encoding is now almost universally used.

Compression and binary encoding both introduce a serious problem: data transformations that were formerly simple in uncompressed plain text now become immensely more complicated. A great many of the problems posted to the PDF user and developer mailing lists would have relatively simple solutions with plain text files.

What is needed is a standard tool for dumping PDF into a text format that can be edited, then converted back to the binary form, much as Geoffrey Tobin's extremely useful dv2dt and dt2dv tools (<ftp://ctan.tug.org/tex-archive/dviware/dt1>) do for DVI files, and Lee Hetherington's and Eddie Kohler's t1disasm and t1asm utilities (<ftp://ctan.tug.org/tex-archive/fonts/utilities/tlutils>) do for Type 1 outline font files. To my knowledge, no such freely-distributable tool exists for PDF files.

PDF's numbered, rather than named, object structure means that modifications generally require complete parsing of PDF, because objects must be renumbered if any are added or removed. Any future PDF disassembler/assembler tool must take this into account: it should be possible to hide this design blemish entirely.

The PDF file structure makes it impossible to simply concatenate multiple PDF documents to obtain a single document, something that is generally problem free with PostScript files.

Until I wrote this article, I knew of no generally-available free software that can combine PDF files, although there are commercial products for desktop systems that do so.

Now, with the T<sub>E</sub>X Live distribution,<sup>1</sup> it is as simple as this:

```
texexec --pdfarrange *.pdf --result=all
```

The resulting `all.pdf` file will contain all of the PDF files listed on the command line.

The binary format is also a serious problem for indexing of document collections, such as by Web search engines, or search tools like `glimpse` (see <http://webglimpse.net/>) or `mg` [19]. All of these need a PDF disassembler. Adobe's Acrobat Catalog product for indexing PDF file collections is platform specific, and GUI based, making it useless for many applications.

Except for `dvipdfm` (<ftp://ctan.tug.org/tex-archive/dviware/dvipdfm/>), T<sub>E</sub>X DVI drivers are incapable of dealing with PDF. `pdfTEX` can import PDF figures, but it cannot handle PostScript figures, or support the wizardry of the `pstricks` package (<ftp://ctan.tug.org/tex-archive/graphics/pstricks/>).

No PDF viewers provide information about the properties (font, color, texture, metric, ...) of user-selected displayed text.

### PDF disadvantages: no logical markup

The PDF format lacks begin/end markers for identifying words, lines, paragraphs, sections, ... This is a serious design flaw that T<sub>E</sub>X DVI and PostScript also share. UNIX `troff` at least outputs word and line markers. The reason that these boundaries are important is that some operations can reliably only be done on the formatted text, that is, the text that actually appears on the page image. Such operations include text extraction, cataloging and indexing, spell checking, grammar checking, and string searching. Attempting to do so on the input files is problematic: it is unreliable in the presence of macro expansion (such as in T<sub>E</sub>X files), and the job must be done differently for each possible document input format. It would be far better to perform these actions on the final typeset text in PDF form,

<sup>1</sup> The T<sub>E</sub>X Live CD-ROM lacked space to include precompiled formats for ConT<sub>E</sub>Xt, so you first have to build them, like this:

```
env TEXMFINI ..some-path../texmf/web2c: \  
texexec --make en nl metafun
```

so that the programming job could be done just *once* for *all* input formats.<sup>2</sup>

The begin/end marker lack is just a special case of a more general problem: *all* current page description languages (DVI, PCL, PDF, PostScript, ...) completely lose all logical markup that was present in the input. The PDF discussion lists again provide ample evidence that what users really need is a page description language in which all logical markup is *preserved*, allowing recovery of the input and reliable translation into *any* markup system. It is simply not the case that one can always go back to the original document: often, that document is no longer available, or is in a proprietary format that is no longer available or supported, or is not usable on the current platform.

The recent PDF version 1.3 [3, Section 8.4.3] has some logical structure facilities, and PDF version 1.4 [4] introduces the notion of 'Tagged PDF'. These may supply the needed features to preserve logical markup. One reviewer, however, expressed reservations at their complexity, and it remains to be seen whether PDF-producing applications will take advantage of them.

### PDF disadvantages: design limitations

Cut-and-paste with Acrobat Reader is deficient: ligatures (fi, fl, ffi, ffl, ...) are lost, or corrupted, on every MacOS, UNIX, and Windows platform that I've used. `xpdf` does *not* have this problem.

While PDF viewers offer page selection for printing, only the now-dropped Acrobat Exchange viewer had the ability to clip out a rectangular region of a page and save it as a separate file, with the 'supercrop' toolbar item. That feature is poorly documented, hard to use, imposes an obnoxious minimum crop size, and requires installation of an additional plugin software component. Borrowing figures and text snippets from other documents is a common need in document preparation, so perhaps it is fear of copyright violation that discourages software developers from including the capability in PDF viewers.

Although the Acrobat product family is released in numbered versions for multiple platforms, the viewer features differ between platforms. For example, Acrobat Reader 4 on Apple MacOS has a page cropping feature that is absent from the same version on Microsoft Windows, and the toolbar and menus differ between the two versions. While the

<sup>2</sup> The `dvispell` utility, announced by Daniel Taupin on the `tex-euro` mailing list on 29-Oct-2001, does something similar: it reconstructs text to be spell checked directly from the T<sub>E</sub>X DVI file.

differences are not major, they still require a certain amount of mental retooling for the human user.

In my view, such differences are simply poor software design and management. The window system interface, while platform-dependent, should be a relatively small portion of the PDF viewer code, most of which has the much more difficult task of dealing with complex PDF and font file formats. For example, in xpdf version 0.92, less than 10% of the code deals with the window system (as evidenced by inclusion of window-system-related header files), out of a total of 175,000 lines of C++ code (about nine times as much as either T<sub>E</sub>X or METAFONT have).

The color matching problem is still not satisfactorily solved, although other page description formats have the same problem. We have no technological way yet to guarantee that colors that the author used are very close to what the remote reader or printer gets.

Text searching, and page changing, in all current PDF viewers are vastly slower than those operations in a good text editor on a similarly-sized body of text on the same platform, and regular-expression pattern matching searches are unavailable. PDF viewer startup times are also far too long. Sometimes, performance gets worse instead of better: version 0.91 of xpdf introduced a much more powerful font rendering engine that has dramatically slowed that viewer. A test of viewing each page of this document showed that the new version runs two to fourteen times slower, depending on whether the file server and display are local or remote. Fortunately, the new rendering can be turned off with a command-line option, restoring performance to about the same as that of Acrobat Reader and gv.

The original PDF specification anointed 14 fonts as standard, requiring them to be supported by all viewers, and therefore, eliminating the need to store them in PDF files. When fonts are omitted from PDF files, their metrics are still stored, so that when the required font cannot be found, PDF viewers can substitute other fonts and obtain correct letter spacing, even though the letter shapes are wrong.

With the release of Acrobat version 4, the standard font set was abandoned, and some viewers changed their default fonts, so that displayed documents now look different. Unfortunately, Distiller does not give the user sufficient control to ensure that all fonts will be embedded or subsetted, so users may not be able to ensure the same appearance everywhere for their PDF files. This particular flaw has caused users of the U.S. National Science Foundation FastLane grant proposal process a huge

amount of grief, since PDF files that do not include full embedding and subsetting are rejected.

PDF version 1.4 [4] introduced a transparency feature, something that is completely absent from the PostScript imaging model of opaque paint. It is uncertain how such documents will be converted back to PostScript. So far, the transparency feature is little used, because most software cannot yet produce it. Its omission from PostScript, along with support for 3-D coordinates (and 4-D homogeneous coordinates), are the major flaws in that language that prevent PostScript from serving as a universal output format for modern computer graphics.

### PDF disadvantages: bugs

In order to simplify, or compress, complex PostScript files that use language features (see [1, Appendix H.2.4]) that prevent their inclusion in other documents as Encapsulated PostScript figures, it can be helpful to convert such files to PDF, and then back to PostScript.

Unfortunately, Distiller has an automatic page rotation feature that is beyond user control, even though there is an option for it. I posted an example to the PDF developers list showing two small PostScript files differing only by a single comment: one was rotated by Distiller, and the other was not. This has to be a bug, and it completely prevents automated PostScript → PDF → PostScript cleanup of collections of figure files. ps2pdf does *not* have this problem.

Several PDF producers incorrectly rename subsetted fonts, causing the UniqueID problem discussed in the *T<sub>E</sub>X Font Panel* article elsewhere in these proceedings.

One audience member reported that the Hewlett-Packard 4550N has problems with some PDF files that other HP printer models with PostScript Level 3 support do not have. This may perhaps be traced to the lag between specification and software: the former should always come first.

Some PDF viewers incorrectly handle fonts with characters in positions 0...31 or 128...159.

Despite the fact that Adobe's own co-founder, and chief architect of PostScript, showed over twenty years ago how to use monitor gray scale for effective display of fonts [17], Acrobat Reader does a completely unacceptable job of displaying PDF files that use bitmap fonts (see <http://www.math.utah.edu/~beebe/fonts/outline-vs-bitmap-fonts.html> for further discussion, and visual comparisons). There is *no excuse* for this! PostScript and PDF are capable of handling several different font formats, and

PDF viewers should be able to perform equally well with all of them, subject to bitmap resolution in the original font data. By contrast, Paul Vojta's `xdvi` (<ftp://ctan.tug.org/tex-archive/dviware/xdvi/>) does an excellent job with bitmap fonts.

When Adobe announced Acrobat version 5.0 in August 2000, there was soon a flurry of correspondence on the PDF user and developer mailing lists about problems with the new release. These appeared serious enough, and were shown to impact PDF files produced from  $\text{T}\text{E}\text{X}$  documents, that we chose not to install the new version on local desktops at my site, even though we are licensed to do so. A year later, these problems are still under investigation, and the available version remains at 5.0. This seems to indicate quality control problems that should not be present in commercial offerings (even though they are rampant in the desktop software industry).

PDF files may contain metadata, such as text annotations, bookmarks, hypertext links, and so on, which must be coded in either Unicode, or in a superset of 7-bit ASCII called PDFDocEncoding, defined in [3, Table D.1, p. 551]. The encoding defines 194 characters, which should be the same in all applications. However, while writing software to convert text using standard  $\text{T}\text{E}\text{X}$  accents to PDFDocEncoding, Hans Hagen discovered platform differences for at least these characters: L, Œ, Š, Ÿ, Ž, Ø, and their lowercase companions. Such deviations are simply inexcusable when the specification is so clear.

### Further reading

The three editions of the PDF Reference Manual [7, 3, 4] define the PDF language.

Thomas Merz's books [11, 10, 12] are a good source of information about PDF, and PDF forms. They are also typographically interesting, having been typeset with colored ink.

There are a few other books on PDF that I've recorded, but not yet seen [5, 6, 13, 14, 15, 18].

Finally, an extensive bibliography of publications about PDF and PostScript is available in the  $\text{T}\text{E}\text{X}$  Users Group bibliography archive at <http://www.math.utah.edu/pub/tex/bib>.

### Acknowledgements

Thanks go to my fellow panel members, my wife Margaret, and L. Peter Deutsch of Aladdin Software, for many helpful comments on, and historical background for, drafts of this article.

### References

- [1] Adobe Systems Incorporated. *PostScript Language Reference Manual*. Addison-Wesley, Reading, MA, USA, second edition, 1990. ISBN 0-201-18127-4. viii + 764 pp. LCCN QA76.73.P67 P67 1990.
- [2] Adobe Systems Incorporated. *Programming the Display PostScript System with X*. Addison-Wesley, Reading, MA, USA, 1993. ISBN 0-201-62203-3. LCCN QA76.73.P67 D57 1993. US\$29.95.
- [3] Adobe Systems Incorporated. *PDF reference: Adobe portable document format, version 1.3*. Addison-Wesley, Reading, MA, USA, second edition, 2000. ISBN 0-201-61588-6. xvi + 679 pp. LCCN QA76.76.T49 P38 2000. US\$49.95. <http://partners.adobe.com/asn/developer/acrosdk/DOCS/PDFRef.pdf>.
- [4] Adobe Systems Incorporated. *PDF reference: Adobe portable document format, version 1.4*. Addison-Wesley, Reading, MA, USA, third edition, 2001. ISBN 0-201-75839-3. US\$54.95.
- [5] Ted Alspach and Jennifer Alspach. *PDF with Acrobat 4*. Peachpit Press, Inc., 1085 Keith Avenue, Berkeley, CA 94708, USA, 1999. ISBN 0-201-35461-6. 240 pp. US\$17.99. <http://www.amazon.com/exec/obidos/ASIN/0201354616/thepdfzone/002-7978061-6024210>.
- [6] Mattias Andersson et al. *PDF Printing and Publishing: the next revolution after Gutenberg*. Micro Publishing Press, Torrance, CA, USA, 1997. ISBN 0-941845-22-2. vii + 198 pp.
- [7] Tim Bienz and Richard Cohn. *Portable Document Format Reference Manual*. Addison-Wesley, Reading, MA, USA, 1993. ISBN 0-201-62628-4. xii + 214 pp. LCCN QA76.9.F5P67 1993. US\$24.95.
- [8] James Gosling, David S. H. Rosenthal, and Michelle Arden. *The NeWS Book: an introduction to the Network/extensible Window System*. Springer-Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 1989. ISBN 0-387-96915-2. vi + 235 pp. LCCN QA76.76.W56 A731 1989.
- [9] David A. Holzgang. *Display PostScript Programming*. Addison-Wesley, Reading, MA, USA, 1990. ISBN 0-201-51814-7. x + 406 pp. LCCN QA76.73.P67 H63 1990.
- [10] Thomas Merz. *PostScript and Acrobat/PDF: applications, troubleshooting, and cross-platform publishing*. Springer-Verlag, Berlin,

- Germany / Heidelberg, Germany / London, UK / etc., 1997. ISBN 3-540-60854-0. xiii + 418 pp. LCCN QA76.73.P67M4713 1997. US\$69.50.
- [11] Thomas Merz. *Mit Acrobat ins World Wide Web: Effiziente Erstellung von PDF-Dateien und ihre Einbindung ins Web*. dpunkt Verlag, Ringstraße 19, 69115 Heidelberg, Germany, 1997. ISBN 3-9804943-1-4. 225 pp. DM 69,00; ATS 504,00; CHF 61,00. Includes CD-ROM. Also available in a revised edition in an English translation [12].
- [12] Thomas Merz. *Web Publishing with Acrobat/PDF*. Springer-Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 1998. ISBN 3-540-63762-1. xi + 234 pp. LCCN TK5105.888.M47 1998. Includes CD-ROM. Revised and extended English translation of the original German edition, [11].
- [13] Bruce (Chauncey Bruce) Page and Diana Holm. *Web publishing with Adobe Acrobat and PDF*. John Wiley and Sons, New York, NY, USA; London, UK; Sydney, Australia, 1996. ISBN 0-471-14948-9. xx + 363 pp. LCCN Z286.E43P34 1996. US\$35.96.
- [14] Frank Romano. *Acrobat PDF and Workflow In-Detail*. P T R Prentice-Hall, Englewood Cliffs, NJ 07632, USA, 2000. ISBN 0-13-088948-2. ca. 522 pp. US\$39.99. [http://www.phptr.com/ptrbooks/ptr\\_0130889482.html](http://www.phptr.com/ptrbooks/ptr_0130889482.html).
- [15] Frank Romano, Melbert B. Cary Jr., Mattias Andersson, William Eisley, Amie Howard, and Mark Witkowski. *PDF Printing and Publishing: The Next Generation After Gutenberg: The definitive guide to creating and using Adobe Acrobat 3.0 files*. Agfa Division, Bayer Corporation, 100 Challenger Road, Ridgefield Park, NJ 07660, 1997. US\$27.95. <http://www.AgfaHome.com/publications/dcp8text.html>; <http://www.amazon.com/exec/obidos/ASIN/0941845222/qid%3D905955098/002-1104103-2057202>.
- [16] Philip Taylor and Jiří Zlatuška. The  $\mathcal{N}\mathcal{T}\mathcal{S}$  project: from conception to birth [Abstract]. *TUGboat*, 21(3):304, September 2000. ISSN 0896-3207.
- [17] J. E. Warnock. The display of characters using gray level sample arrays. *Computer Graphics*, 14(3):302-307, July 1980. CODEN CGRADI. ISSN 0097-8930.
- [18] Mark Witkowski. *The PDF Bible: The Complete Guide to Adobe Acrobat 3.0*. GATF-Press, 200 Deer Run Road, Sewickley, PA 15143-2600, USA. Telephone 412-741-6860, Fax 412-741-2311, Toll Free 1-800-662-3916, 1998. ISBN 0-941845-23-0. 438 pp. <http://www.bookbuyer.com/aisles/titles/834607.htm>; <http://www.gatf.lm.com/98-10.HTM>.
- [19] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishers, San Francisco, CA, USA, second edition, 1999. ISBN 1-55860-570-3. xxxi + 519 pp. LCCN TA1637.W58 1994. US\$54.95. [http://www.mkp.com/books\\_catalog/1-55860-570-3.asp](http://www.mkp.com/books_catalog/1-55860-570-3.asp); <ftp://munnari.oz.au/pub/mg>; <http://www.cs.mu.oz.au/mg/>; [http://www.cs.mu.oz.au/~alistair/arith\\_coder/](http://www.cs.mu.oz.au/~alistair/arith_coder/); <ftp://ftp.math.utah.edu/pub/mg/>; <http://www.math.utah.edu/pub/mg/>; <ftp://ftp.math.utah.edu/pub/mg/mg-1.3x/bibsearch-1.02.tar.gz>; <http://www.math.utah.edu/pub/mg/mg-1.3x/bibsearch-1.02.tar.gz>.

# Adobe ‘Marked Objects’ plugin for WaRMreader

Wendy McKay

Control and Dynamical Systems, CalTech, Pasadena, USA

[wgm@cds.caltech.edu](mailto:wgm@cds.caltech.edu)

<http://www.cds.caltech.edu/~wgm/>

Ross Moore

Mathematics Department, Macquarie University, Sydney

[ross@maths.mq.edu.au](mailto:ross@maths.mq.edu.au)

<http://www.maths.mq.edu.au/~ross/>

Thomas Ruark

Adobe Systems Inc., San Jose

[truark@adobe.com](mailto:truark@adobe.com)

## Abstract

The WaRMreader [4] method of using X<sub>Y</sub>-pic for placing labels over imported graphics was presented at TUG’99\* in Vancouver [3]. Central to this method is the use of a .bb file, which contains information concerning “marked points” within the graphic, as well as specifying the bounding box.

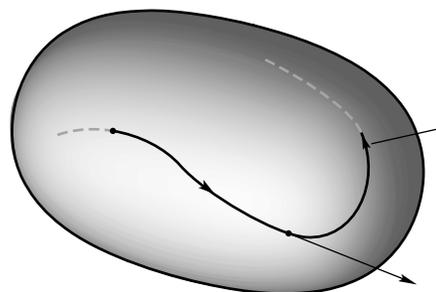
A plugin module has been developed, for use with Adobe’s *Illustrator* [2] (vers. 9, and later), which makes it easy to specify the desired marked points, and store the corresponding information within a .bb file. This information is valuable markup which could be used also for different purposes, with other software packages.

In this talk we demonstrate some possible work-flows for using the new plugin tool. Also, we describe the work done to convince the *Illustrator* Development Team, at Adobe Systems Inc., that such a tool is a simple and useful addition to their software.

## Using the WaRMreader macros

The WaRMreader [4] method is an extension of X<sub>Y</sub>-pic’s `\xyimport` command, that facilitates placement of labels over imported graphics. It works by having knowledge of the location of interesting places in the imported graphic, stored within a separate file in an easily readable text-based format. Since this file must also contain the bounding box information for the graphic, we refer to it as the .bb file, even though filename extensions other than .bb can (and should) be used, according to the format of the information within the file.

Back in 1999, when WaRMreader was presented at TUG’99, there was no convenient way for automatically generating a .bb file from graphics software on most computing platforms. In this paper we describe a new plugin module for Adobe’s *Illustrator* [2] software that provides an intuitive windowed interface for the easy creation of such a



**Figure 1:** Image created with Adobe’s *Illustrator* program, with no visible labels. Such labels will be added later, when the image is included within a L<sup>A</sup>T<sub>E</sub>X document.

file, by inserting ‘marked points’ and other associated information. Before showing how this works, within the *Illustrator* program, we first revisit how the information is used within a (L<sup>A</sup>)T<sub>E</sub>X document

```
%%Creator:Adobe Illustrator 9.0, Macintosh 2000
%%Title: Fig8_2_1.eps
%%Date: 7/6/2001 7:16:6 PM
%%BoundingBox: 0 0 165 108
%%Coordinates: LL
%%StartMarkedPoints
%%MarkedPoint:(144,75) : point(0,0) : M %M, manifold
%%MarkedPoint:(164,62) : point(0,0) : traj %trajectory of\\fluid particle\\moving through\\$$x$ at time $t=t$
%%MarkedPoint:(151,4) : point(0,0) : u %u(\\varphi_t(x),t)
%%MarkedPoint:(105,22) : point(0,0) : phi %\\varphi_t(x)
%%MarkedPoint:(38,61) : point(0,0) : x %x
%%EndMarkedPoints
```

Figure 2: Contents of the file `exempl.bb` defining the ‘marked points’ for `exempl.eps`, as shown in figure 3.

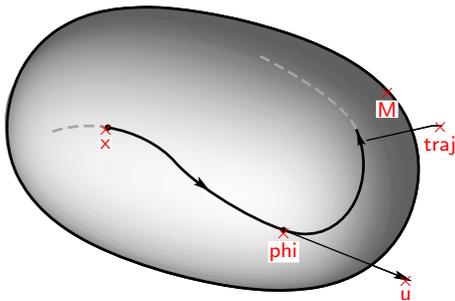


Figure 3: This is the image in figure 1, now showing the locations of ‘marked points’ whose coordinates and ID names have been read from a .bb file (`exempl.bb`).

for placing nicely typeset labels over an imported graphic image. This will help in understanding the nature of the information that needs to be provided within the .bb file, and consequently gives a motivation for the operations that can be performed with the new plugin tool and its associated windows and dialogs.

Figure 1 shows a mathematical diagram created using the *Illustrator* software. The need for explanatory labels is clear, but just which parts of the image should be labelled, and what these labels should say, is dependent upon the context in which the image is to be used. In figure 3 several important features of the diagram are marked with small crosses, each with an identifying string. These crosses and identifying names correspond to the information in the .bb file, listed in figure 2.

A possible set of labels is shown in figure 4, using the X<sub>y</sub>-pic environment and coding shown in figure 5.<sup>2</sup> Essentially the same coding will work in

<sup>2</sup> The 2nd argument to `\xyWARMprocessMoEPS` is ‘pdf’ when the processor is pdf-L<sup>A</sup>T<sub>E</sub>X and `exempl.pdf` has been created from `exempl.eps`. When the processor is L<sup>A</sup>T<sub>E</sub>X+dvips (or other driver) this would be ‘eps’ with the imported graphic named `exempl.eps`.

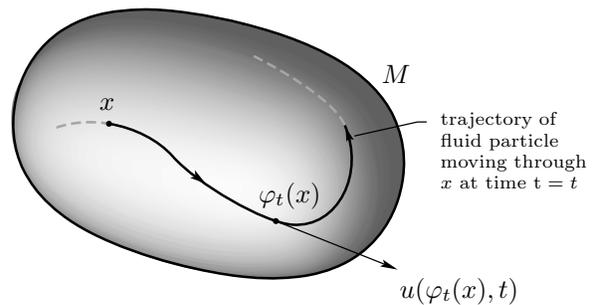


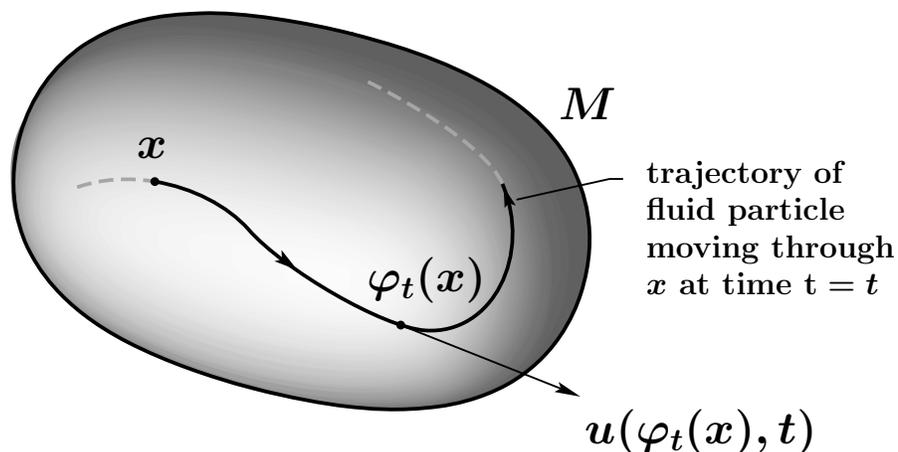
Figure 4: Imported image, fully annotated with meaningful labels, typeset by T<sub>E</sub>X and positioned using the intuitive X<sub>y</sub>-pic notation, as in figure 5.

```
\begin{center}
\def\Fname{exempl}%
\renewcommand{\xyWARMinclude}[1]{%
\includegraphics[scale=.95]{#1}}
\begin{xy}
\xyWARMprocessMoEPS{\Fname}{pdf}
\xyMarkedImport{\Fname}% repeat name for pdfTeX
\xyMarkedPos{x}***!D{x}
\xyMarkedPos{phi}***!D!L(.3){\varphi_t(x)}
\xyMarkedPos{M}***!LD{M}
\xyMarkedPos{u}***!UL{u(\varphi_t(x),t)}
\xyMarkedPos{traj}***!L!U(.5){\parbox{6em}%
{\scriptsize trajectory of\\fluid particle\\
moving through\\$$x$ at time $\mathrm{t}=t$}}
\end{xy}
\end{center}
```

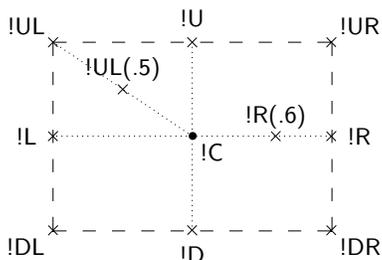
Figure 5: This is the L<sup>A</sup>T<sub>E</sub>X coding for the placement of labels, as in figure 4, over the imported graphic of figure 1.

Plain T<sub>E</sub>X, or other T<sub>E</sub>X format, by using macros `\xy ... \endxy` as delimiters, instead of the L<sup>A</sup>T<sub>E</sub>X `xy` environment. Also another macro for importing the image could replace `\includegraphics`, provided that this macro creates a box of the correct size to exactly contain the image.

Placement of the labels is best done using a `\xyMarkedPos` command, defined in `warmread.sty`,



**Figure 6:** The imported graphic has been resized and the styles used for the labels has been made larger and heavier. Yet the coding to position the labels over the graphic is essentially unchanged.



**Figure 7:** Xy-pic modifiers corresponding to the center (!C), the edges (!L, !R, !U, !D), all 4 corners (!UL, !UR, etc.), and arbitrary locations (!R(.6) etc.), to be the anchor point for positioning of an object over a marked point.

for each marked-point using its identifier. There is one line of coding for each label placed, with the T<sub>E</sub>X coding for the label itself as the {...} on each such line, assuming math mode. Detailed positioning of the labels is achieved using short Xy-pic ‘modifier’ strings. For example, with the u point, the !UL means that the Upper-Left corner of the rectangle containing  $u(\varphi_t(x), t)$  should be at the location of the marked point. In fact the preceding + (within \*\*!UL) has “grown” the rectangle to include a margin of 3pt on all sides; it is the upper-left corner of this enlarged rectangle that is positioned exactly over the marked point. For more details, consult the Xy-pic Reference Manual [5].

Although the WaRMreader [4] macros provide some alternative ways to position labels, the above method using Xy-pic modifiers is the most flexible for ensuring that labels can be easily positioned

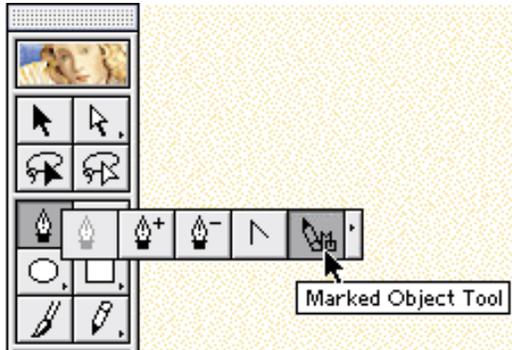
over parts of the image which are both near to the feature being labelled, and are relatively clear of other features prominent within the graphic. For example, the larger image with bold-faced labels, shown in figure 6, uses exactly the same coding as in figure 5 to position all the labels. That heavier style of labelling is more suited for display as a lecture slide, whereas the lighter style of figure 4 is better suited to a journal or book. (Indeed the image in figure 4 was scaled to 95%, so that after adding labels the whole figure would still fit snugly into the 2-column format of this journal.) The difference in coding is simply to precede the xy environment with L<sup>A</sup>T<sub>E</sub>X and Xy-pic style-changing commands, as follows:

```
\def\Fname{exampl}%
\mathversion{bold}\LARGE\bfseries
\renewcommand{\xyWARMinclude}[1]{%
  \includegraphics[scale=1.3]{#1}%
\objectmargin{5pt}%
\begin{xy}
  ...
```

and to change the \scriptsize within the \parbox to \large, for one of the labels.

The primary advantages of using this method to label graphics are as follows.

- The style, size, and font-face of labels placed over imported images necessarily matches that used in the surrounding document.
- Since labels are added within the L<sup>A</sup>T<sub>E</sub>X source, the wording of labels can be changed without need to alter the underlying graphic in any way.
- The same image can be used in many settings, with different sizes, styles or text for the labels;



**Figure 8:** The Pen tool pop-out has a new icon, corresponding to the ‘Marked Objects’ point selector.

there is no need to make any edits within the image file itself.

A subsidiary advantage is that the method of labelling is unaffected by compatibility problems in font technologies within the software used to create the graphic images. For example, there are known problems with the use of Computer Modern fonts in artwork created with earlier versions of *Illustrator*. When such images are viewed in recent versions of Adobe software, either font substitutions occur, or some glyphs may fail to appear at all. With the WaRMreader method such problems become irrelevant, since no text fonts are needed at all within the imported images.

### Adobe ‘Marked Objects’ plugin tool

In response to a request from Wendy and Ross, Tom Ruark has been working on a plugin module for Adobe *Illustrator* which greatly simplifies the task of constructing a `.bb` file while composing artwork using that program. This work is not yet complete, but a beta version of the plugin is available at Wendy’s site.<sup>3</sup> There is one variant for Macintosh and another for Windows-based platforms.

When the plugin is installed (in the sub-folder named `Plug-ins/tools/` within *Illustrator*’s installation folder) an icon for an extra tool (MO-pen) appears within the pop-out ‘Pen Tool’ menu, see figure 8. When selected, the cursor changes; any click within the artwork canvas creates a ‘Marked Object’ within a new non-printing layer. Selecting ‘Show Marked Objects window’ from the ‘windows’ menu, reveals a window that allows properties of marked objects to be seen and edited. Figure 9

shows a view of this dialog, along with the ‘Layers window’ showing the presence of the special layer.

As a new marked point is created, by a mouse-click with the MO-pen tool, it is allocated a default ID which is a count of the number of marked objects that have been created. (This count also includes any that have been subsequently deleted.) Using the Marked Objects window and selecting the list entry for the marked object, this ID can be adjusted to something that is more meaningful within the particular artwork. Also a text-string may be associated with the marked point. This allows a short meaningful description of the place being marked to be included with the artwork. These strings are located in the special layer, so will normally be non-printing, however the layer can be made printing if desired; e.g., for editing purposes. Furthermore, by click and drag with the usual selection tool, these string objects can be shifted around within the artwork for clarity. The font and other attributes of these strings can be adjusted to taste; this includes altering the default position for the location of these strings relative to the marked point. See figure 10 for an example showing the appearance of some artwork, both when the marked objects are hidden and when visible. In future it may become possible to specify also a shape and size for marked objects, other than just points.

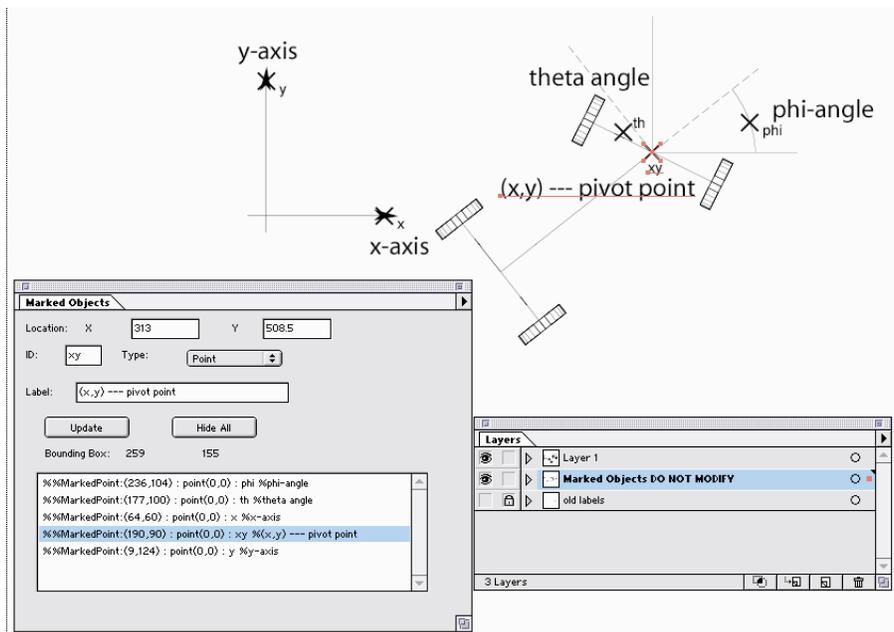
Saving the marked point information in a `.bb` file occurs automatically when the artwork is saved.<sup>4</sup> However a variant of the `.bb` file can be saved at any time, with an arbitrary filename prefix, using a file dialog accessed from the Marked Objects window, as shown in figure 11.

### Getting help from Adobe

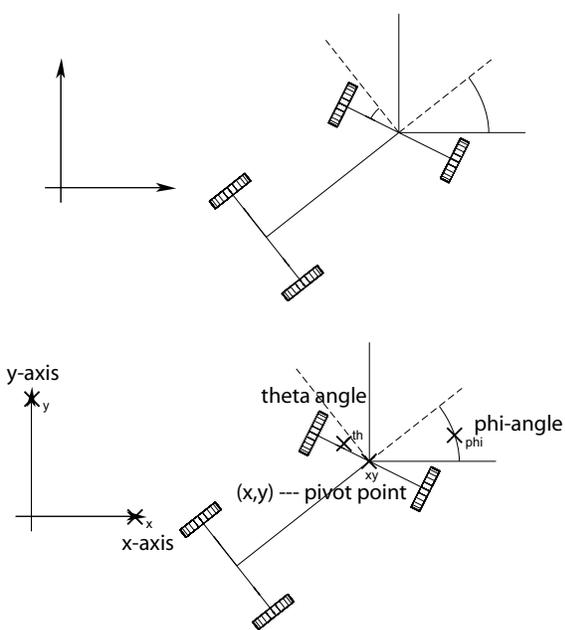
When Wendy first approached Adobe with the proposal for a plugin to create `.bb` files, Tom requested that she provide a work-flow for how the module would be used. Figure 12 shows such a work-flow. As well as this, Wendy and Ross designed a possible interface to the plugin tool, based on extending existing interfaces within *Illustrator*. The realisation was that the kind of coordinate information required in the `.bb` file is similar to what is needed for creating image-maps for graphics on web pages. That is, it was recognised that all the information that might be needed for marked objects should be already available (at least internally) for artwork being edited with *Illustrator*. To emphasise this, the original proposal suggested that the ‘Marked

<sup>4</sup> The current version of the plugin is slightly buggy in that it always saves a `.bb` file, even when there are no marked objects.

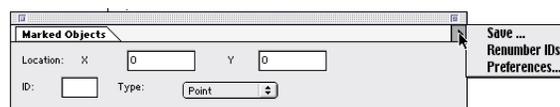
<sup>3</sup> <http://www.cds.caltech.edu/~wgm/WARM/adobe/>



**Figure 9:** The ‘Marked Objects’ window lists all the marked points and allows their locations and other properties to be adjusted. Marked objects are created within a separate layer which should be ‘hidden’ and made non-printing before the artwork is saved into an .eps file.



**Figure 10:** The upper image shows some artwork with the Marked Objects layer hidden; i.e., as it would be seen when imported into a  $\LaTeX$  document. The lower image shows also the marked objects with their IDs and associated strings.



**Figure 11:** Saving a .bb file happens automatically, or can be done at any time from the ‘Marked Objects’ window, using the menu item shown here.

Object’ interface could be developed as extensions to the existing interfaces showing ‘Document Info’ and other ‘Attributes’. These suggestions are illustrated in figures 13 and 14, which accompanied the workflow (in figure 12) for the proposal.

In fact Adobe’s *Illustrator* team of developers decided that a separate interface would be best. Programming this should not present great difficulty, as indeed all the APIs necessary to collect the required information were already available within the existing code-base for the *Illustrator* program. Sure enough, Tom was able to produce the first attempt at a working plugin within a very short time. Subsequent revisions have been to improve the interface and to fix bugs that have emerged during testing.

**PROPOSAL FOR MODIFICATIONS/PLUG-IN TO ILLUSTRATOR 9.0**

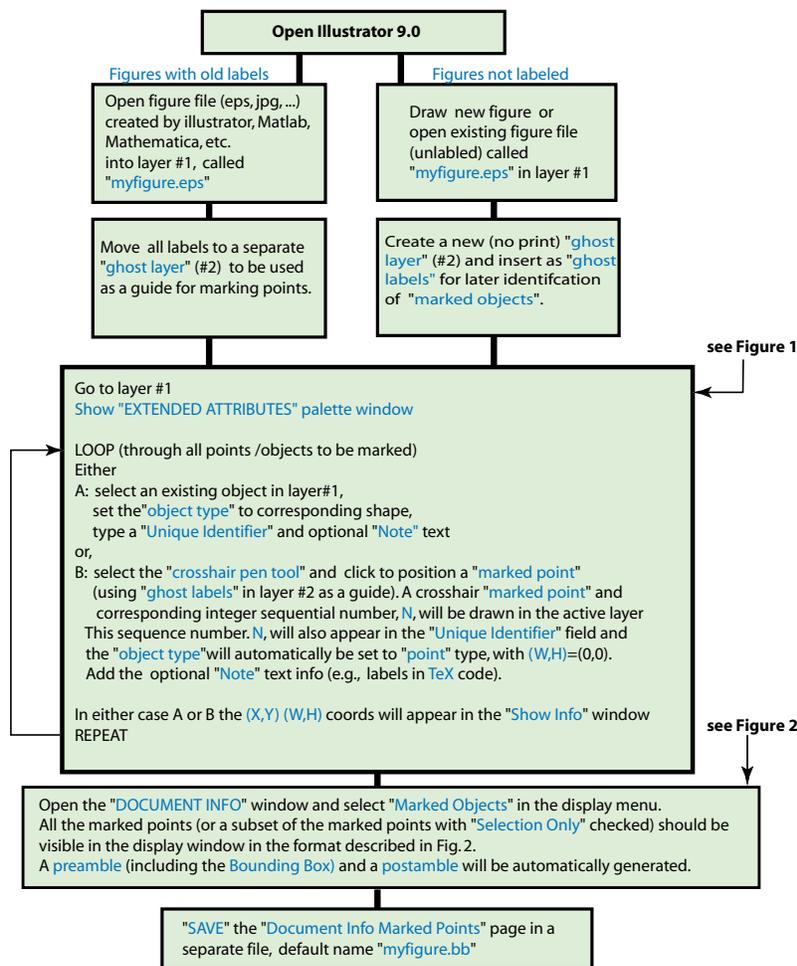
**New features:**

1. "Extended Attributes Window" to include "Marked Objects",
2. "Extended Pen Tool" for displaying Marked Anchor Points, and an
3. "Extended Document Info" to have a "Marked Objects" option added to the menu; to display the "Extended Attributes - Marked Objects" information to selected objects in a specified format; to add some preamble and posamble text with "bounding box" information; to add an option to the "Save" function for this page of displayed information of selected marked objects to be saved as a separate (.bb) file.

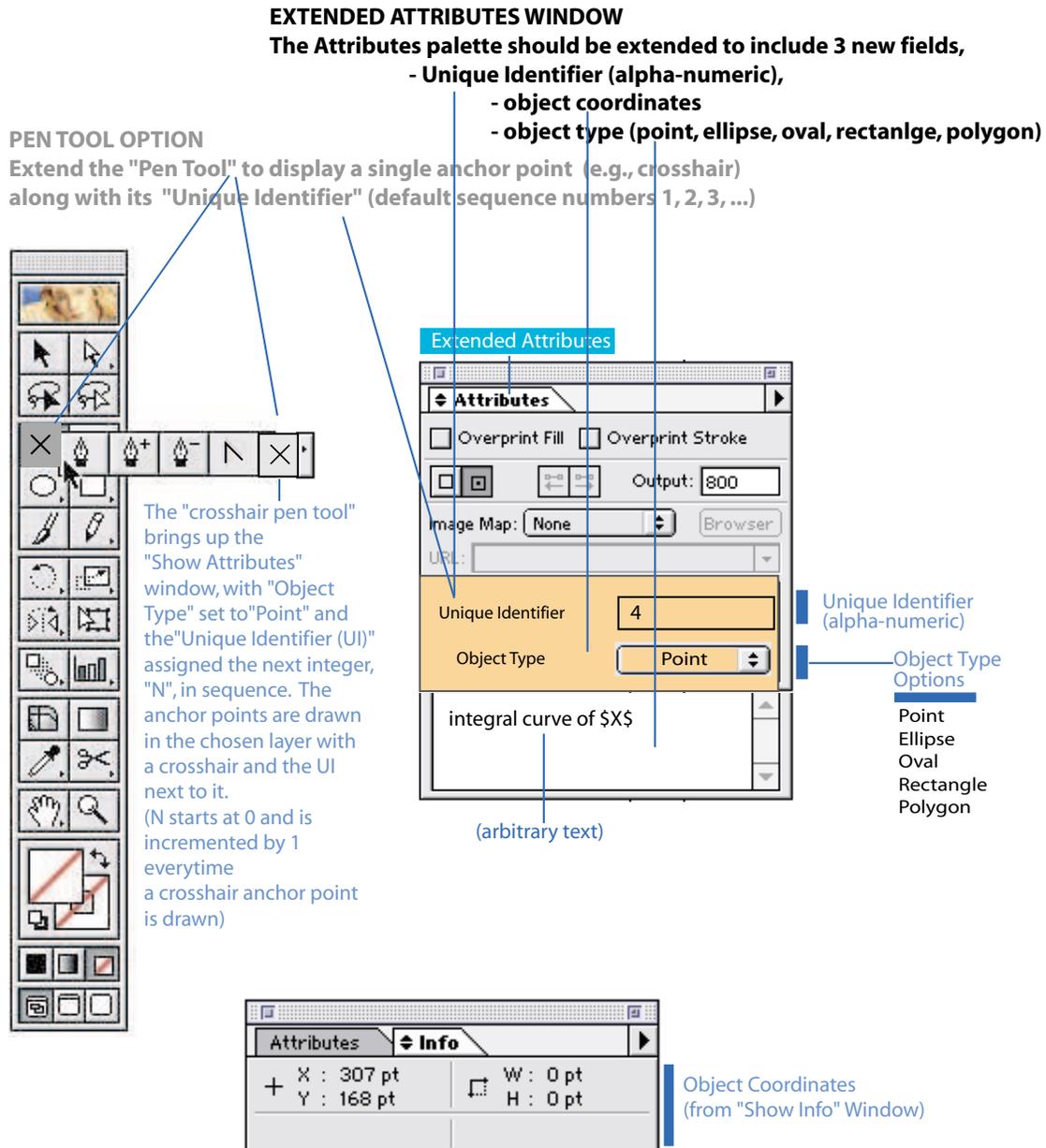
<http://www.cds.caltech.edu/~wgm/WARM/extensions/>

Wendy McKay (wgm@cds.caltech.edu)  
 Ross Moore (ross@cds.caltech.edu)  
 January 6, 2001

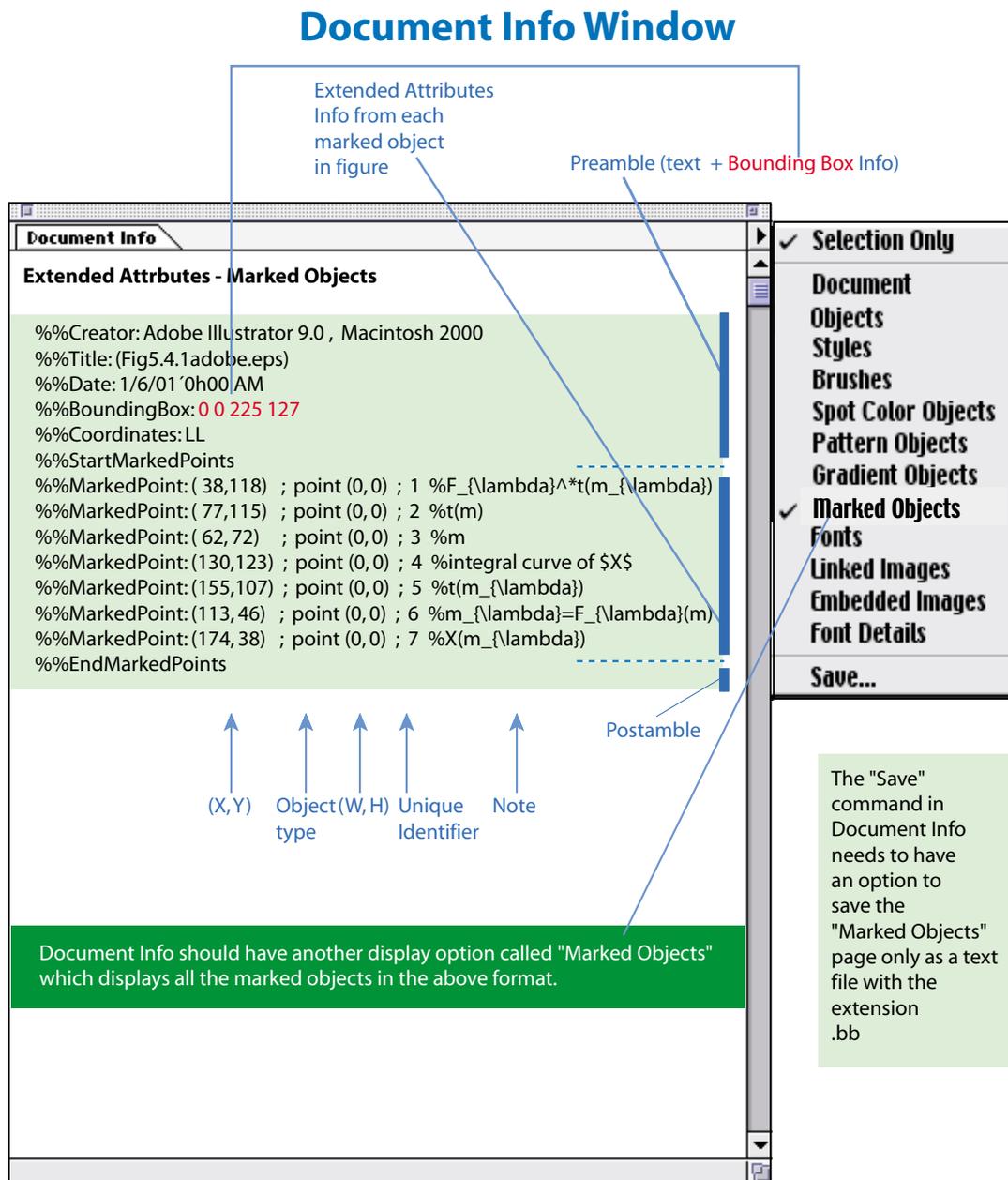
### Extended Attributes & Marked Objects



**Figure 12:** Reproduction of the proposal to produce a plugin module for Adobe's *Illustrator* software. Top: front-page with brief summary. Bottom: workflow, for how the plugin tool might typically be used.



**Figure 13:** Reproduction of Figure 1 from the proposal to produce a plugin module: extensions to the pen-tool and 'Attributes' window. In the actual module the MO-pen icon was redesigned. These extensions to the windows, and that to the 'Document Info' window shown in figure 14, were combined into a single new 'Marked Objects' window; see figure 9.



**Figure 14:** Reproduction of Figure 2 from the proposal to produce a plugin module: proposed extensions to the 'Document Info' window, to display information about marked points. These extensions were not implemented as shown, but were combined with the coordinate data and presented in the 'Marked Objects' window, as shown in figure 9.

### Known bugs and workarounds

As mentioned earlier, the plugin module is still under development, mainly to ensure that the interface is reliable and easy to use. Some ‘bugs’ are known to exist. Mostly these can be worked around, with advance knowledge of what can go wrong.

The following problems have been encountered while working with *Illustrator* and the Marked Objects plugin.

#### bounding-box wrong, by large amount:

This occurs when marking points in graphics originally created using older versions of *Illustrator*. These can have a bounding-box whose bottom-left corner is not at the origin of coordinates (0, 0), which is usual for .eps files created in *Illustrator* version 9.0 and later.

One ‘fix’ is to open the corresponding .eps file in a text editor, copy its %%BoundingBox, then paste this into the .bb file to replace the incorrect information there.

A better ‘fix’ is to create a new document for artwork in *Illustrator* 9, then ‘Select All’ objects in the older artwork. Copy and paste these into the new document. Do all marking of points in the new version only. (Note that it is *not* sufficient to do a ‘Save As...’ version 9.0 update of the older artwork. This will *not* make any difference to its %%BoundingBox, so the problem will persist.)

#### bounding-box wrong, by small amount:

This occurs when there are hidden layers in the artwork, other than the special Marked Objects layer, in which there is an object that lies (at least partially) outside the bounding-box of the unhidden layers. The %%BoundingBox recorded in the .bb file encloses the hidden objects, whereas that for the .eps file only covers objects in the unhidden layers.

The fix is to copy the %%BoundingBox comment from the .eps file into the .bb file, as in the first ‘fix’ above.

#### absurd coordinates for a marked point:

4- or 5-digit numbers, perhaps negative, lying clearly outside the boundary of the artwork, have been observed to occur as coordinates for marked-points. This appears to happen only

with artwork created with earlier versions of *Illustrator*, after an ID or attached string is moved manually.

The above solution of copying all the artwork into a new *Illustrator* 9 document should ensure that this problem does not occur.

#### badly sorted numerical IDs:

This occurs when the default numerical IDs for marked points are never changed. Then the order will be lexicographic; e.g., 1, 10, 11, 12, 2, 3, 4, 5, . . . .

It is better to use alphanumeric identifiers, which have names that are meaningful to the places being marked.

### Acknowledgement

Two of the authors are deeply indebted to Professor Jerrold Marsden for continued support in the development of WaRMreader and other T<sub>E</sub>X-related software projects. The images used as examples for labelling graphics have come from one [1] of the many mathematical texts which Jerry has either written or co-authored.

### References

- [1] Abraham, R., J.E. Marsden and T. Ratiu; “Manifolds, Tensor Analysis and Applications”. Applied Mathematical Sciences Series, Vol. 75; Springer–Verlag, New York, (1988) 654pp.
- [2] Adobe Systems Inc.; Adobe *Illustrator* 9 (latest version 9.02); purchase online from <http://www.adobe.com/products/illustrator/main.html>.
- [3] Moore, Ross and Wendy McKay; “Convenient Labelling of Graphics, the WARMreader Way”. TUGboat Vol. 20 (3), 1999; pp. 262–271.
- [4] Moore, Ross; “What is WaRMreader?”, examples, documentation and macros for `warmread.sty`; available online at <http://www-texdev.mpce.mq.edu.au/WARM/> and <http://cds.caltech.edu/wgm/WARM/reader2001.html>.
- [5] Rose K. and R. Moore; X<sub>Y</sub>-pic Reference Manual; accompanies the X<sub>Y</sub>-pic diagram macros for T<sub>E</sub>X; version 3.7 available from <http://www.ctan.org/ctan/tex-archive/CTAN::/macros/generic/diagrams/xypic/>.

# The T<sub>E</sub>XspecTool for Computer Aided Software Engineering

Stephen E. Oliver

Whiteshell Labs, Pinawa, Manitoba, Canada R0E1A0  
[seolivers@bellsouth.net](mailto:seolivers@bellsouth.net)

## Abstract

This paper reviews the development of the T<sub>E</sub>Xspec tool, which assists in the development and documentation of quality assured software in a regulated environment. The tool can assist in the development of a broad range of software, but targets the development of software that implements mathematical models. The original application relates to the development of models of a repository for Canada's high level nuclear waste, but is not limited to this use. T<sub>E</sub>Xspec is particularly useful when documenting models and associated programs which rely on mathematical notations to communicate the intent of the software.

## Problem Definition

Canada has developed computer programs to model a deep geologic repository for used nuclear fuel [3, 2]. Regulators require that these programs be of demonstrably high quality to support licence applications.

In 1999, the Canadian Standards Association (CSA) adopted standard N286.7 [6] for the development of nuclear safety related computer programs, a scope that includes the AECL models. While the software development process used to date had been considered robust, it required refinement in order to achieve compliance with this standard.

The T<sub>E</sub>Xspec project seeks to address the issue of compliance with CSA N286.7. The tool supports a compliant software development procedure, while imposing a minimum of additional overhead. While optimised to meet requirements associated with the modeling nuclear fuel waste, it is hoped that T<sub>E</sub>Xspec will find more common usage.

Several commercial Computer Aided Software Engineering (CASE) tools will support a robust software development methodology, but none provide support for the mathematical notations that are common in scientific models. T<sub>E</sub>Xspec provides extensive support for this notation.

**Software Development Methodology.** Although Object Oriented (OO) analysis and design is appropriate for documenting many software applications, there are still applications for procedure/flow based software. In particular, some models which are basically linear in structure are best described using structured (non-OO) methodologies.

Many scientific models have, to date, been described using a modified Yourdon/DeMarco

methodology [5, 11]. Although OO methods would perhaps be more appropriate for some models, priority is given to the more common Yourdon/DeMarco analysis methodology. Products associated with this methodology are:

- data flow diagrams (DFDs),
- process descriptions (mini-specs),
- structure charts,
- subprogram design descriptions, and
- data dictionary listings.

DFDs and mini-specs comprise the requirements specification, while structure charts and subprogram design descriptions document the design. Data dictionary listings may be separated into requirements and design, or combined into a single product.

**Requirements Specification.** Figure 1 illustrates the main concepts of data flow diagrams. Diagram 0 shows the input and output 'flows' to/from a single 'process'. This high level abstraction is intended to allow the reader to identify the functions of the complete system. Process 1 is broken into components in Diagram 1. The diagrams may be thought of as a hierarchy, with higher level diagrams having shorter process numbers (i.e., Diagram 1.2.3 is 'higher' than Diagram 1.2.3.4) The numbering convention of the diagrams and the processes allows the decomposition to be clearly seen. Once processes are decomposed to a point where they can be clearly specified in a short textual description, they appear on a DFD with a double circle, as Process 1.4 illustrates. Such 'atomic' processes are associated with a mini-spec, rather than a lower level diagram.

Data flows, like processes, can be broken up into constituent parts on lower diagrams. In figure 1, for example, ‘Implements’ on Diagram 0 becomes ‘Measuring Cup’, ‘Bowl’, ‘Oven’, and ‘Pan’ on Diagram 1. These flows are associated with multiple processes on Diagram 1, so they must be shown individually.

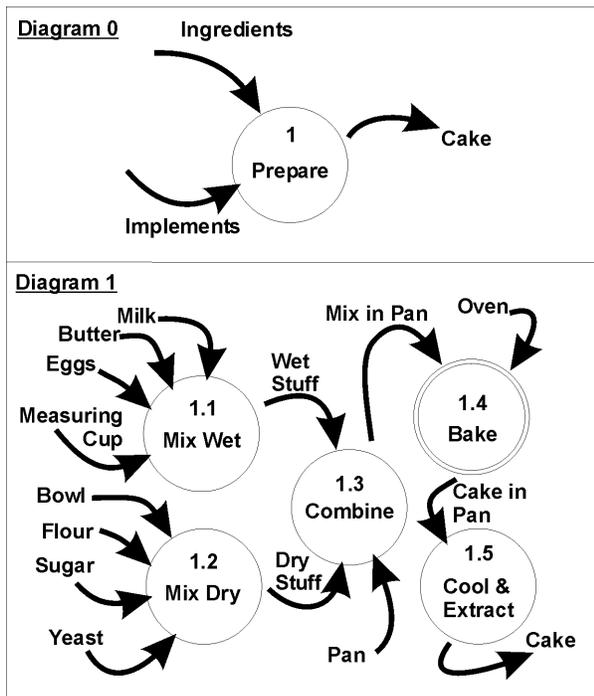


Figure 1: Example Data Flow Diagrams (DFDs)

Mini-specs for each atomic process repeat some of the information on the DFD, and also detail the requirements for a low level process in any manner deemed suitable by the author. For scientific codes, mini-specs often make extensive use of diagrams and mathematical notations.

**Design Specification.** Figure 2 illustrates the main concepts of structure charts. The boxes represent ‘subprograms’ to be composed in a procedural programming language such as FORTRAN. The chart is intended to illustrate the nature of the interface between subprograms. The lines between the subprograms indicate a ‘calling’ relationship, with the subprogram which is closer to the top of the structure chart invoking the lower. The transfer of data at these interfaces is also shown. Data can be passed from one subprogram to another via an argument list (shown along the connecting line), or through common storage that can be accessed from multiple subprograms (shown inside the subprogram box). The interface variables can be input,

or both, as denoted by arrowheads next to the variable name.

Structure charts do not have a hierarchical organization paralleling DFDs. However, a large structure chart may span many pages using off-page connectors.

Each subprogram must itself be documented. Subprogram design descriptions repeat some of the information on the structure chart, and document the algorithm and design details of the subprogram. This may include material common with mini-specs, as the design reflects the requirements. In particular, many of the mathematical equations are referenced in both places.

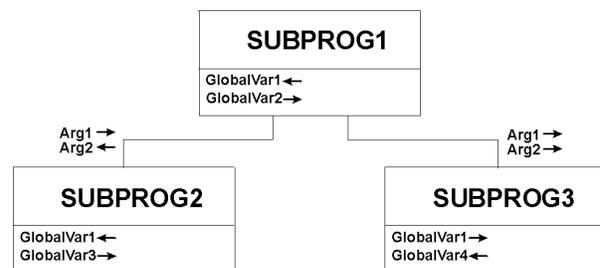


Figure 2: Example Structure Chart

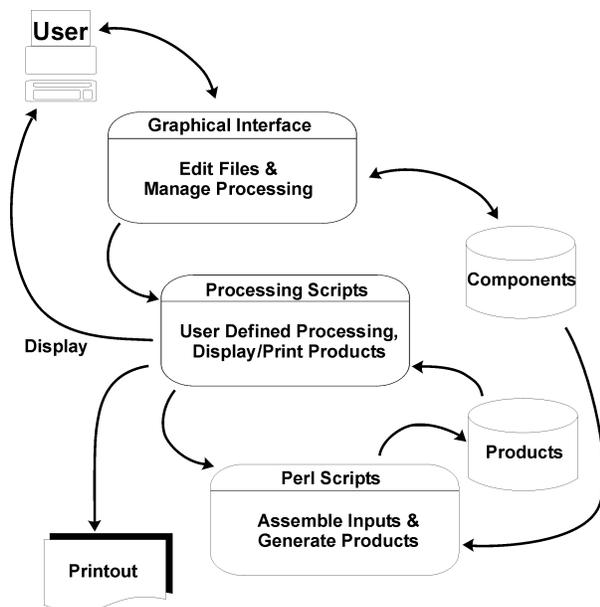
**Consistency Between Products.** Experience at AECL has shown that lack of consistency between software products has been a major source of software defects [7].

Commercial CASE tools have helped to reduce this inconsistency, but these tools all have difficulty in one or more critical areas:

- Lack of support for scientific and mathematical notations. The nature of scientific software demands that mathematical notations ( e.g.,  $A_i(t) = \int_0^t [F_i^{IN}(\tau)]d\tau$  ) be permitted in specifications.
- Insufficient accountability. The principle of ownership and accountability for products is not strictly enforced. While a record of who updated products is often kept, the process control is typically inadequate.
- Assembling large products from smaller components is not adequately supported. Many defects originate as transcription errors between products. Mathematical equations are particularly susceptible.
- Insufficient consistency checking between products.

### The T<sub>E</sub>Xspec Solution

T<sub>E</sub>Xspec takes advantage of the plain text nature of L<sup>A</sup>T<sub>E</sub>X input to permit processing and tracking of shared components. The main T<sub>E</sub>Xspec processing is performed by modules which have been implemented in Perl [10], as indicated in figure 3. A graphical user interface (GUI) captures interactions with the user. Most of this interaction consists of displaying and manipulating ‘component’ files, which form the inputs for the T<sub>E</sub>Xspec scripts that select components and assemble them into products. These products are primarily L<sup>A</sup>T<sub>E</sub>X or Noweb [8] input files, which can be post-processed to produce output suitable for viewing, printing, or compiling.



**Figure 3:** T<sub>E</sub>Xspec Architecture

While the GUI is a convenient way to construct components and initiate processing, it can be bypassed if required. The components can be generated by any means that can generate a plain text output file, including a text editor. More importantly, the processing can be controlled by any means that can initiate a process, with no requirement for interaction with a GUI. When processing many components, or when a log of processing is required, this ‘batch’ style processing is a useful alternative.

Neither the T<sub>E</sub>Xspec scripts, nor the GUI can display or print the products. Figure 3 indicates that an intermediate script, which is intended to be edited by the user, initiates T<sub>E</sub>Xspec to produce the product files, then controls post-processing as

appropriate. This flexibility allows the user to integrate T<sub>E</sub>Xspec into existing procedures. For example, if a static code analyzer such as Floppy [1] is in use, it can be run automatically on code as it is generated. Interaction with a version control system might be desired, or the user may even wish to compile code as it is generated. Alternatively, processing that is not needed can be removed, such as removing documentation generation (including L<sup>A</sup>T<sub>E</sub>X processing) until the code is stable.

In order to support sharing of equations and data definitions, while tracking ownership and responsibility for content, T<sub>E</sub>Xspec supports a fine granularity of components. Each T<sub>E</sub>Xspec component is tracked independently by placing each in a unique file which is mapped by the file name to the name of the component, and by the file name ‘extension’ (in the tradition of MS-DOS or CP/M) to the type of component.

T<sub>E</sub>Xspec components, with associated file name extensions, are:

- Requirements Data Dictionary entries (**.rdd**),
- Design Data Dictionary entries (**.ddd**),
- Equations (**.teq**),
- Data Flow Diagrams (**.dfd**),
- Mini-Specs (**.ms**),
- Structure Charts (**.sc**),
- Subprogram Design Descriptions (**.ds**), and
- Manuals (**.tex**).

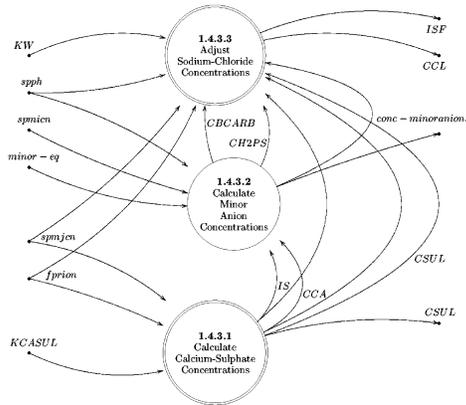
**Data Flow Diagrams.** DFDs such as figure 4 are produced using the xy-pic package [9] under L<sup>A</sup>T<sub>E</sub>X.

T<sub>E</sub>Xspec shows the details of composite data flow decomposition explicitly on the DFD. In figure 4, for example, flow ‘conc-majoranions’ (which would appear on Diagram 1.4.3) is shown as constituent components ‘CCL’ and ‘CSUL’. If a composite flow contains components which do not appear on the current diagram, they are shown in a regular font, while components that do appear on the current diagram are shown in a bold font.

Consistency between DFDs is monitored by T<sub>E</sub>Xspec. A warning messages is generated for any inconsistency between a DFD and it’s parent.

Labels can be shown with mathematical notation, rather than the plain text shown. Switching from plain text to mathematical labels is simple, since flows are taken from the requirements data dictionary (file **name.rdd**), which typically contains both a mathematical and a plain text label. Although this is an interesting capability, there has

INROC-LE		DataFlowDiagram.doc Ver 1.1	
Data Flow Diagram 1.4.3 Determine Speciation of Groundwater	Version 01B	intro	
Author: Ted Melnyk	Feb 22, 2000		
Implementer: Steve Oliver	Sep 29, 2000		
Reviewer:	September 29, 2000		



Implemented by SPCGCN  
 equilibrium-constants = {minor\_eq, KCASUL, KW}  
 sp-ion = {sp-mjcn, sp-mjcn, f-prion}  
 gw-speciation = {conc.anions, ISF}  
 conc-majoranions = {CCL, CSUL}  
 conc-anions = {conc.majoranions, conc.minorations}

Figure 4: T<sub>E</sub>Xspec Data Flow Diagram (DFD)

been little enthusiasm among users to take advantage of it.

The format of the header in figure 4 is common to all T<sub>E</sub>Xspec products, detailing the project, the responsible author, implementer, and reviewer, along with an indication of the genealogy of the product (in very small type), which can be used to trace back the source of any defects.

**Mini-Specs.** Atomic processes are not broken down into lower level DFDs, but are further specified using a mini-spec. This document is intended to be flexible in format, permitting the author freedom to communicate the intent of the process in whatever manner is most effective.

The standard T<sub>E</sub>Xspec header is generated for each mini-spec, as shown in figure 5. The author must explicitly state input and output flows, which are presented in tabular format and verified for consistency with the DFD.

Equations appearing in mini-specs are often referenced in other documents. Authors are encouraged, but not required, to place each equation in a separate file (**name.rdd**), and reference that file from within the mini-spec. The equation can then be reused in subprogram design descriptions or manuals. At AECL, a commercial package is used to create equations that can be saved in L<sup>A</sup>T<sub>E</sub>X format which includes information encoded as L<sup>A</sup>T<sub>E</sub>X comments which permits reuse by word processors.

CC4		MiniSpec.doc Ver 1.1	
Process: 1.1: Determine Container Failures	Version 01A	INROC	
Author: T.E. Melnyk	Feb 22, 2000		
Implementer: S.E. Oliver	Apr 10, 2000		
Reviewer:	March 17, 2001		

Determine the number of containers that have failed at the start of the simulation.

Variable	Symbol	Long Name	Units	I/O
NCONFS.sec	$N_F$	Number of containers failed in a sector		O
IFAILQ.sec	$Q^F$	instant container failure quantile		I
IFRACT	$P_F$	instant failure fraction		I
NCONSC.sec	$N_T$	containers in a sector		I

The failure probability of any individual container  $P_F$  is constant and the same for all containers, so the number of failed containers,  $N_F$ , out of  $N_T$  total containers is determined from the cumulative binomial distribution:

$$\text{If } Q^F \leq P(0; N_T, P_F) \quad N_F = 0 \quad (1)$$

$$\text{Otherwise determine } N_F \in \{1..N_T\} \text{ such that } P(N_F - 1; N_T, P_F) < Q^F \leq P(N_F; N_T, P_F) \quad (2)$$

Where  $P(m; N, p)$  is the cumulative binomial probability distribution. The quantity  $m$  is called the number of "successes" (container failures) from  $N$  trials (total containers), each having probability  $p$  of "success".

$$P(m; N, p) = \sum_{j=0}^m \binom{N}{j} p^j (1-p)^{N-j} \quad (3)$$

This is discussed in the "Container Failures" section of the Inroc Theory Manual.

Figure 5: T<sub>E</sub>Xspec Mini-Spec (MS)

By keeping an equation in a single file, available for reuse, transcription errors are reduced.

**Structure Charts.** T<sub>E</sub>Xspec structure charts such as figure 6 are produced using the xy-pic package [9] under L<sup>A</sup>T<sub>E</sub>X. Subprograms can be grouped using

CC4		StructureChart.doc Ver 1.1	
SIMALL	Version 01A	vault	
Author: S. Oliver	December 17, 2000		
Implementer: S. Oliver	December 17, 2000		
Reviewer: none	NA		

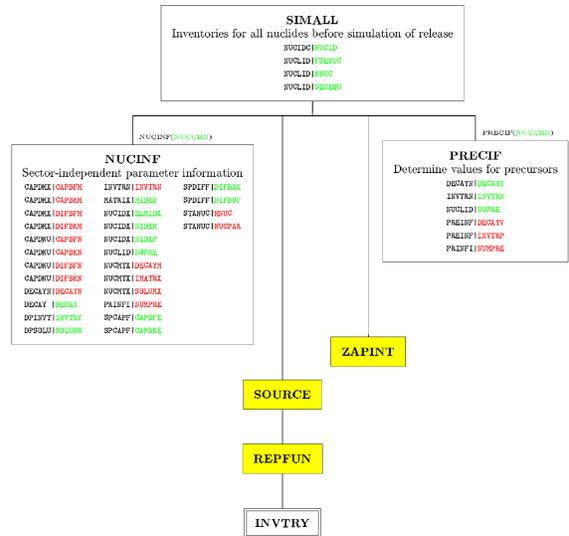


Figure 6: T<sub>E</sub>Xspec Structure Chart

colour coded backgrounds. In figure 6, subprograms

‘SOURCE’, ‘ZAPINT’, and ‘REPFUN’ are grouped with a yellow background indicating that they are library routines, not part of the software being documented.

Input and output variables are also colour coded: green for input, red for output, and blue for both. This applies to both the arguments to a subprogram and to the common storage variables.

FORTTRAN groups common storage variables into named blocks, which are indicated to the left of each variable name. Blocks are sorted alphabetically, and variables are sorted alphabetically within a block.

For subprograms that are functions, rather than subroutines, an additional output variable is provided in the name of the subprogram itself. In this case the name of the subprogram appears in red.

Most of the information on the structure chart is extracted from the subprogram design description (**subprogram.ds**) file for each subprogram on the chart. Presentation details are contained in a structure chart file (**name.sc**). The structure chart file lists the subprograms to be placed at specified locations. It also indicates the calling relationship between the subprograms and the location of the connecting lines. T<sub>E</sub>Xspec generates a warning message if the specified calling relationship is not consistent with the FORTTRAN code.

The description of the subprogram, and the argument list, are extracted from the subprogram design description file, and can be quite long. The structure chart may specify a maximum line length for these elements, and T<sub>E</sub>Xspec inserts line breaks appropriately.

The double box ‘INVTRY’ at the bottom of figure 6 is an off-page connector to another structure chart. This chart is referenced by an off-page connector ‘SIMALL’ on another chart.

**Subprogram Design Descriptions.** T<sub>E</sub>Xspec supports literate programming techniques [4] via use of the Noweb [8] package. Some preprocessing and postprocessing is required to achieve the desired products, but Noweb users will be immediately familiar with the format.

Subprogram design descriptions can be long documents, but an abbreviated product is shown in figure 7. The T<sub>E</sub>Xspec header is followed by a Noweb-style list of code blocks that comprise the subprogram.

Code blocks to declare variables are replaced by a tabular form which contains additional data dictionary information. Of particular interest is

the ‘Symbol’, which is the mathematical notation for a variable. This allows variables to be traced through equations, making the relationship between code and requirements much easier to follow. Also specified is the input/output status of each variable, which T<sub>E</sub>Xspec validates against the contents of the code blocks.

Several tables may be generated. One each for subprogram calling arguments, common storage variables, local storage variables, and constants. After each table, the design may optionally specify preconditions and postconditions for the tabulated variables. Specifying valid ranges for data has proven to make testing much more accurate, and the process of specifying those ranges has identified many defects. Specifying preconditions and postconditions early in the design process is an effective and inexpensive quality control device.

Following the tables are the code blocks, each including commentary in L<sup>A</sup>T<sub>E</sub>X format which may feature equations shared with mini-specs or other documents. Sharing equations ensures consistency. Consistency between subprogram design descriptions and compilable code is guaranteed, since Noweb extracts the code from the subprogram design description. Information for each subprogram on a structure chart is also extracted from the subprogram design description, ensuring that all design documentation is consistent.

**Data Dictionaries.** T<sub>E</sub>Xspec distinguishes between dictionary entries for requirements and design specification. Some design information is never applicable to requirements (e.g., a common storage block name).

It is possible to have a close correlation between entries in the requirements and design dictionaries. Design entries may optionally state a requirements dictionary entry which is related. When this is done, fields in the design dictionary acquire default values equivalent to the requirements data dictionary. This is particularly useful to inherit the mathematical symbol and description.

T<sub>E</sub>Xspec produces a data dictionary listing which can show a cross reference of which products use which dictionary entries.

**Graphical User Interface.** There is a considerable amount of data contained in many plain text files in a typical T<sub>E</sub>Xspec documented project. To assist users, a GUI has been developed as a Java application to act as a front end to the process. While there is little new technology embedded in the GUI, it is interesting to note that the GUI, at over

```

Thu Mar 13 10:29:46 2003
Module: VLGDEP - Determine time-independent vault parameters Version 06E
Author: S.E. Oliver Feb 28, 2001
Implementer: S.E. Oliver Mar 13, 2001
Reviewer: none NA
CC4 INROC
-----
Module components:
(*)=
  (interface)
  (description)
  (directives)
  (include)
  (local)
  (data)
  (main)

Description:
(description)=
Determine time-independent vault parameters that require
parameters determined in QGDDP

Calling interface:
(interface)=
SUBROUTINE VLGDEP()

Shared (COMMON) variables:
-----
| Shared | Long Name | Symbol | Units | Dimension | Data Type | I/O | |
|---|---|---|---|---|---|---|---|
| BKFRAR | frac of vault with backfill | A_F = 2T_F/S | | | scalar | double | I |
| BKPERM | backfill permeability | k_F | [m2] | | scalar | double | I |
| BUFRAR | frac of vault with buffer | A_B = 2T_B/S | | | scalar | double | I |
| CAPDMR | damaged zone capacity fctrs | K_{L,Z} | | | MXCHEM,MAXSEC | double | O |
| CAPRKV | capacity factor at geo-vault interface | K_{L,R} | | | MXCHEM,MAXSEC | double | I |
| : | : | : | : | : | : | : |
| SGTDSF | tran dispersion length factor | f_{s} | | | scalar | double | I |
| THKBAR | thickness of backfill | T_F | [m] | | scalar | double | I |
| THKDAM | thickness of damaged zone | T_Z | [m] | | scalar | double | I |
| TORRKR | tortuosity in bottom geosphere seg | \tau_R | | | MAXSEC | double | I |
-----

Preconditions:
BKFRAR: 0 <= BKFRAR <= 1
BKPERM: > 0
BUFRAR: 0 <= BUFRAR <= 1
CAPRKV: >= 0 for (1, NELMNT, 1, NUMSEC)
:
:

Postconditions:
CAPDMR: > 0 for (1, NELMNT, 1, NUMREG)
CAPRRE: > 0 for (1, NELMNT, 1, NUMREG)
DARBUR: >= 0 for (1, NUMREG)
DARDRA: >= 0 for (1, NUMREG)
:
:

VLGDEP implements Data Flow Diagram process 'Interface with Surrounding Geosphere'.
Additionally, VLGDEP derives parameters for vault regions, based on the properties of the component vault
sectors. The accumulation of multiple vault sectors into a single vault region is a design artifact intended to
improve computational efficiency.

The module consists of two sections
-----
• Evaluate Darcy velocities and dispersion coefficients ('Interface with the Surrounding Geosphere' in the
Theory Manual).
• Evaluate regionalized vault properties.

(main)=
  (geosphere)
  (regional)
  RETURN
  END
-----

Evaluate components of Darcy velocity in rock for one sector (SEC).
The room axis is assumed to be parallel to the X component of the geosphere network cartesian coordinate
system so the axial component is simply V_d^x = V_x.
The transverse groundwater velocity in the rock is correspondingly assumed to be in the YZ plane of the
geosphere network cartesian coordinate system and is evaluated as V_d^t = sqrt(V_y^2 + V_z^2).
Define theta = angle between the axis of the room and the direction of water flow. Compute sin(theta) and cos(theta).
Assume permeability of buffer is zero, and hence, Darcy velocity in buffer is zero V_d^b = 0.
(darcyComponents)=
C.....Compute axial and radial components of Darcy velocity
DARKVA(SEC) = DARKK(SEC)
DARKVR(SEC) = SQRT(DARKRY(SEC)**2 + DARKRZ(SEC)**2)
C.....Evaluate sin and cos of angle between room axis and flow
RKVSIW = DARKVR(SEC) / DARKK(SEC)
RKVCSIW = DARKVA(SEC) / DARKK(SEC)
-----

```

Figure 7: A portion of a TeXspec Subprogram Design Description

20,000 lines of code, is much larger than the TeXspec scripts.

An example screen is shown in figure 8. Here, the structure chart 'SIMALL' from figure 6 is being edited in the upper window. A subprogram on the chart is being modified in the lower window. The GUI can open many windows, so it is contained within an application desktop, which produces only one icon on the user's desktop.

When editing a subprogram (module) on a structure chart, the user specifies the subprogram

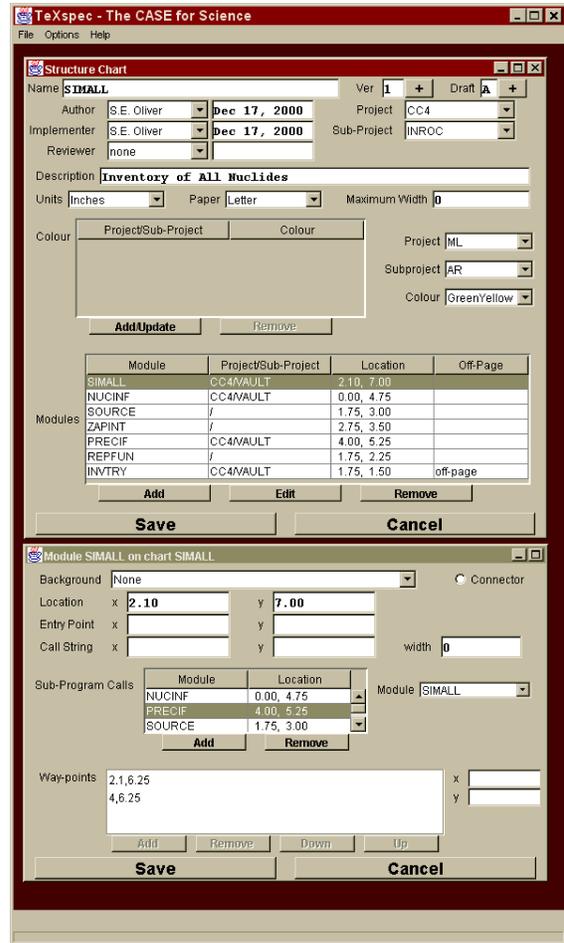


Figure 8: TeXspec GUI editing a Structure Chart

name, and sets a position in x,y coordinates. If the user wants to show an entry point on the chart to this subprogram (useful for charts with multiple entry points), then the location of the entry point must be specified. The location of the call string, and the maximum width of that string is also entered. Calls to other subprograms can be added from a dropdown list of all available modules. For each called subprogram, 'waypoints' determine the shape of the line connecting the two boxes.

Similar editing capability is provided for the overall structure chart in the upper window. Selecting a subprogram from the scroll list at the bottom of the upper window causes the lower window to appear.

### Future Development

The next stage of TeXspec development will be to add some object oriented programming extensions, and a rudimentary interface between the GUI and

an Integrated Development Environment (IDE) to assist in debugging and performance analysis.

The plain text files that store T<sub>E</sub>Xspec data are formatted to be human readable and editable. This allowed T<sub>E</sub>Xspec to be used before the GUI was developed. With the advent of a GUI to interface with this data, the file format may be redefined to an XML syntax.

The Perl scripts may be reimplemented in Java, to permit a more seamless interface between the GUI and the main application.

The application may be divided into a client and a server. This would improve performance, assist in sharing data between users and projects, and provide more robust auditing and version tracking. The system could allow installation of files into a configuration management system. Dependencies between files would be monitored by the server, and ownership would be enforced.

A number of extensions may be made to the GUI, including preview capability for mathematical notations.

More diagram types and programming languages may be supported. In particular, object oriented diagrams may be added, and the full FORTRAN-9x, Java, or Perl syntax may be added.

The GUI support for the graphical products (Data Flow Diagrams and Structure Charts) could be based on editable graphics, or perhaps provide a ‘preview’ window. Having to process the file to see the format of the output is not a optimal.

Some allowance for formal tracing between design and requirements could be provided.

## Conclusion

T<sub>E</sub>Xspec provides a workable solution to computer aided software engineering requirements that are peculiar to scientific programs. It is a significant quality assurance device for these programs.

T<sub>E</sub>Xspec is in use on several projects relating to modeling the disposal of Canada’s nuclear waste. As such, it is a working tool, but is still in the early phases of development. Further enhancement will improve the capability of meeting quality assurance requirements imposed by standards such as CSA N286.7.

## References

- [1] J.J. Bunn. Floppy and flow user manual. 1997.
- [2] B.W. Goodwin, T.H. Andres, D.C. Donahue, W.C. Hajas, S.B. Keeling, C.I. Kitson, D.M. LeNeveu, T.W. Melnyk, S.E. Oliver, J.G. Szekely, A.G. Wikjord, K. Witzke, and L. Wojciechowski. The disposal of Canada’s nuclear fuel waste: A study of postclosure safety of in-room emplacement of used candu fuel in copper containers in permeable plutonic rock. Volume 5: Radiological assessment. Technical Report AECL-11494-5, COG-95-552-5, Atomic Energy of Canada Ltd, 1996.
- [3] B.W. Goodwin, D.B. McConnell, T.H. Andres, W.C. Hajas, D.M. LeNeveu, T.W. Melnyk, G.R. Sherman, M.E. Stephens, J.G. Szekely, P.C. Bera, C.M. Cosgrove, K.D. Dougan, S.B. Keeling, C.I. Kitson, B.C. Kummen, S.E. Oliver, K. Witzke, L. Wojciechowski, and A.G. Wikjord. The disposal of canada’s nuclear fuel waste: Postclosure assessment of a reference system. Technical Report AECL-10717, COG-93-7, Atomic Energy of Canada Ltd, 1994.
- [4] D.E. Knuth. *Literate Programming*. Center for the Study of Language and Information, 1992.
- [5] D.M. LeNeveu. Analysis specifications for the cc3 vault model. Technical Report AECL-10970, COG-94-100, Atomic Energy of Canada Ltd, 1994.
- [6] Quality assurance of analytical, scientific, and design computer programs for nuclear power plants. Technical Report N286.7-99, Canadian Standards Association, 178 Rexdale Blvd. Etobicoke, Ontario, Canada M9W 1R3, 1999.
- [7] S. Oliver, K. Dougan, K. Kersch, C. Kitson, G. Sherman, and L. Wojciechowski. Unit testing - a component of verification of scientific modelling software. In T.I. Oren and G.B. Birta, editors, *1995 Summer Computer Simulation Conference*, pages 978–983. The Society for Computer Simulation, 1995.
- [8] N. Ramsey. Literate programming simplified. *IEEE Software*, September 1994.
- [9] K. Rose. Very high level 2-dimensional graphics. In *1997 TeX User Group Conference*. TeX User Group, 1997.
- [10] L. Wall, T. Christiansen, and R. Schwartz. *Programming Perl*. O’Reilly & Associates, 101 Morris Street, Sebastopol, CA 95472, second edition, 1989.
- [11] E. Yourdon. *Modern Structured Analysis*. Yourdon Press/Prentice-Hall, 1989.

# GELLMU: A Bridge for Authors from L<sup>A</sup>T<sub>E</sub>X to XML

William F. Hammond

Department of Mathematics & Statistics, University at Albany

hammond@math.albany.edu

<http://www.albany.edu/~hammond/>

## Abstract

GELLMU, which stands for “Generalized Extensible L<sup>A</sup>T<sub>E</sub>X-Like Markup”, is a system for using L<sup>A</sup>T<sub>E</sub>X-like markup, though not L<sup>A</sup>T<sub>E</sub>X itself, to write consciously for a markup language in the SGML category or in its popular XML subcategory. The *basic* level of GELLMU offers a way to use L<sup>A</sup>T<sub>E</sub>X-Like notation together with a L<sup>A</sup>T<sub>E</sub>X-Like *newcommand* (with arguments) macro facility to write web pages. The *advanced* level of GELLMU enables one additionally to incorporate certain L<sup>A</sup>T<sub>E</sub>X-Like features, such as the use of a blank line for a new paragraph, in writing for an SGML language. The didactic GELLMU production system provides an “article” XML language, with some resemblance to L<sup>A</sup>T<sub>E</sub>X itself, that is a rigorous domain for translation to other formats.

## Author Level Markup

Inasmuch as the World Wide Web is becoming an important library resource, one wants one’s publications to be accessible online, and one wants web-crawling robots to be able to catalogue them properly.

Despite the popularity of Adobe’s Portable Document Format (PDF) the distribution of PDF reading software is not as widespread as the distribution of web browsing software, and web-crawling robots often do not scan the contents of PDF documents.

What is available for the L<sup>A</sup>T<sub>E</sub>X author toward this end? More specifically, consider the following situations:

**Online publication archives** Specifically, I would like to cite the T<sub>E</sub>X/L<sup>A</sup>T<sub>E</sub>X-based e-print archive begun at Los Alamos in the early 1990’s by Paul Ginsparg, now known as “ArXiv”, a participant in the [Open Archives Initiative](#). While in its early time the term *e-print* was understood to mean “electronic pre-print”, ArXiv has more recently become a repository for established journals including, for example, the highly regarded *Annals of Mathematics*, which was founded in 1884 by Ormond Stone of the University of Virginia, and Ginsparg now tells us that the term *e-print* denotes “self-archiving by the author” under a new overall academic publication<sup>1</sup> design.

<sup>1</sup> Paul Ginsparg, “Electronic Clones vs. the Global Research Archive”, <http://arXiv.org/blurp/pg00bmc.html>.

**Course handouts** How can a college teacher prepare course handouts for both paper and online distribution? If the teacher writes L<sup>A</sup>T<sub>E</sub>X, some manual intervention will likely be needed in order to obtain correct HTML. If the teacher writes HTML, then the paper distribution<sup>2</sup> will be limited by what can be expressed in HTML, which is not as rich a markup as L<sup>A</sup>T<sub>E</sub>X.

**TUG articles** Before preparing a TUG 2001 article an author is asked to read *Preparation of documents for multiple modes of delivery* by Ross Moore, which is available on the web only as a two-column PDF printed page image. From this article one might conclude that carefully prepared L<sup>A</sup>T<sub>E</sub>X may be suitable for translation to HTML although no HTML version of the article seems to be available.

**GNU documentation** While working for TUG on the T<sub>E</sub>X Directory System (TDS) guidelines — see [/tds/standard/](#) at CTAN — in January 1998, Ulrik Vieth produced a L<sup>A</sup>T<sub>E</sub>X document and a tailored program for translating that document into *Texinfo*, the language of the GNU documentation system.

*Texinfo* is a T<sub>E</sub>X-based system that pre-dates HTML. Its original purpose was to provide both print and (early online hypertext) *Info*

<sup>2</sup> If the HTML is correctly written, then robust translation to L<sup>A</sup>T<sub>E</sub>X is possible.

versions of GNU software project documentation. When HTML came along, it was possible to provide fairly reliable translation from *Texinfo* to HTML because *Texinfo* is a well-structured markup. In fact, *Texinfo* is very nearly equivalent to an SGML language, and, Daniele Giacomini in August 2000 came up with an effort in that direction: *Sgmltexi*.

Although programs are available for translating carefully structured L<sup>A</sup>T<sub>E</sub>X into HTML and sometimes into XML extensions of HTML, this method of generating online content for the basic level of the web sometimes requires manual intervention. A more direct approach to the world of SGML offers better prospects for long-term access to new web formats without sacrificing access to the quality of print typesetting that is available through L<sup>A</sup>T<sub>E</sub>X.

### The basics of *basic* GELLMU

In looking over Vieth's set-up for the TDS document in the late spring of 1998, I arrived at the idea of using L<sup>A</sup>T<sub>E</sub>X-like notation for conscious writing in document languages under SGML and I have written a program in the GNU Emacs Lisp language, the GELLMU syntactic translator, for converting this L<sup>A</sup>T<sub>E</sub>X-like markup to SGML markup.

The advantage of SGML markup is that each markup language (formally document type) under the SGML umbrella constitutes a structured domain for the application of automatic processors that are easy to create under any of a number of structured processing frameworks. There are frameworks accessible in standard computing languages, and there is also a recent framework *xmltex* by David Carlisle for writing T<sub>E</sub>X typesetting routines for XML document types.

The root idea in using L<sup>A</sup>T<sub>E</sub>X-like markup for the conscious writing of markup under SGML is the simple syntactic correspondence between markup such as

```
some \em{emphasized} text
```

on the one hand, and the markup

```
some <em>emphasized</em> text
```

on the other.

Most L<sup>A</sup>T<sub>E</sub>X commands are analogous to SGML elements. Moreover, the attribute list associated with an SGML element can be made to correspond with a L<sup>A</sup>T<sub>E</sub>X command option. For example,

```
\a[href="http://foo.dom/"
]{The Foo Domain}
```

matches

```
<a href="http://foo.dom/"
>The Foo Domain</a>
```

### Enhancement with `\newcommand`

The idea of using L<sup>A</sup>T<sub>E</sub>X-like syntax for conscious writing under an SGML document type gains power when one realizes that although the notion of SGML entity provides, among other things, simple macro expansions, there is no provision under SGML for macros that take arguments. Moreover, there is no obvious method of extending SGML systems to accommodate macros with arguments apart from the idea of extending a document type.<sup>3</sup>

GELLMU provides a L<sup>A</sup>T<sub>E</sub>X-like meta-command<sup>4</sup> called *newcommand* that may be invoked with arguments. For example, if one writes

```
\newcommand{\afoo}[2]{%
\ a[href="http://www.foo.org/#1 "] {#2}}
```

then a subsequent invocation

```
\afoo{tex-archive/tds/}{TDS at Foo}
```

will yield (without line breaks):<sup>5</sup>

```
<a href=
"http://www.foo.org/tex-archive/tds/"
>TDS at Foo</a>
```

This *newcommand* markup differs from that of L<sup>A</sup>T<sub>E</sub>X in that it is classical macro substitution rather than vocabulary expansion. Since the syntax of a *newcommand* invocation is very similar to that of an SGML element, the use of *newcommand* can, apart from its on-the-fly convenience, be a help in the development of SGML document type extensions. A new name in a test document can be moved from being that of a macro to being that of an element simply with the removal of a *newcommand* definition.

### SGML vs. XML

Basic GELLMU as enhanced by its macro facility is as far as one can sensibly go toward conscious writing under a language in the restricted subfamily of SGML document types known as XML.

From one viewpoint the differences between SGML and XML are not very important since most correct documents under the larger category can, if

<sup>3</sup> While document type extensions require enough work that they cannot be spontaneous, they provide a sound way to avoid the tangles that can arise working with T<sub>E</sub>X or L<sup>A</sup>T<sub>E</sub>X when attempting the simultaneous use of conflicting macro packages.

<sup>4</sup> In GELLMU while a *command* corresponds to an SGML element, a *meta-command* is something having the same syntax as a command that does not correspond to an SGML element and instead receives resolution into other SGML markup under the syntactic translator.

<sup>5</sup> The syntactic translator maintains line number alignment between its input and its SGML output so that line numbers used by SGML parsers in flagging errors match those in source markup.

correct, be automatically translated into equivalent documents under XML. For example, classical HTML that passes validation can be translated into the newer XML form of HTML using either James Clark's classical SGML library *SP* or Dave Raggett's program *tidy*.

However, the rules for XML were designed to make things easy for processors rather than for humans, and for that reason an author writing toward an ultimate XML document type usually is well-advised to write for a version of the document type under more author-friendly SGML rules.

For example, if in an SGML language a forced linebreak is represented by the defined-empty element *brk*, then the markup `<brk>` is sufficient, whereas under the more restrictive XML version of the same language, either the markup `<brk></brk>` or its abbreviated form `<brk/>`<sup>6</sup> must be used. For GELLMU this means that the markup `\brk` and the markup `\brk{}` are interchangeable under SGML, except for the case of `\brk` abutting a following character without intervening whitespace, but not equivalent under XML. GELLMU provides the form `\brk`; to represent the abbreviated form `<brk/>` of an element that is defined as empty.

### Advanced GELLMU

Basic GELLMU deals with markup languages more or less at the level of syntax without getting to the level of grammar.

Advanced GELLMU may be used to roll language-independent grammatical concepts into the picture.

The first of these is L<sup>A</sup>T<sub>E</sub>X-like multiple argument/option syntax. For example under advanced GELLMU the markup

```
\frac{a x + b}{c x + d}
```

is converted in syntactic translation to

```
<frac><ag0>a x + b</ag0><ag0>c x + d</ag0>
```

That is, a chain of '{', '}' pairs and '[', ']' pairs following a command without intervening white space between the command name and the first delimiter nor between a close delimiter and the next open delimiter in the chain, constitutes an SGML element whose content begins with a sequence of generic positional arguments (tag name *ag0*) and options (tag name *op0*). Without knowledge of the document type it cannot be determined if a name used with multiple argument/option syntax has only *ag0*, *op0* content. The syntactic translator

<sup>6</sup> Moreover, some confusion may arise from the fact that under the SGML syntax (formally SGML declaration) specified for HTML neither of these XML forms of markup would not be permitted.

provides a list variable consisting of names that have only this type of content and that, therefore should be given close tags after the sequence of arguments and options. Absent that, the author must provide a close tag unless an SGML parser can infer it, and even in that case, if the element can appear in the mixed content model of another element such as, for example, a paragraph, then the parser's automatic placement of a close tag could lead to the unwanted collapse of a word boundary similar to that which occurs in L<sup>A</sup>T<sub>E</sub>X when an author's careless markup

```
\TeX benchmark
```

gets typeset as "T<sub>E</sub>Xbenchmark" instead of as "T<sub>E</sub>X benchmark".

If multiple argument/option syntax is used, then there is ambiguity on the nature of the first pair of chained delimiters if it is the pair '[', ']' — whether it represents an *op0* or an attribute list. Therefore, in this case it is required that it is an attribute list if the first character after its '[' opening delimiter is a colon (':').

In basic GELLMU the following four of the ten L<sup>A</sup>T<sub>E</sub>X-special characters are special:

```
\ { } %
```

Additionally, the character '#' is special when used in the definition of a *newcommand*, the characters '[' and ']' are special when used for L<sup>A</sup>T<sub>E</sub>X-like option syntax, and the character '&' is special when followed immediately by a letter since then it is the introducer for SGML entity invocations.

Advanced GELLMU provides for the possibility of giving traditional L<sup>A</sup>T<sub>E</sub>X meaning to '&' when not followed immediately by a letter and also to the other four L<sup>A</sup>T<sub>E</sub>X-special characters, which are

```
_ ^ $ ~
```

Additionally, it provides for the possibility of giving traditional L<sup>A</sup>T<sub>E</sub>X-like meaning to other short forms of markup such as "`\( . . . \)`" for inline math, "`\[ . . . \]`" for displayed math, "`--`" for a range-dash, "`---`" for a punctuation-dash, "`\`" for an inter-word space, "`\,`" for a short space, and others including also, if desired, the use of blank lines, as appropriate, for paragraph boundaries.

### The didactic production system

The conversion of both *basic* and *advanced* GELLMU source markup to SGML is performed by my program called the *syntactic translator*.

If one wishes to write consciously for a public document type such as HTML or the [Text Encoding Initiative's](#) TEI using GELLMU's L<sup>A</sup>T<sub>E</sub>X-like

syntax, the syntactic translator is the only part of GELLMU that will be of interest.

The optional features of advanced GELLMU described above can only be used when one is writing for a document type that provides markup in which the corresponding concepts have representation. For example, L<sup>A</sup>T<sub>E</sub>X-like use of the character ‘~’ for non-breaking space requires a markup that provides non-breaking space.

Moreover, if blank lines are going to be paragraph boundaries, then the syntactic translator will need a list of element names before which a new paragraph does not make sense, and, since there is no separate provision for a list of names after which a paragraph must end, the document type cannot be XML.

The GELLMU didactic production system provides such a document type and also provides tools, which can be used as inter-changeable components, for working with that document type.

The didactic production system consists of the syntactic translator and the following additional components:

1. An SGML document type called “article”.
2. Its XML cousin, also called “article”.
3. A program for translating SGML article to XML article.
4. A program for translating XML article to HTML.
5. A program for translating XML article to L<sup>A</sup>T<sub>E</sub>X.

The document type is intended to be comfortable for authors with past experience in L<sup>A</sup>T<sub>E</sub>X.

The document type and the components are didactic. They are intended to illustrate how such a system can be assembled from inter-changeable components. They are not finished in any sense, and each has shortcomings.

They do serve, I hope, to demonstrate to the community of L<sup>A</sup>T<sub>E</sub>X authors that it will find no limitations in this approach to document production.

At the same time it is intended to provide a whole new way of thinking about the subjects of package design and class design.

Its unfinished nature is intended to make it relatively easy for those who are so inclined to move in various ways to finish such a system that fits their needs.

### Production of this Document

This document and the slides used during its presentation were prepared with the GELLMU didactic production system. Pre-publication versions of the sources and various automatic formattings are available in [the author’s web](#).

Subsequent to the GELLMU run on this document a copy of its L<sup>A</sup>T<sub>E</sub>X output was manually modified for conformance with TUG guidelines. If I were going to submit a number of such TUG articles, it would be worthwhile to make another variant of the L<sup>A</sup>T<sub>E</sub>X formatter for TUG.

# Creating Math Web Documents

Bob Caviness

University of Delaware

[caviness@mail.eecis.udel.edu](mailto:caviness@mail.eecis.udel.edu)

## Abstract

In this workshop, we will show how to combine `techexplorer`, Java, and JavaScript to produce interactive web content with a mathematical flavor. We will start with a brief overview of `techexplorer` (the IBM browser plugin for displaying  $\TeX$  and MathML markup on the web), Java, and JavaScript. Then we will discuss in some detail an application that shows how to tie all these tools together with HTML in a natural way that leverages each of their strengths. The result will be a rather sophisticated interactive calculator, but the ideas are useful for many other applications.

# The Makor System for Typesetting Hebrew

Alan Hoenig

Department of Mathematics, John Jay College, City University of New York

[ahoenig@suffolk.lib.ny.us](mailto:ahoenig@suffolk.lib.ny.us)

## Introduction

I'd like to describe this morning a new  $\text{\TeX}$ -based system for typesetting Hebrew. I call this system *Makor*, from the Hebrew word for 'source'. I don't know how many Hebrew speakers there are here today—I'm not one myself—so I'll describe this project from the point of view of coping with yet another new font subject to some curious rules. The worth of this project—insofar as there is any—lies in a few areas:

1. First of all, I hope it will be of use not only for documents in Hebrew, but also in languages also using Hebrew fonts, such as Yiddish and Ladino.
2. Secondly, I hope this project serves as a model for adapting the  $\text{\TeX}$  engine for typesetting foreign languages. The problems associated with Hebrew are quite different from ones associated with European-language typesetting, and I hope this inspires people who want to typeset, say, Devanagari or Arabic efficiently with  $\text{\TeX}$ .
3. Finally, this project would have been orders of magnitude more difficult to implement without the convenience and power of virtual fonts. Virtual font technology is now over ten years old, and yet this area remains *terra incognita* for most  $\text{\TeX}$  users. Perhaps this project can be interpreted as additional instruction in virtual fonts.

**Prior work** Any work on Hebrew typesetting must acknowledge Yannis Haralambous's great *Tiqwah* system. It is described in several places [1], and the last illustration in that paper must rank as one of the milestones in  $\text{\TeX}$  typesetting history. Not only does it include two systems of diacritic vocalizations (one for pronouncing—see below—and one for liturgical chanting), but it does so so that the final product is a thing of great beauty. His METAFONT-created font is especially noteworthy. It's not clear, though, how to extend Yannis's work to apply to other fonts.

More recently, Sivan Toledo [2] has developed a PostScript system which cooperates with  $\text{\TeX}$  for Hebrew. The illustrations that appear in his article are intriguing, but I have not been successful in

using it myself. This article contains a useful list of sources for additional information.

The CTAN archives contain a generous collection of Hebrew meta-fonts. Generally speaking, though, these fonts are not of sufficiently-high caliber for use with a comprehensive typesetting system. Most of them, for example, do not contain dotted letter variants or the vowel marks (see below for explanations). In the same way, the dozens of Hebrew fonts available on the Web are not  $\text{\TeX}$ -worthy. So far, the only acceptable and freely-available font I've been able to lay my hands on is the font that is part of the Omega system (due to Yannis Haralambous and John Plaice). If anyone can point me to high quality fonts in addition to `OmegaSerifHebrew`, I would be grateful.

## Describing Hebrew

For typesetting purposes, we can consider Hebrew in the following terms. It is a caseless language read from right to left (but from top to bottom). Although it is caseless, most letters have two forms—with and without a (more-or-less centrally located) dot. There are a total of twenty-seven letterforms (not counting the dotted variants), but five of those occur only at word endings. You can see these characters in figure 2. There they are presented in three groups separated by asterisks '\*'. The last group (read right-to-left!) are the word-final letters.

Actually, all these letters are consonants. Vowel sounds are represented by diacritical marks which appear below the consonant. However, these marks are not mandatory, most often being omitted entirely from adult reading matter. They appear in language texts, children's books, and on occasion to make clear how to pronounce a new word or a foreign name. These diacritics differ from European accents in one intriguing way. They are centered *not* with respect to a central axis but with respect to a particular axis whose location varies from letter to letter. Actually, the location of this axis should be specified by the font designer. (I am indebted to Yannis Haralambous for this crucial observation.)

Vocalized consonants appear in figure 3. The same vowel (best described as a simple dot under a

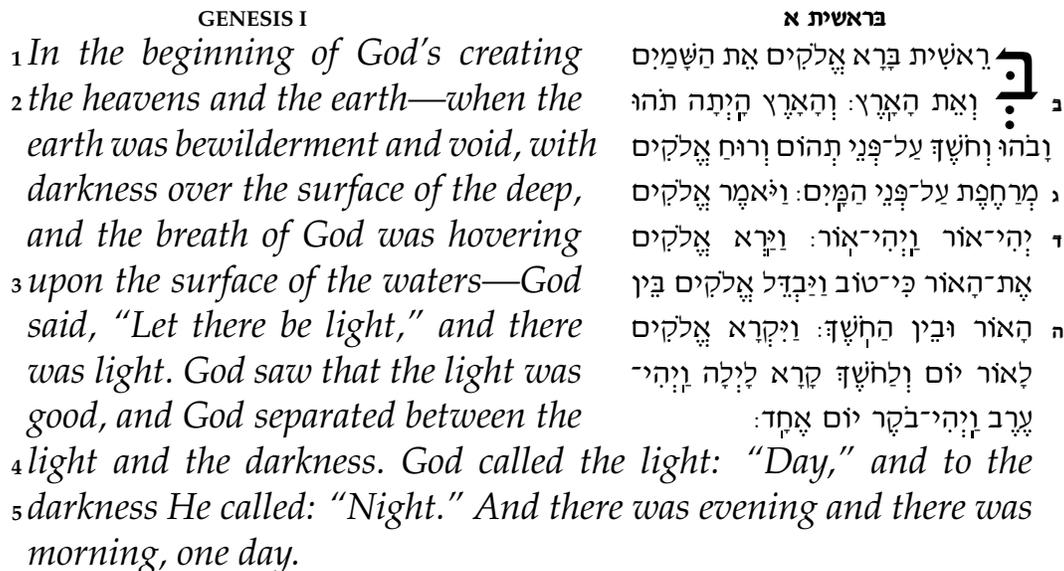


Figure 1: A proposed Biblical layout.

א ב ג ד ה ו ז ח ט י כ ל מ נ ס ע פ צ ק ר ש ט  
ש ת \* א ב ג ד ה ו ז ח ט י כ ל מ נ ס ע פ צ  
ק ר ש ט ש ת \* ך ם ן ף ץ

Figure 2: The base characters in the Hebrew alphabet.

consonant) appears under two different letters. As you see, in the left consonant, it appears under the right letter stroke—about as far away from centering as you can get! In the remaining letter, the vowel is closer to the center, but not exactly at it.

Further considerations pertain to keyboard entry. I wanted to be able to use a standard computer keyboard to type my Hebrew source. Most of the Hebrew letters correspond to Latin letters but there are complications. For various reasons, several Hebrew consonants represent the same sound, and there are a handful of Hebrew letters that have no English counterpart.

For those readers who have trouble delaying their gratification, I present in figure 4 a sample of fully-vocalized Makor output. This selection is drawn from the final few verses of Deuteronomy.

**Goals and implementation** Don't worry—I have no intention of describing the features of Makor that properly belong in the User manual. But even without these details, there is enough interesting stuff going on that I hope you won't be bored, or at least *too* bored. On occasion, I suppose I will have to refer to details of the system, but I will try to keep those references to a minimum.

I set myself the goal of designing a system that uses a reasonable keyboard entry mechanism to typeset Hebrew according to the rules set out above and consistent to the high quality of which T<sub>E</sub>X is capable. You can decide how well I succeeded.

Several of these considerations are easily dealt with. To get right-to-left text, we demand that you use Makor with any of the extended T<sub>E</sub>X's that do this, such as Omega or eT<sub>E</sub>X. I noticed some minor but distinctive differences in the typeset output depending on which of these programs you use, so try both of them and use the best. The differences all appear in spacing before and/or after a bit of right-to-left text.

T<sub>E</sub>X's virtual font mechanism easily handles end-of-word glyphs, so that's not a problem.

T<sub>E</sub>X's ligature mechanism is quite helpful in this project. Although several Hebrew sounds are foreign to English, an English (or at least American) set of conventions has arisen for their representation. For example, Hebrew contains the guttural throat-clearing sound that is present in some German words, such as in the name of the composer J. S. Bach. We agree in Makor to type *ch* whenever we



**Figure 3:** Hebrew vowel diacritics are not centered.

וַיִּמַּת שֵׁם מֹשֶׁה עֶבֶד־יְהוָה בְּאֶרֶץ מִזְרַח עַל־פִּי  
 יְהוָה: וַיִּקְבֹּר אֹתוֹ בְּגִי בְּאֶרֶץ מִזְרַח מִבַּיִת פְּעוֹר  
 וְלֹא־יָדַע אִישׁ אֶת־קְבֻרָתוֹ עַד הַיּוֹם הַזֶּה: וּמֹשֶׁה  
 בְּן־מֵאָה וָעֶשְׂרִים שָׁנָה בָּמָתוֹ לֹא־כָּהָתָה עֵינָיו  
 וְלֹא־נָס לְחָה: וַיִּבְכּוּ בְּנֵי יִשְׂרָאֵל אֶת־מֹשֶׁה בְּעֶרְבַת  
 מִזְרַח שְׁלֹשִׁים יוֹם וַיִּתְּמוּ יָמָיו בְּכִי אֲבָל מֹשֶׁה:  
 וַיְהוֹשֻׁעַ בְּן־נוּן מֵלֵא רֹחַ חֲכָמָה קִי־סַמָּךְ מֹשֶׁה  
 אֶת־יָדָיו עָלָיו וַיִּשְׁמְעוּ אֵלָיו בְּנֵי־יִשְׂרָאֵל וַיַּעֲשׂוּ  
 כְּאֲשֶׁר צִוָּה יְהוָה יְהוָה יְהוָה אֶת־מֹשֶׁה: וְלֹא־קָם נָבִיא עוֹד  
 בְּיִשְׂרָאֵל כְּמֹשֶׁה אֲשֶׁר יָדָעוּ יְהוָה יְהוָה אֶת־פְּנִיָּם:  
 לְכֹל־הָאָרֶץ וְהַמּוֹפְתִים אֲשֶׁר שָׁלַח יְהוָה לַעֲשׂוֹת  
 בְּאֶרֶץ מִצְרַיִם לְפָרְעֹה וְלְכֹל־עַבְדָּיו וְלְכֹל־אֶרֶץ  
 וְלְכֹל הַיָּד הַחֲזָקָה וְלְכֹל הַמּוֹרָא הַגָּדוֹל אֲשֶׁר עָשָׂה  
 מֹשֶׁה לְעֵינֵי כָּל־יִשְׂרָאֵל:

**Figure 4:** Fully vocalized Hebrew output.

want that gutteral to appear in text. In a Hebrew font, `ch` is a special ligature that selects that letter.

**Vowels** The nexus of the project involved the representation and typesetting of vowels. Deciding on input conventions was straightforward—first of all, I let English consonants and only these consonants represent Hebrew consonants, and the flip side of that coin is that English vowels and only these vowels represent Hebrew vowels. However, the conventions of reading Hebrew demand that the vowel—which is a diacritic accent, remember, not a distinct letter—*follows* the consonant, a convention contrary to normal `TeX` conventions. At the same time, each time a consonant is typeset, we need to access somehow the location of its visual axis to know where to place the diacritic accent.

It's too bad that there is no `\lastchar` primitive in `TeX`, the way there is a `\lastbox` and a few other `\last...` things. To keep track of the most recent character, I had to coopt the italic correction, one of the few signposts that `TeX` erects to mark a recently set character. Since there really is no such thing as a Hebrew italic, at least in my fonts, its loss for marking genuine italic corrections is not missed

by anyone. In my special fonts, I redefined the italic corrections to equal the character code of the glyph. Italic corrections are just kerns, and they can be removed with `\lastkern` and examined to see how big they are, and thus which character has just been placed in the `.dvi` file.

Once I know which character I've just set, I can get the location of the optical axis. I previously made sure to store the distance of this optical axis from the central axis as a *kern* between the character in question and a special character which, although present in the font, is not used otherwise to typeset. It's possible to measure this kern, and so to know where and how to center a vowel. The Makor macros do all this for us.

There's one other aspect of vowels that I wanted to take into account. I mentioned earlier on that vowels are often omitted in actual Hebrew typesetting. Nevertheless, one might want to include them in the source document, if for no other reason than to make reading of the source file less of a bother. For example, there is a (reasonably) well-known Jewish holiday that occurs usually in December. According to my Makor input conventions, you would typeset it, fully vocalized, as

`chanookauh`.

To eliminate the vowels in the output, you eliminate them in the input, so the source should now read `chnkh`.

Does any reader doubt that this input is ugly, unpleasant, and difficult to read? In Makor, there is a companion font which maps the vowels to null characters, so you could de-vocalize the output while maintaining ease of reading in the input by typing something like

```
{\CXLV chanookauh}
```

Here, the command `\CXLV` is not a Roman numeral, but rather the instruction to cancel the vowels. But it's really just a font selection command, and you can play with it as you would any font selection command.

**Active characters** The only way I could differentiate between consonants and vowels in the source is by making all vowels active. For those still breaking their teeth on  $\TeX$ , you should know that making a character active makes that character eligible to receive macro treatment—you can assign it a definition as you would to a control sequence.

This dual categorization leads to far few problems than I expected, but of course one immediate consequence is that virtually all of our familiar arsenal of  $\LaTeX$  and  $\TeX$  commands appear to become useless. A simple command like `\hspace` is no longer—at least from the point of view of the  $\TeX$  engine—an escape character followed by a sequence of letters. Here, we have an escape character (the backslash) followed by the letters `hsp`, followed by an active character `a` whose definition  $\LaTeX$  tries to plug in, followed by a `c` (which is typeset in the document), followed finally by an active `e`. There are straightforward ways out of this impasse, but it is a sticky wicket to be sure.

**Numbers** Numbers introduce an amusing problem as well. Although it's true that all Hebrew is typeset right to left, the only exception is for (ironically named) Arabic numerals when they appear in a Hebrew document—they are to appear left to right. How then do we typeset them?

If you think about it just a bit, you would be tempted (as I was) to simply come out of Hebrew mode to typeset the numerals and return to Hebrew mode for the remainder of the text. If `\[` and `\]` are the toggles for entering and exiting Hebrew mode, then you might typeset

אבגדה12345קרשת

schematically as

```
\[ first bit of Hebrew \]%
12345%
```

```
\[ second bit of Hebrew \]
```

If you do that, you get instead

קרשת12345אבגדה

which, for those not proficient in Hebrew reading, is schematically the same as

```
\[ second bit of Hebrew \]%
12345%
```

```
\[ first bit of Hebrew \]
```

If you think hard about this for a few moments, you see why this is so, and why it's necessary to introduce a `\NUM` macro to typeset the numbers. `\NUM` involved a recursive macro, easier to create than you might think since I plundered it wholesale from a timely example in the *TeXbook*.

**Fleshing out the font** Hebrew fonts tend to be sparse. What you get for your money is generally just the special Hebrew glyphs. If you're lucky, you get matching numerals, but almost never do you get a full complement of punctuation and special symbols. By virtue of the magic of virtual fonts, it's easy to flesh out the virtual Hebrew fonts with characters from an existing Latin font, but from which one?

It seems to me that the best match, at least for the `OmegaSerifHebrew` I used for my development work, is Palatino regular. However, it's a proprietary PostScript font, and although common (it's one of the fonts typically resident in any PostScript-enabled printer), I still wasn't comfortable assuming that everyone had access to it, and access to it under the fontname scheme that most modern  $\TeX$ 's adhere to.

Consequently, I used my second choice, although it's still a good choice. I chose the Computer Modern Fibonacci font `cmfib8` as the font-flesher-outer. Although `cmfib8` is a favorite of mine, I have heretofore not been able to find a legitimate use for it. Recall that this quirky font has `META-FONT` parameter values chosen from the sequence of Fibonacci numbers, an interesting group, at least from a mathematician's point of view. I used `cmfib8` scaled by 699, where 699 is the sum of the eleventh and fifteenth Fibonacci numbers (if that's at all significant!).

The two examples above show the Hebrew letters in combination with Fibonacci numerals.

**Further examples** I'd like to present now just a few examples of Hebrew-English typesetting with Makor that at least look interesting. In figure 5 you

Rabbinic Hebrew (RH) does not differ greatly from Biblical Hebrew (BH) in its inflection of the noun, although the neutralization of final *mem* and *nun* means that the masculine plural is often, as in Aramaic, יָ. Apart from the more frequent use of the archaic feminine suffix ת־ as in בְּהֵנֶת ‘priest’s wife’ and אֲקֵמֶת ‘dumb woman’, RH also employs the suffixes ית־ and ות־ for example אַרְמִית ‘Aramaic’ and עֲבָדוֹת ‘servitude’. RH developed distinctive feminine plural suffixes in אֲוֹת־ (Babylonian) or יוֹת־ (Palestinian), for example מְרַחְצֵיּוֹת/מְרַחְצֵיּוֹת ‘bath-houses’ and יוֹת־, as in מְלְכֵיּוֹת ‘kingdoms’ for BH מְלְכֵי, for nouns ending in ות־ in the singular. Masculine plural forms sometimes differ from those that would be expected, or are normally found, in BH, for example, נִזְקֵי from נִזֵּק ‘damage’, שְׁוֹרִים from שׁוֹר ‘ox’, שְׁוֹקֵים from שׁוּק ‘market’, צְדָדִים from צֵד ‘side’, חֲצֵי from חֶצֶה ‘half’, and שְׁלֵחִין from שְׁלִיחַ ‘envoy’. The same is true of feminine nouns, for example אוֹתֵי from אוֹת ‘letter (of alphabet)’, בְּרִיתוֹת from בְּרִית ‘covenant (without plural in BH)’, and אֲמֵהוֹת from אִם ‘mother’.

Some masculine nouns take the feminine plural suffix ות־, for example, חֲנוּת from חֵן ‘favour’, כְּלָלוֹת from כָּלַל ‘rule’, תִּינְקוֹת from תִּינוּק ‘baby’, חֲיָלוֹת from חַיַל ‘army’, עִירוֹת from עִיר ‘city’, and מִימֵת from מַיִם ‘water’. Similarly, there are some feminine nouns which take the masculine plural suffix ים—יוֹנֵים from יוֹנָה ‘dove’, נְמֵלִים from נְמֵלָה ‘ant’, and בֵּיצִים from בֵּיצָה ‘egg’, for example. Occasionally, both types of plural are evidenced, as with יָמֵם/יָמֵם from יוֹם ‘day’ or שְׁנֵים/שְׁנוֹת from שָׁנָה ‘year’, with each form having a slightly different shade of meaning and the ‘feminine’ variant only used with suffixes. In RH we sometimes find plurals of nouns only attested in the singular in BH, for example אֲבָרִים from אֲבָר ‘limb’, דְּשָׁאִין from דְּשָׁא ‘grass’, and תְּמִידִים from תְּמִיד ‘daily sacrifice’. Likewise, there are singular forms of nouns only attested in the plural in BH, for example אֲלֵמוֹג ‘coral-wood’, בֵּיצָה ‘egg’, and בָּצֵל ‘onion’. The dual is used more than in BH, with existing forms retained and new ones created, for example מְסַפְרִים ‘scissors’ and בְּנֵתִים ‘meanwhile’. (1993: Saenz-Badillos, *A History of the Hebrew Language*, Cambridge University Press, pp.188-89.)

**Figure 5:** Mixing Hebrew and English together.



Figure 6: Page from a proposed Hebrew primer.

see how easy it is to mix lots of Hebrew with lots of English.

Although you do have to keep aware that vowels are active, it's surprising how few restrictions there actually are on Makor typesetting. In figure 6, we see the page of a book as it might be typeset for beginning students in the language.

Liturgical books frequently demand interesting layouts. In figure 1, verse numbering is automatic. The look of figure 7 mimics that of certain books of legal exposition.

### Future work and further comments

Several important tasks remain in this project. Of course, as in all such projects, it's important to eliminate remaining bugs, continue macro support and so on; that goes without saying. The user manual has to be greatly expanded. Eventually, the Makor package will consist of a Perl script in addition to fonts. The script will greatly aid users who want to convert their own fonts to the form the Makor macros expect.

I also plan to add additional fonts—fonts for the typesetting of Yiddish and Ladino, and fonts which allow users to type their input from an Israeli keyboard. The Perl script I just mentioned will facilitate this task immeasurably. There are apparently a few other dialects of Hebrew plus whatever that use the Hebrew alphabet, and if I can find out more about them, I would like to support them as well. (Most people know what Yiddish is—sort of a creole between medieval German and Hebrew. Ladino stands in the same relation to medieval Spanish and Hebrew.)

**Extended T<sub>E</sub>X's** I've been out of the loop for a long time—not that I was really ever *in* the loop—as far as the development of extended T<sub>E</sub>X goes. The two such projects I know about are Omega, under the direction of Yannis Haralambous and John Plaice, and a larger team for eT<sub>E</sub>X under the stewardship of Phil Taylor. There were several facilities that are missing in T<sub>E</sub>X whose presence would have made my life much easier.

First off, I would have loved a facility that gives me the character number of the most recently set character, perhaps to be called `\lastchar`. Because of the presence of ligatures in a font, even knowing the last character in the source file is not sufficient for knowing the last typeset character.

I would have liked the possibility of having an entire series of character dimensions allotted for each character, in the same way that font dimensions are allotted for each font. The first four character dimen's would coincide with the width, height, depth, and italic correction of the character, but any additional `\chardimen`'s would have meanings given to them by the font designer. For example, a fifth and sixth `chardimen` for these Hebrew fonts might correspond to the location of the upper and lower visual axes. Some letters might have additional `\chardimen`'s giving the location of forbidden portions, areas where diacritical marks may never appear. I feel sure that this would be useful concept for other languages, for reasons I could not begin to guess. Of course, I have no idea how easy that would be to implement, nor what it would do to the size of the resulting `.tfm` file.

Finally, I would have liked additional category codes available to me. Classifying vowels as active, though useful, is fraught with some peril. Moreover, I suspect that for other languages, there are several categories of glyphs that I would not have been able to handle without additional `\catcode`'s.

### Getting the software

The current version of Makor software is always available in the CTAN archives, in the directory

`languages/hebrew/makor`

### Plea to the reader and user

If any reader or user could point me to additional high-quality Hebrew fonts, I'd be most grateful. As of this writing, I am unable to test my macros and information-encoding schemes on other fonts.

Please don't hesitate to forward additional suggestions, queries, requests for clarification, and so on to me; `email` is best. Thanks for your interest.



# Modernizing Computer Modern Fonts

Alan Hoenig

Department of Mathematics, John Jay College, City University of New York  
[ahoenig@suffolk.lib.ny.us](mailto:ahoenig@suffolk.lib.ny.us)

Many T<sub>E</sub>X users know that the workhorse T<sub>E</sub>X font, Computer Modern Roman (CMR; we'll refer frequently to the Computer Modern fonts as CM), was modelled after a real-life font called Modern 8a. To name a font in this way is to invite questions: are there other members in the Modern font sequence? what does the suffix 'a' denote? I can't claim comprehensive answers to these questions, but I've kept my eyes and ears open over the years, and have occasionally come across other members of this font family. One of my most intriguing possessions is of a novel by Herman Melville set entirely in Modern fonts, namely numbers 8 (no suffix) and 4. Appropriately enough, the novel is called *Typee*! These types are extremely readable. In fact, they positively invite the reader to set a spell and lose themselves in the volume.

The digital age has not been kind to this super-family of fonts. As far as I can tell, only one of the Modern types is available in digital form. (I welcome correction on these matters.) The font in question is called simply Monotype Modern, and is, to my eyes, extremely handsome; see figure 1 for an extended selection .

Monotype Modern bears an unquestioned resemblance to Computer Modern (or is it the other way 'round?), but somehow the printed page using this font comes across as brighter and more interesting, at least to my eyes. The T<sub>E</sub>X user quickly comes a cropper when she tries to adapt this font for use with T<sub>E</sub>X—there are no expert fonts available for Monotype Modern, so matching small caps and the full suite of double-f ligatures appear to be unavailable.

The purpose of my work in this area has been to try to flesh out this font to the point where not only are all ligatures and small caps available, but matching math fonts and some others specialty fonts are available too.

## A Meta-font Monotype Modern?

My first approach was reasonable, but misguided. I reasoned that if I carefully measured the dimensions of the anatomy of letters in a Modern font, and then put those values in a METAFONT parameter file,

This is a test of the T<sub>E</sub>X typesetting system. Abcd Efgh Ijkl Mnop Qrst Uvwx Yzab; Cdeg Ghij Klmn Opqr Stuv Wxyz. (0123456789!)

Nàive garçõñ.

The quick brown fox jumps over the lazy cow. Now is the time for all good men to come to the aid of the party.

Efficient effluent effectively fights flightiness.

Do you see this Ring?

    'T is Rome-work, made to match

(By Castellani's imitative craft)

Etrurian circlets found, some happy morn,

After a dropping April; found alive

Spark-like 'mid unearthed slope-side figtree-roots

That roof old tombs at Chiusi; soft, you see,

Yet crisp as jewel-cutting. There's one trick,

(Craftsmen instruct me) one approved device

And but one, fits such slivers of pure gold

As this was, such mere oozings from the mine,

Virgin as oval tawny pendent tear

At beehive-edge when ripened combs o'erfow,

Since hammer needs must widen out the round,

And file emboss it fine with lily-flowers,

Ere the stuff grow a ring-thing right to wear.

(Browning, *The Ring and the Book*)

**Figure 1:** An extended selection set in Monotype Modern.

This is a test of the T<sub>E</sub>X typesetting system. Abcd Efgh Ijkl Mnop Qrst Uvwx Yzab; Cdeg Ghij Klmn Opqr Stuv Wxyz. (0123456789!)

Nàive garçõñ.

The quick brown fox jumps over the lazy cow. Now is the time for all good men to come to the aid of the party.

Efficient effluent effectively fights flightiness.

**Figure 2:** Monotype Modern plus meta-ligatures in one virtual font.

then METAFONT could simply generate a variant of Computer Modern which would essentially “be” Monotype Modern. This approach appealed to me for a variety of reasons. First of all, it was an excuse to mess around with METAFONT, one of the neater computer languages around! Second, I looked forward to being able to “discover” in this way a new and exciting typeface that lay hidden in the space of all possible values of parameter values.

Although this approach remained my meat-and-potatoes approach for this project, it was far from satisfactory. Basically, the letter forms of the Monotype font were often different in non-trivial ways than their meta-font cousins. The strokes were different in kind. For example, many thin strokes in CM (Computer Modern) are modelled by a curved pen stroke: . However, the corresponding part of a MM (Monotype Modern) font did not have the feel of a stroke: . There’s no reason why METAFONT’s `stroke` commands couldn’t mimic this kind of form, but that’s not how the CM fonts have been set up.

I had looked forward to a meta-Monotype Modern font, but *c’est la vie*. Really, though, such a font would have been overkill—after all, Monotype’s own Modern font is perfectly good enough—or is it? Could I create proper sets of ligatures and small caps in the closest Meta-equivalent to MM, to create a virtual font which would combine the meta-ligatures and small caps to the Monotype glyphs, and to see how all this looks?

I decided that measuring the parameters of Monotype Modern and inserting them into a METAFONT parameter file would give me the “best” approximation to MM possible from the existing CM-METAFONT program files. Measuring was easy, but a bit tedious. You can easily do this by creating a document with the MM font at some humongous design size, say 500-pt. Then, you view it under Ghostscript. It’s a little-known fact, but under the ‘Edit’ pull-down menu, there’s an option called ‘Measure’. If you click on this option, you awaken the utility that allows you to measure distances on screen between a click of the mouse and any other position. Thus, it’s easy enough to measure the 61 parameters that go into a meta-font. The resulting font, though, didn’t look much like Monotype Modern—it still bore the look and feel of a CM font.

It was my hope, though, that the double-f ligatures, the oldstyle numerals, and the small caps generated in this manner were visually compatible with MM. It was relatively easy to cobble together a virtual font that incorporates the glyphs from MM

THIS IS A TEST OF THE T<sub>E</sub>X TYPE-  
SETTING SYSTEM. ABCD EFGH IJKL  
MNOP QRST UVWX YZAB; CDEG GHIJ  
KLMN OPQR STUV WXYZ. (01 23 456  
789!) (THIS IS THE FAKE SMALL CAPS  
FONT.)

THIS IS A TEST OF THE T<sub>E</sub>X  
TYPESETTING SYSTEM. ABCD EFGH  
IJKL MNOP QRST UVWX YZAB; CDEG  
GHIJ KLMN OPQR STUV WXYZ. (01  
23 456 789!) (THIS IS THE MM PLUS  
METAFONT-MODIFIED SMALL CAPS FONT.)

**Figure 3:** Comparison viewing—fake small caps (top) versus the real thing (bottom).

together with the METAFONT ligatures. A specimen from this font appears in figure 2.

Many people think that SMALL CAPS fonts are easy to create—simply use scaled-down uppercase letters as the lowercase counterparts in a small-caps font. This is not a bad approximation to such a font, but the proportions are not quite right, and any hard-core T<sub>E</sub>X user rejects this solution! In figure 3, you see the comparison between a MM fake small caps font, and a “real” small caps font, created with uppercase characters drawn from MM and lowercase letters drawn from a METAFONT approximation to MM-small caps. You should see that the second example is better.

**Bold-face** I’ve always found the decision to use `cmbx` as the workhorse **bold** font in T<sub>E</sub>X to be a mysterious one. To my mind, this font is far too in-your-face. Moreover, the bold face in all Modern, non-T<sub>E</sub>X contexts never resembles these fonts, but rather the bold font that’s part of the MM suite of fonts, which closely resembles `cmb10` (at least at a 10-pt design size).

In figure 4, we see the virtual, improved version of MM Bold. The strategy is the same: we begin with font `mmb10`, the closest approximation we can make to MM Bold, and then we “virtualize” a new bold font that contains the `mmb10` ligatures with the MM Bold glyphs.

**Italic** Italic follows the same pattern. We create a font `mmi10`, a close approximation to MM Italic, and then we form the appropriate virtual font. This font appears in figure 5. To my eyes, this is the least successful experiment of all the ones I describe in this article, for the CM curves are somehow less pleasing than the MM curves.

This is a test of a modified bold font. This is a test of the T<sub>E</sub>X typesetting system. Abcd Efgh Ijkl Mnop Qrst Uvwx Yzab; Cdeg Ghij Klmn Opqr Stuv Wxyz. (0123456789!)

Nàive garçõñ.

The quick brown fox jumps over the lazy cow. Now is the time for all good men to come to the aid of the party. Efficient effluent effectively fights flightiness.

**Bold with roman.** This is **bold** together with Modernized Roman.

**Figure 4:** Testing a modified bold font.

*This is a test of mcmri7t, a modified italic font. This is a test of the T<sub>E</sub>X typesetting system. Abcd Efgh Ijkl Mnop Qrst Uvwx Yzab; Cdeg Ghij Klmn Opqr Stuv Wxyz. (0123456789!)*

*Nàive garçõñ.*

*The quick brown fox jumps over the lazy cow. Now is the time for all good men to come to the aid of the party. Efficient effluent effectively fights flightiness.*

**Figure 5:** Testing a modified italic font.

**Sans-serif and typewriter fonts** For some reason, a matching sans-serif to Monotype Modern was difficult to conjure up. In the course of my experiments, I stumbled upon one version, a sans serif of figure 6 that has an interesting 1920’s *art deco* flavor to it that would not be out of place for the title of an Agatha Christie reprint or the cover of the *New Yorker* magazine. Close scrutiny shows some strokes of some types that need fixing; these are high on my to-do list. The best I could do for a more normal sans serif appears in figure 7.

I don’t quite know how to characterize the quirkiness of the `mmtt10` font, although I personally like it. Take a look for yourself at figure 8 to decide how to characterize it. The proportionally greater

This is a test of the T<sub>E</sub>X typesetting system. Abcd Efgh Ijkl Mnop Qrst Uvwx Yzab; Cdeg Ghij Klmn Opqr Stuv Wxyz. (0123456789!)

Nàive garçõñ.

The quick brown fox jumps over the lazy cow. Now is the time for all good men to come to the aid of the party. Efficient effluent effectively fights flightiness.

**Figure 6:** Modernized Modern ‘art deco’ sans serif.

Here’s some Roman. This is a test of the T<sub>E</sub>X typesetting system. Abcd Efgh Ijkl Mnop Qrst Uvwx Yzab; Cdeg Ghij Klmn Opqr Stuv Wxyz. (0123456789!)

Nàive garçõñ.

The quick brown fox jumps over the lazy cow. Now is the time for all good men to come to the aid of the party. Efficient effluent effectively fights flightiness. Here’s some Roman.

**Figure 7:** Modernized Modern sans serif.

depth of the descending characters is the strongest contributor to this slight idiosyncrasy.

### Math fonts

In addition to your choice of text fonts, you need three kinds of special fonts in order to typeset mathematics properly. The first is the extension font, `cmex10`, and I decided to use this unchanged for “`mmex10`.” In the same way, we need special symbol fonts, this time at three sizes, and I’ve appropriated them unchanged from CM. These have the names `cmsy10`, `cmsy7`, and `cmsy5`.

We also need three sizes of special math italic fonts. Although with other font families, we have to somehow adapt the text italic for math, it’s nice here to be able to adapt the *cmmi* fonts. That’s because Knuth paid close attention to certain details, such as a special math italic *a* (compare the text italic *a*) that you just don’t see in these other fonts. I adapted them simply by adjusting the gross parameters—figure height, cap height, x-height, and so on—but leaving other parameters alone. The resulting math italics don’t quite match the text font, but perhaps they shouldn’t. After all, math *is* different from text. Anyway, the match is close enough. A sample appears in figure 9, and I hope you agree.

### Installing and using the fonts

Notice: **Whenever you install new files or fonts, don’t fail to update or refresh the filename database used by your version of T<sub>E</sub>X!** This filename updating is a consequence of Kpathsea’s file searching mechanism which has become an integral component of most modern T<sub>E</sub>X’s, or so I believe.

Let me emphasize that these fonts are no good without the Monotype Modern fonts in hand. You need them—you need to *buy* them, plus you need the adjunct files (`.tfm`, `.vf`, `.fd`, etc. files) to make them usable by T<sub>E</sub>X. These additional files

should be available for downloading from CTAN; the fontname family name is `mno`.

The fonts here require ten METAFONT source files. These files have names like `mnr10.mf`, and should all be placed in a directory like

```
<texmf>/fonts/source/public/modern
```

In addition, place all `.tfm` and `.vf` font files in your T<sub>E</sub>X system in places like

```
<texmf>/fonts/tfm/public/modern
```

and

```
<texmf>/fonts/vf/public/modern.
```

Finally, put the files `moderniz.tex` and `moderniz.sty` in

```
<texmf>/tex/plain/public/modern
```

and

```
<texmf>/tex/latex/public/modern.
```

**Plain documents** To create documents in plain T<sub>E</sub>X, place the statement

```
\input moderniz
```

at the beginning of your source file. Thereafter, nicknames like `\it` and `\bf` use the modernized fonts. Note that this macro file provides two new commands, `\sc` and `\sans` for small caps and sans serif. To use the art deco font, place a statement like

```
\font\deco=mmdeco10
```

in your source, after which `\deco` invokes this font.

**L<sup>A</sup>T<sub>E</sub>X documents** Place the incantation

```
\usepackage{moderniz}
```

following the `\documentclass` command. In the document, commands like `\rm`, `\bf`, `\it`, `\sc`, and `\tt` select modernized fonts. To use sans serif fonts, select font family `mmss` with the medium and normal series and shape:

```
\fontfamily{mmss}\fontseries{m}
\fontshape{n}\selectfont
```

To use the art deco font, use the `de` fontshape:

Here's some Roman. This is the slightly quirky typewriter font that is part of the Modernized Computer Modern family. This is a test of the T<sub>E</sub>X typesetting system. Abcd Efgh Ijkl Mnop Qrst Uvwx Yzab; Cdeg Ghij Klmn Opqr Stuv Wxyz. (01 23 456 789!)

Nàive garçõñ.

The quick brown fox jumps over the lazy cow. Now is the time for all good men to come to the aid of the party. Efficient effluent effectively fights flightiness. Here's some Roman.

**Figure 8:** Modernized Modern typewriter.

This is a test of the Modernized Computer Modern fonts. Let's do math:  $\kappa = \int_0^{\frac{\pi}{2}} e^{-x^2} dx = \frac{\sqrt{2}}{\phi!}$ . This doesn't look so bad.

Let's do some display math!

$$\kappa_{54321} = \int_0^{\frac{\pi}{2}} e^{-x^2} dx = \frac{\sqrt{2}}{\phi!} \quad f'(x) = \frac{df}{dx} = \sum_{n=0}^{\infty} \frac{1}{x^n}$$

That's all, folks!

**Figure 9:** A snippet of modernized math, together with some text fonts.

```
\fontfamily{mmss}\fontseries{m}
\fontshape{de}\selectfont
```

The normal math delimiters work to select the modernized math fonts.

### Conclusion

This package can be obtained from CTAN. Alas, as of this writing, the exact folder there is not known. Please report all bugs, suggestions, or comments to the author at

[ahoenig@suffolk.lib.ny.us](mailto:ahoenig@suffolk.lib.ny.us)

Thanks!

# The T<sub>E</sub>X Font Panel

Nelson H. F. Beebe (chair)

Center for Scientific Computing

University of Utah

Department of Mathematics, 322 INSCC

155 S 1400 E RM 233

Salt Lake City, UT 84112-0090

USA

Email: [beebe@math.utah.edu](mailto:beebe@math.utah.edu), [beebe@acm.org](mailto:beebe@acm.org), [beebe@computer.org](mailto:beebe@computer.org), [beebe@ieee.org](mailto:beebe@ieee.org) (Internet)

URL: <http://www.math.utah.edu/~beebe>

Telephone: +1 801 581 5254

FAX: +1 801 585 1640, +1 801 581 4148

## Introduction

The TUG'2001 Font Panel convened on Thursday, August 16, 2001, with members William Adams, Nelson H. F. Beebe (chair), Barbara Beeton, Hans Hagen, Alan Hoenig, and Ross Moore, with active participation by several attendees in the audience. The list of topics that was projected on the screen makes up the sectional headings in what follows, and the topics are largely independent.

Any errors or omissions in this article are solely the fault of the panel chair.

## Unicode

The work of the Unicode Consortium, begun in 1988, and first reported on for the T<sub>E</sub>X community in a *TUGboat* article [9], has reached version 3.0 of the Unicode Standard [29]. Version 3.1 appeared about the time of the TUG'2001 conference, and version 3.1.1 shortly thereafter. Unicode is a proper subset of the ISO/IEC 10646 Universal Character Set Standard [14], but publication of the latter lags.

Unicode defines a character set that is intended ultimately to cover all of the world's writing systems. Its first 128 entries are identical to the ASCII character set (dating from 1964) used by most of the world's computers.

There is a very active Unicode technical discussion e-mail list: send subscription requests to [unicode-request@unicode.org](mailto:unicode-request@unicode.org). The list is archived at <http://www.unicode.org/mail-arch/>.

Unicode conferences are held twice a year, with the twentieth in late January 2002; see <http://www.math.utah.edu/pub/tex/bib/index-table-u.html#unicode> for a bibliography of publications about Unicode.

Since most programming languages, operating systems, file systems, and even computer I/O and CPU chips, have character knowledge designed into

them, changing the character set has huge ramifications for the computing industry and for worldwide business data processing, data exchange, and record keeping.

Fortunately, a particular encoding scheme called UTF-8 makes it possible for files encoded in pure ASCII to also be Unicode in UTF-8 encoding, easing the transition to the new character set.

Up to version 2.0 in 1996, the Unicode character repertoire could be fit into a table of  $2^{16} = 65\,536$  entries. Version 3.0 in 2000 increased the count to over a million, although just under 50 000 are assigned and tabulated in the book. Version 3.2 in 2002 has just over 95 000 assigned. Consortium members hold the view that 20 or 21 bits per character (just over two million) may ultimately be necessary by the time all historical scripts have been covered.

Despite the Consortium's warning that the collection was expected to grow, several vendors did not pay attention, and prematurely adopted 16-bit entities to hold Unicode characters.

Thus, the C language data type, `wchar_t`, introduced in 1989 Standard C [7, 13, 28], is implemented as a 16-bit unsigned integer in many C and C++ compilers, with a companion function library that also has this limitation.

Even worse, the popular Java programming language is defined in terms of an underlying virtual machine [23, 24], already implemented in hardware, whose instructions are permanently designed for 16-bit characters.

These 16-bit limitations can be overcome by representation of Unicode values with variable numbers of bytes, as was done with the UTF-8 encoding. Unfortunately, the opportunity to simplify character processing significantly by having fixed-size units is tragically lost.

In the panel chair's view, these design errors will rank with the infamous ASCII/EBCDIC split

in 1964, with IBM System/360 adopting EBCDIC, and everyone else (by about 1980) adopting ASCII, with enormous economic costs, and user confusion, that lasted for decades.

Newer operating systems are already designed to use Unicode as the native character set, and vendors of older ones are migrating in that direction through UTF-8 encoding.

Of course, jumping from a 256-character set to one with potentially millions of characters poses an almost impossible problem for font vendors. It will be a very long time before the Unicode font repertoire is adequate. Current systems with native Unicode support generally provide only a subset of characters, and then sometimes only in low-resolution screen bitmaps. Bitstream for a while offered their Cyberbit Unicode font, but in July 2001, withdrew it without explanation.

Thanks to fine work by fellow TUG members Yannis Haralambous and John Plaice [26], T<sub>E</sub>X has been extended to fully support Unicode. Their system is known as  $\Omega$  (Omega), and it has been available on the annual T<sub>E</sub>X Live CD-ROM distributions since at least version 5 in 2000. Development has not been as rapid as end users might like, but it must be understood that this is a hugely complex problem, and the  $\Omega$  designers have been proceeding very carefully, cognizant of other T<sub>E</sub>X developments such as pdfT<sub>E</sub>X,  $\varepsilon$ -T<sub>E</sub>X, and  $\mathcal{N}\mathcal{T}\mathcal{S}$ , in addition to the evolution of the Unicode Standards.

### Mathematics fonts

Fonts for mathematics are a substantial problem, because, among the more than twenty thousand fonts on the market, only a handful have a remotely adequate repertoire of mathematical glyphs. These fonts are almost the only choices: *Computer Concrete*, *Computer Modern*, *Informal Math*, *Lucida*, *MathTime*, *PA Math*, *PX*, *Palatino Math*, *Pandora*, and *TX*.

While it is, of course, possible to use an existing mathematics font with any other text font, the results are rarely visually successful. For some careful studies of this, see Hoening's book [11, Chapter 10].

### Font subsetting

DVI drivers for virtually all devices, other than PostScript, subset the fonts that they include in their output streams: descriptions of unused characters are simply omitted.

Doing this for PostScript Type 1 outline fonts has proved considerably more troublesome. These fonts are generally encrypted, but Adobe has published the encryption algorithm and

keys, so software like `tldisasm` (from Lee Hetherington's and Eddie Kohler's `t1utils` package, available at <ftp://ctan.tug.org/tex-archive/fonts/utilities/t1utils>) can readily disassemble a font.

Disassembly reveals essentially a table of *numbered* (not named) subroutines, `Subrs`, each containing positioning commands, and calls to other subroutines, plus a table of character definitions, `CharStrings`, indexed by character name. Each entry of `CharStrings` also consists of positioning and drawing commands, and calls to the numbered subroutines.

Because subroutine numbers could be constructed dynamically, it is in general not possible to identify which of the numbered subroutines can be omitted, but a DVI driver could drop unused entries from the `CharStrings` table. This is transparent to font rendering software, since the entries are named, rather than numbered.

It was reported by a reviewer that computation of subroutine numbers is in practice not done in existing Type 1 and Type 2 Compact Font Format (CFF) fonts, so perhaps it is safe to drop subroutines that are not explicitly called.

Recent versions of Tom Rokicki's `dvips` driver are capable of subsetting PostScript Type 1 outline fonts, as can Adobe Acrobat Distiller and Ghostscript's `ps2pdf`.

However, this subsetting introduces new problems. What if the DVI file also included PostScript figures which themselves used fonts? Subsetting might remove characters needed by those figures.

It is infeasible, or unreliable, for the DVI driver to attempt to examine an included figure file to determine its font requirements, because far too many PostScript producing programs fail to conform to Adobe's Document Structuring Conventions that would otherwise clearly, and simply, record the file's font needs. Those conventions are clearly described in the first two editions of the PostScript Language Reference Manual [1, Appendix C] [3, Appendix G], but were ominously dropped from the third edition [6]. They are, however, documented at the Adobe Web site among the technical notes collected at <http://partners.adobe.com/asn/developer/technotes/postscript.html>, in the file [http://partners.adobe.com/asn/developer/pdfs/tn/5001.DSC\\_Spec.pdf](http://partners.adobe.com/asn/developer/pdfs/tn/5001.DSC_Spec.pdf).

Each Type 1 font contains a special 24-bit (0...16 777 215) unsigned number, the `UniqueID`, which is intended to allow printing devices to cache bitmaps of rendered fonts between jobs. A million of these numbers are reserved for private use, and

the rest are allocated to font vendors on request. A subsetted font is a *different* font, because it lacks some characters, and so must be assigned a UniqueID from the private use area. A random choice from this area would mean a one-in-a-million chance of confusion between fonts in a printer.

Regrettably, several versions of Adobe's own Acrobat Distiller, and most versions of Ghostscript (until the panel chair, who is a long-time Ghostscript beta tester, reported the problem) use a fixed UniqueID and fixed name for subsetted fonts! Happily, versions of Ghostscript released in 2001 no longer have this problem.

This non-UniqueID and fixed fontname problem fouls up more than just printers. It has been a huge headache in the U.S. National Science Foundation FastLane grant proposal project, started about 1998 to speed up, and regularize, proposal submission.

FastLane is a clear case of technology being adopted before its time. Had NSF required submission of a single PDF file for the entire proposal, or not refused to accept documents without font subsetting, the UniqueID and fontname issue would never have been noticed.

Unfortunately, they instead require submission of multiple PDF files, with subsetted and embedded fonts. These files are then merged into one PDF file for the entire proposal, then distributed back to the submitter for printing and verification, and sent electronically to proposal reviewers.

Because of the non-UniqueIDs and fixed fontnames, the software that does the merging gets hopelessly confused, and fails to produce correct output. The proposal submitter is then held to be at fault. The panel chair has spent an inordinate amount of time working with colleagues, and remotely, with NSF staff and administrators, to overcome these problems. After three years of user complaints, NSF has finally relaxed their draconian requirements, and now accepts DVI and PostScript files as well.

### Font substitution

A common problem when documents are distributed is that the required fonts may be missing at the end-user site, and because of font licensing, it may not be possible to include the fonts with the documents.

World-Wide Web browsers usually just ignore requests for missing fonts, falling back to a default font. Adobe Acrobat Reader goes further: it uses the original font metrics embedded in the PDF file, and then substitutes the missing font with another. T<sub>E</sub>X DVI drivers usually complain about missing fonts, but some will then provide a substitute, and

some may even support a user-defined font substitution file.

The PANOSE system [8] is a font classification system that assigns numeric values in 0...15 for ten font attributes (family, serif style, weight, proportion, ...). The system is further described at <http://www.w3.org/Fonts/Panose/pan2.html>, <http://www.w3.org/Printing/stevahn.html>, and [http://www.agfamotype.com/print\\_manu/pan1.htm](http://www.agfamotype.com/print_manu/pan1.htm).

Hewlett-Packard now owns the PANOSE technology, but has proposed it for open international standardization. They recommend that font files be augmented with a PANOSE number that can be used by matching software to find the closest available font. In practice, this classification seems not to have been done: in a scan of more than 8700 font files on my system, I found only two that had an embedded PANOSE number.

Nevertheless, since a lot of work has been done in the PANOSE system to identify characteristic properties of fonts, those features should be carefully considered when new font software is written.

### Font licensing

Noted font designer Chuck Bigelow thoroughly discussed the issue of typeface copyright issues in an article reprinted in *TUGboat* several years ago [10].

With widespread sharing of PostScript and PDF files on the Internet, it is difficult for font vendors to enforce their licenses. Some vendors, including Adobe, take the view that a subsetted font is sufficiently crippled that it is unlikely to be of interest to font miners, and explicitly permit distribution of documents containing subsetted fonts, while forbidding distribution without subsetting. Others are more restrictive: Bitstream does not permit free distribution of their font metric files, and historically, Autologic would not even reveal font metrics to its own font licensees!

Users can always achieve font subsetting for PostScript files from any source by the simple conversion path PostScript → PDF → PostScript, *provided* that Distiller or ps2pdf options have been chosen to turn on subsetting. Regrettably, few Web site owners have the sophistication, or conscience, to do this, and the conversion programs do not subset fonts by default. Versions of ps2pdf after May 2000 subset by default.

More troublesome is the issue of patents on algorithms and file formats, and the U.S. Patent Office, in particular, continues to issue software patents on ideas that are utterly obvious, even to people who have never used a computer. The X

Window System backing store patent held by AT&T essentially says “if you cannot store data here, put it there instead, but only if you license the right to do so from us first”. The European Union, followed by the U.S. Government, has retroactively extended patent and copyright lifetimes, exacerbating the problem.

Adobe has always been very open about publishing specifications of PostScript, PDF, and font formats, allowing anyone to implement them. They copyright their tradenames and license their software (see <http://partners.adobe.com/asn/developer/legalnotices.html>), and by virtue of being both specifiers and implementors, have a head start on their competition. However, they do not interfere with, or discourage, competitors. As a result, several printer vendors have shipped models with non-Adobe PostScript interpreters, and Aladdin and GNU Ghostscript have helped to make PostScript and PDF support almost universally accessible.

TrueType, and possibly OpenType, are covered by Apple patents that restrict what rendering software can do to display the fonts. This is highly unfortunate, and some people may react by refusing to use such fonts. Their widespread use on common desktop operating systems makes it difficult to avoid them, however. For further discussion, see the FreeType Web site, <http://www.freetype.org/patents.html>.

### Font index

Tens of thousands of fonts are available commercially, and hundreds are typically installed on each desktop computer. Sadly, it sometimes takes an expert to hunt down a particular font to make it usable in a particular application.

Font files often have filenames that are not obviously related to font names. For example, BaskervilleMT-BoldItalic might be found in files with names containing `mbvbi8a` or `basbi_`. Designers of deficient file systems with drastic filename length limitations are partly responsible, but some platforms complicate things by concealing fonts in ‘resources’ or ‘registries’, or by converting standard font formats to proprietary internal ones (without offering the reverse conversion).

The problem of mapping font names to filenames has led software designers to invent several different, and mutually incompatible, map file formats, adding to system administrator burdens. Later on the same day that this was written, traffic on the Ghostscript developers list discussed precisely this problem, with respect to new Apple operating systems; it evidently remains an inadequately solved problem!

Font licensing means that few font files are visible to Internet Web search engines, so your hunt for VanDijkBoldPlain might be almost fruitless.<sup>1</sup> While this sentence was being written, a search on <http://www.dogpile.com/>, a site that searches several other search engines, found only one hit from twelve of them. That hit was to the font index of more than 20 000 font names created by, and maintained by, the panel chair at <http://www.math.utah.edu/~beebe/fonts>.

That font index is vendor neutral: each listed font has a vendor name, linked to a vendor page at the index site that gives company name, address, other contact information, and links to the vendor Web site.

The index is maintained as rigorously validated and prettyprinted HTML files, all in a standard format, and all derived from tables, catalogs, CD-ROM contents, and other resources. The index maintainer has no commercial interest in the font industry, and the index may be mirrored by anyone.

### Multiple Master fonts

In 1992, Adobe introduced the concept of ‘Multiple Master fonts’ [4, 5]. These are fonts with one to four user-adjustable parameters that can be tweaked to change character shapes.

Knuth’s METAFONT system is much more general: the Computer Modern fonts have 62 parameters [20, pp. 10–11] whose variation can lead from fixed-width `typewriter` fonts to proportionally-spaced `bold`, `italic`, `roman`, `sans serif`, `slanted`, `SMALL CAPS`, . . . The design of such ‘meta’ fonts is truly a landmark of human ingenuity.

Sadly, it appears that Adobe has withdrawn support for Multiple Master fonts: their documentation has been removed (fortunately, the panel chair has an archived copy), and the fonts may no longer be available, although several are still listed at [http://www.adobe.com/type/browser/C/C\\_4e.html](http://www.adobe.com/type/browser/C/C_4e.html). It has also been reported that the metric files for these fonts were not included in font packages sold by Adobe, and are hard to find.

In the hands of expert programmers, the capabilities of Multiple Master fonts could have been made easily available to the masses, but except for one experiment with Illustrator [12], the opportunity was never taken advantage of, and has now been removed. Such technological shortsightedness, probably influenced by quarterly bean-counting reports, is deplorable.

<sup>1</sup> Curiously, if you search for the logical conjunction of the four parts separately, you may be luckier!

Perhaps the OpenType effort (see <http://www.opentype.org/>) will revive this possibility. In the meantime, some clever programmers could do wonderful things for the font world by creating a fancy interface to METAFONT!

The `t1utils` package mentioned above can handle Multiple Master fonts, and according to <http://developer.apple.com/technotes/tn/tn2029.html>, Apple MacOS X has added new printer support for such fonts.

Eddie Kohler's `mminstance` package, available at <http://www.lcdf.org/type/>, includes `mmafm`, for creating an Adobe Font Metric (AFM) file, and `mmpfb`, for creating a normal Type 1 font file, from a particular instance of a Multiple Master font. These tools have been ported to Microsoft Windows, and are expected to be in the next edition of the  $\TeX$  Live CD-ROM.

### Font file handling problems

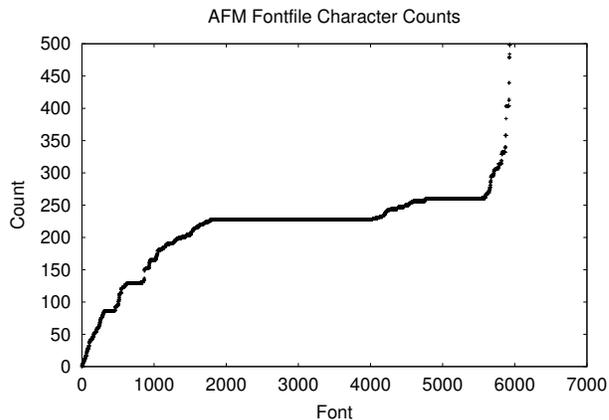
The format of Type 1 font files is described in a small book [2]. Unfortunately, software implementors have not always followed that specification.

Adobe's own Type Manager (ATM), which renders Type 1 fonts for screen display on single-user desktop operating systems, is known to be sensitive to the precise formatting of font files, and fails on some third-party fonts that otherwise conform to the specification, and are handled correctly by PostScript interpreters in printers. Adobe Illustrator may exhibit similar problems.

Recent Adobe products, such as Illustrator 7.0 and later, use a new font-rendering engine, called CoolType, instead of Adobe Type Manager. CoolType tries to reduce font menu clutter, and ends up treating `cmr10` and `cmr12` as the same font. It also has trouble with subsetted fonts with encoding arrays.

The problems with ATM and CoolType are clear cases where Adobe software developers are not adhering to their own published font specifications, and users should complain bitterly. Fortunately, clever people, notably Tom Kacvinski, in the  $\TeX$  community were able to react quickly and find work-arounds.

Some desktop software products fail to handle Type 1 fonts that contain characters in slots  $0 \dots 31$  or  $128 \dots 159$ . The lower 32 slots are vacant in all of Adobe's standard encodings defined in Appendix E of the PostScript Language Reference Manuals [3, 6]; the other 32 are partly used in advanced font encodings, but not in the default StandardEncoding used by most Type 1 fonts. This problem affects most  $\TeX$  fonts, which tend to contain either 128 or 256 characters.



**Figure 1:** Character counts in 6021 Type 1 fonts, ordered by increasing counts. The largest font in this collection contained 26,304 Chinese characters. A total of 1269 fonts (21%), have more than 256 characters, but only 39 (0.6%) have more than 1000.

Type 1 fonts are based on a named collection of characters, and those names are then assembled into a 256-element ‘encoding vector’ to provide character access using 8-bit bytes as indexes into the encoding vector. Many fonts have more than 256 characters, as shown in Figure 1, although 8-bit bytes in character strings limit access to 256 in any one instantiation of the font.

Recently, we have encountered Type 1 fonts designed by people who have completely failed to understand the importance of encoding vectors and named characters. These fonts have an encoding vector with the Adobe StandardEncoding names, but the characters with those names have shapes that bear no relation to their names! This makes constructing an encoding vector to use them exceedingly painful and error prone.

The Apple MacOS operating system does not directly use fonts in Type 1 format; instead, it converts them to a proprietary internal format. While this conversion is quite well debugged, it is another possible point of failure. It means that software ported from other platforms must be rewritten to use the MacOS font format. It may also be nontrivial to export fonts from MacOS to other operating systems. Observation of MacOS since its introduction in 1984 clearly shows that its design discourages software developers from porting to it. This may change rapidly once the new UNIX-based MacOS X becomes widely deployed.

## GNU Project and Aladdin font-related activities

Work on the GNU font utilities (<ftp://ftp.gnu.org/pub/gnu/fontutils>) ceased from 1993 to 1998, but a lot of changes were logged for 2000. Much more could be done; can you volunteer?

The already-noted FreeType Project is not part of the GNU Project, but some of the FreeType people involved work in both, and the goals are much the same: freely-distributable software of wide utility.

Although only delayed releases of Aladdin Software's Ghostscript fall into the GNU archive, it is worth noting here that in the last year, Ghostscript development has moved to the bazaar model [27], but an expert chief architect maintains final control over quality, and the developers list is quite active.

Just a month before the TUG'2001 Conference, the first release of ghostpcl was made (see <http://www.artifex.com/downloads/>). This builds on the Ghostscript source tree, but provides instead, for the first time, a portable and distributable interpreter for Hewlett-Packard's Printer Command Language, PCL. Many low-cost desktop printers, and most laser printers, recognize PCL, even if they lack PostScript support.

The lack of PCL screen display, until ghostpcl was released, has hindered software development of PCL tools. As ghostpcl matures from its current shaky state, PCL may become of more interest in the T<sub>E</sub>X community, and that is especially important because PCL is available to those on a low budget. Although Ghostscript can create page bitmaps for many different non-PostScript printers, the files are large, and slow to print.

## Other font developments

Some unattributed Polish programmers produced the ttf2pf package for converting TrueType fonts to Type 1 format. It is available at <ftp://ftp.gust.org.pl/TeX/GUST/contrib/BachoTeX98/ttf2pf.zip>

The CTAN archives in <ftp://ctan.tug.org/tex-archive/fonts/utilities/> collect more than 40 packages for dealing with font conversions.

Oleg Motygin's ttf2mf package, available at <ftp://ctan.tug.org/tex-archive/support/ttf2mf/>, converts TrueType fonts to METAFONT source code. It does so by invoking an operating system function on Microsoft Windows, so it runs only on that platform. Nevertheless, in announcements this year on the tex-euro mailing list, Daniel Taupin has demonstrated that ttf2mf can be used quite effectively to convert such fonts, making them accessible to the majority of T<sub>E</sub>X

DVI drivers, which are incapable of handling the TrueType format. There are, of course, thorny license issues that must be dealt with, and ttf2mf does not handle hinting, so the METAFONT programs are not expected to produce good results at low resolution. Despite the lack of hinting, I have found that screen display of these fonts is quite acceptable.

A month after the TUG'2001 conference, Vladimir Volovich announced the release of his CM-Super font package, available in <ftp://ctan.tug.org/tex-archive/fonts/ps-type1/cm-super>. It contains a large collection (376) Type 1 representations of all the EC, TC, and LH fonts, including Cyrillic. These fonts contain characters needed for a half-dozen common L<sup>A</sup>T<sub>E</sub>X font encodings, and for Adobe's StandardEncoding. This greatly extends the set of T<sub>E</sub>X fonts that are available in Type 1 form, making them usable in many other software tools, and importantly, for labelling in illustrations.

The T<sub>E</sub>X Live 6 CD-ROM contains Scott Pakin's mf2pt1 tool for converting a subset of METAFONT to Type 1 format.

The EuroT<sub>E</sub>X 2001 proceedings are expected to contain an article by Bogusław Jackowski, Janusz M. Nowacki, and Piotr Strzelczyk about their work on METAType1, a METAPost-based engine for generating Type 1 fonts. Their software is available at <ftp://bop.eps.gda.pl/pub/metatype1>.

My personal hope is that software will eventually be developed, and made freely available across all major platforms, for automatic and reliable conversion between any of the common outline font formats. Not only would this be of considerable convenience for users, but font designers could then take advantage of features offered by a particular format, such as METAFONT's shaped pens.

## Further reading and tool pointers

Tracy's book [30] is an interesting history of type design.

Karow's books [15, 16, 17, 18] are among the few devoted entirely to practical aspects of font design, and font file data representation.

Level, Newman and Newman [21, 22] provide a convenient catalog of thousands of fonts, with samples of each.

Moye [25] describes the commercial Fontographer package (<http://www.macromedia.com/software/fontographer/>), which is one of the more widely used desktop publishing systems for font design, and font manipulation on desktop platforms.

The commercial FontLab system (<http://www.fontlab.com/>) provides tools for font design and

manipulation. It began as a program on the Atari ST, and was later ported to Microsoft Windows and Apple MacOS.

Adobe's Type 1 Font Format is described in [2]; that reference contains pointers to an electronic version of the book, and a supplement that covers later developments.

Hoening's book (in the biased view of the panel chair) is an excellent one for L<sup>A</sup>T<sub>E</sub>X and T<sub>E</sub>X users, because it is all about using fonts with T<sub>E</sub>X.

The online notes for David Kindersley's Workshop [19] make interesting reading about font design and legibility issues.

There are extensive bibliographies on fonts and related issues in the T<sub>E</sub>X Users Group bibliography archive at <http://www.math.utah.edu/pub/tex/bib>.

### Acknowledgements

Thanks go to my fellow panel members, my wife Margaret, and L. Peter Deutsch of Aladdin Software, for many helpful comments on, and historical background for, drafts of this article.

### References

- [1] Adobe Systems Incorporated. *PostScript Language Reference Manual*. Addison-Wesley, Reading, MA, USA, 1985. ISBN 0-201-10174-2. ix + 321 pp. LCCN QA76.73.P67 A33 1985.
- [2] Adobe Systems Incorporated. *Adobe Type 1 Font Format—Version 1.1*. Addison-Wesley, Reading, MA, USA, August 1990. ISBN 0-201-57044-0. iii + 103 pp. LCCN QA76.73.P67 A36 1990. US\$14.95. [http://partners.adobe.com/asn/developer/pdfs/tn/T1\\_SPEC.PDF](http://partners.adobe.com/asn/developer/pdfs/tn/T1_SPEC.PDF). See also supplement [5].
- [3] Adobe Systems Incorporated. *PostScript Language Reference Manual*. Addison-Wesley, Reading, MA, USA, second edition, 1990. ISBN 0-201-18127-4. viii + 764 pp. LCCN QA76.73.P67 P67 1990.
- [4] Adobe Systems Incorporated. Adobe Type 1 font format Multiple Master extensions. Technical note 5086, Adobe Systems Incorporated, 1585 Charleston Road, P. O. Box 7900, Mountain View, CA 94039-7900, USA, Tel: (415) 961-4400, February 14, 1992. Withdrawn and no longer available: superseded by [5].
- [5] Adobe Systems Incorporated. Type 1 font format supplement. Technical note 5015, Adobe Systems Incorporated, 1585 Charleston Road, P. O. Box 7900, Mountain View, CA 94039-7900, USA, Tel: (415) 961-4400, May 15, 1994. [http://partners.adobe.com:80/asn/developer/pdfs/tn/5015.Type1\\_Supp.pdf](http://partners.adobe.com:80/asn/developer/pdfs/tn/5015.Type1_Supp.pdf).
- [6] Adobe Systems Incorporated. *PostScript Language Reference*. Addison-Wesley, Reading, MA, USA, third edition, 1999. ISBN 0-201-37922-8. xii + 897 pp. LCCN QA76.73.P67 P67 1999. US\$49.95, CDN\$74.95. <http://partners.adobe.com/supportservice/devrelations/PDFS/TN/PLRM.pdf>; <http://partners.adobe.com/asn/developer/PDFS/TN/PLRM.pdf>. This new edition defines PostScript Language Level 3. An electronic version of the book is available at the Adobe Web site, and is also included in a CD-ROM attached to the book.
- [7] American National Standards Institute and Computer and Business Equipment Manufacturers Association and Secretariat. *American National Standard for information systems: programming language: C: ANSI X3.159-1989*. American National Standards Institute, 1430 Broadway, New York, NY 10018, USA, December 14, 1989. 219 + vi + 119 pp.
- [8] Benjamin Bauermeister. *A manual of comparative typography: the PANOSE system*. Van Nostrand Reinhold, New York, NY, USA, 1988. ISBN 0-442-21187-2. xxvii + 257 pp. LCCN Z250 .B34 1988.
- [9] Nelson Beebe. Character set encoding. *TUGboat*, 11(2):171–175, June 1990. ISSN 0896-3207.
- [10] Charles Bigelow. Notes on typeface protection. *TUGboat*, 7(3):146–151, October 1986. ISSN 0896-3207.
- [11] Alan Hoening. *T<sub>E</sub>X Unbound: L<sup>A</sup>T<sub>E</sub>X and T<sub>E</sub>X Strategies for Fonts, Graphics, & More*. Oxford University Press, Walton Street, Oxford OX2 6DP, UK, 1998. ISBN 0-19-509686-X (paperback), 0-19-509685-1 (hardcover). ix + 580 pp. LCCN Z253.4.L38H64 1997. US\$60.00 (hardcover), US\$35.00 (paperback). [http://www.oup-usa.org/gcdocs/gc\\_0195096851.html](http://www.oup-usa.org/gcdocs/gc_0195096851.html).
- [12] Joe Holt. Notes from a slave to Multiple Master fonts. *Adobe Developers Association News*, 3(2), February 1994. <http://partners.adobe.com/asn/developer/techjournal/archived/Vol3no02.pdf>.
- [13] International Organization for Standardization. *ISO/IEC 9899:1990: Programming languages — C*. International Organization for Standardization, Geneva, Switzerland, 1990.
- [14] International Organization for Standardization. *ISO/IEC 10646-1:1993, Information technology. Universal Multiple-Octet Coded Character Set (UCS). Part 1: Architecture and Basic Multilingual Plane*. International Organization for Standardization, Geneva, Switzerland,

1993. 754 pp. CHF 360. <http://www.iso.ch/cate/d18741.html>. About two dozen corrections and amendments to this Standard have been published.
- [15] Peter Karow. *Digital Formats for Typefaces*. URW Verlag, Hamburg, Germany, 1987. ISBN 3-926515-01-5. 400 pp. LCCN Z253.3 .K371 1987.
- [16] Peter Karow. *Digitale Schriften, Darstellung und Formate, Geleitwort von Hermann Zapf (Digital Fonts, Representation and Formats, Forward by Hermann Zapf)*. Springer-Verlag and URW Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc. and Hamburg, Germany, 1992. xiii + 457, with 230 illustrations pp. English translation in [17].
- [17] Peter Karow. *Digital typefaces: description and formats*. Springer-Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 1994. ISBN 0-387-56509-4. xiii + 448 pp. LCCN Z250.7 .K37 1994. Foreword by Hermann Zapf. Translated by Bill Horton from German edition [16].
- [18] Peter Karow. *Font Technology: Description and Tools*. Springer-Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 1994. ISBN 0-387-57223-6. xvi + 460 pp. US\$79.00.
- [19] David Kindersley and Lida Lopes Cardozo. David Kindersley's workshop. World-Wide Web document, 1987. <http://www.kindersleyworkshop.co.uk>.
- [20] Donald E. Knuth. *Computer Modern Typefaces*, volume E of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986. ISBN 0-201-13446-2. xv + 588 pp. LCCN Z250.8.M46 K574 1986.
- [21] Jeff Level, Bruce Newman, and Brenda Newman. *Precision type font reference guide, version 5.0*. Precision Type, Commack, NY, USA, 1995. ISBN 0-9646252-0-2. xxv + 653 pp. LCCN Z250.7 .P74 1995.
- [22] Jeff Level, Bruce Newman, and Brenda Newman. *Precision type font reference guide, version 5.0*. Hartley & Marks Publishers, Point Roberts, WA, USA and Vancouver, BC, Canada, 2000. ISBN 0-88179-182-2. xxv + 653 pp. US\$39.95, CDN\$59.95. Republication of [21].
- [23] Tim Lindholm and Frank Yellin. *The Java Virtual Machine Specification*. Addison-Wesley, Reading, MA, USA, second edition, 1999. ISBN 0-201-43294-3. xv + 473 pp. LCCN QA76.73.J38L56 1999. US\$42.95.
- [24] Jon Meyer and Troy Downing. *Java Virtual Machine*. O'Reilly & Associates, Inc., 103a Morris Street, Sebastopol, CA 95472, USA, Tel: +1 707 829 0515, and 90 Sherman Street, Cambridge, MA 02140, USA, Tel: +1 617 354 5800, February 1997. ISBN 1-56592-194-1. xxiv + 426 pp. LCCN QA76.73.J38M49 1997. US\$32.95. <http://www.ora.com/www/item/javavm.html>.
- [25] Stephen Moye. *Fontographer: type by design*. MIS:Press, New York, NY, USA, 1995. ISBN 1-55828-447-8. xi + 275 pp. LCCN 250.8.F65 M69 1995.
- [26] John Plaice and Yannis Haralambous. The latest developments in  $\Omega$ . *TUGboat*, 17(2):181–183, June 1996. ISSN 0896-3207.
- [27] Eric S. Raymond. *The cathedral and the bazaar: musings on Linux and open source by an accidental revolutionary*. O'Reilly & Associates, Inc., 103a Morris Street, Sebastopol, CA 95472, USA, Tel: +1 707 829 0515, and 90 Sherman Street, Cambridge, MA 02140, USA, Tel: +1 617 354 5800, revised edition, 2001. ISBN 0-596-00131-2, 0-596-00108-8 (paperback). xiv + 241 pp. LCCN QA76.76.O63 R397 2001.
- [28] Herbert Schildt, American National Standards Institute, International Organization for Standardization, International Electrotechnical Commission, and ISO/IEC JTC 1. *The annotated ANSI C standard: American National Standard for Programming Language C: ANSI/ISO 9899-1990*. Osborne/McGraw-Hill, Berkeley, CA, USA, 1990. ISBN 0-07-881952-0. xvi + 219 pp. LCCN QA76.73.C15S356 1990. US\$39.95.
- [29] The Unicode Consortium. *The Unicode Standard, Version 3.0*. Addison-Wesley, Reading, MA, USA, 2000. ISBN 0-201-61633-5. xxx + 1040 pp. LCCN QA268.U56 1996. US\$50.00. <http://www.unicode.org/unicode/standard/versions/>. Includes CD-ROM. The principal authors and editors of *The Unicode Standard, Version 3.0* are Joan Aliprand, Julie Allen, Joe Becker, Mark Davis, Michael Everson, Asmus Freytag, John Jenkins, Mike Ksar, Rick McGowan, Lisa Moore, Michel Suignard, and Ken Whistler.
- [30] Walter Tracy. *Letters of credit: a view of type design*. David R. Godine, Publisher and G. Fraser, Boston, MA, USA and London, UK, 1986. ISBN 0-87923-636-1 (Godine), 0-86092-085-2 (Fraser). 219 pp. LCCN Z250.A2 T73 1986. US\$27.50.

# Managing Multiple TDS Trees

Michael J Downes

American Mathematical Society, Providence, Rhode Island 02904, USA

[mjd@ams.org](mailto:mjd@ams.org)

## Abstract

The TDS specification for  $\text{\TeX}$  directory structure provides a good framework for organizing a single  $\text{\TeXMF}$  tree such as the one at the heart of Thomas Esser's  $\text{te}\text{\TeX}$  distribution. In practice, however, because of the normal processes of local changes and ongoing upgrades it is often advantageous to set up a more elaborate system of multiple TDS trees. This permits organizing the pieces that have been added above and beyond the original base tree in a way that makes it easier to cope with continuing change.

I could have more accurately indicated the scope of this article by using a longer title: *Managing multiple TDS trees in a **web2c**-based  $\text{\TeX}$  system*. But it should be fairly clear how to apply the general principles discussed here to other kinds of  $\text{\TeX}$  systems with similar file searching capabilities.

## Introduction

**$\text{\TeX}$  Directory Structure** After  $\text{\TeX}$  had been in use for some years people started to notice that the set of files needed by  $\text{\TeX}$  when processing documents was growing rather large and unwieldy. The simplest way for  $\text{\TeX}$  to look for input files was to keep them all in a single directory. But this made maintenance difficult.

In response a volunteer committee analyzed the categories of files involved and drew up a specification known as "TDS":  $\text{\TeX}$  Directory Structure. (See <http://tug.org/tds>.)

The TDS spec gives a recommended hierarchical structure for collections of  $\text{\TeX}$ -related files in a generic form that is intended to be usable and compatible across the various kinds of computers in common use nowadays: PC/Windows, Macintosh, Unix, and so forth. A **TDS tree** is a tree of directories with each major subcategory of files residing in its own subtree — fonts, macros, and documentation, to name a few.

But a TDS tree has everything branching from a single root. In practice it may often be desirable to set up more than one TDS tree, as mentioned in the TDS documentation:

```
... we shall designate the root
TDS directory by 'texmf' (for "TeX
and METAFONT"). We recommend using
that name where possible, but the
actual name of the directory is up to
the installer. On PC networks, for
```

```
example, this could map to a logical
drive specification such as 'T:'.
```

```
...
...
```

A site may choose to have more than one TDS hierarchy installed (for example, when installing an upgrade).

In fact, my experience suggests that in most cases it is useful to set up quite a few trees, branching freely whenever you find a key difference in usage or upgrade policy, as long as your  $\text{\TeX}$  system can support it. For example, consider this quote from a recent message on the  $\text{te}\text{\TeX}$  mail list:

```
(We're even using one and the same
nfs archive for three platforms,
Solaris, Linux and W2k. Only trouble
is you cannot update executables
for one platform without putting
the *@:format files in special local
places.)
```

A solution for such a problem would be to make a separate, very sparse TDS tree that only holds format files (and any other files that are tied to specific binaries). I would probably put the binaries in the same tree, if that could be done without causing extra work elsewhere.

**$\text{\TeX}$  systems: **web2c**, **te $\text{\TeX}$** , **MiK $\text{\TeX}$** , etc.**  
**Installing a  $\text{\TeX}$  system** The  $\text{\TeX}$  Users Group has a good listing of  $\text{\TeX}$  systems for various platforms at <http://www.tug.org/interest.html>. I am going to place my discussion of TDS trees within the framework of a  $\text{\TeX}$  system called  $\text{te}\text{\TeX}$ , which

is layered in turn on top of another system called **web2c**. One of the most important elements in **web2c** is a library called `kpathsea`, designed as a tool for other programs to use when they need to efficiently search a tree containing a large collection of files, and in particular to provide ways for users to specify search rules that reflect the needs of a given application. Thus, the versions of  $\TeX$ , METAFONT, `dvips`, `bibtex`, and other programs included in the **web2c** distribution are all adapted to use the `kpathsea` library.

The use of the `kpathsea` library is the critical factor that determines the exact syntax and configuration methods one uses when making arrangements to have  $\TeX$  search multiple TDS trees. Other  $\TeX$  systems such as MiK $\TeX$  or True $\TeX$  that support recursive searching through a TDS tree differ in various details, but many of the general principles will be the same.

The **web2c** guys, notably **Karl Berry & Olaf Weber** have pulled the source code for the programs  $\TeX$ , METAFONT, `dvips`, `xdvi`, into a coherent collection that compiles easily and plugs in the `kpathsea` path searching library where applicable. (Thank you, Karl and Olaf, for all your hard work!)

The  $\TeX$  guys, notably **Thomas Esser**, have built a big standard collection that includes **web2c** as a base, but (a) is designed to simplify installation for ordinary users who don't necessarily want to have all the sources and don't plan to make a practice of recompiling things; and (b) includes a rather comprehensive set of commonly needed runtime files, which makes the programs easier to use after they are installed. (Thank you, Thomas, for all your hard work!)

## Documentation

Much of the most useful documentation for **web2c** and `kpathsea` is in "Info" form, which can be read with Emacs or with the standalone `info` application. I found the following docs especially useful:

**web2c:** `web2c.info`

**kpathsea:** `kpathsea.info`

**dvips:** `dvips.info`

**xdvi:** man page for `xdvik`

**te $\TeX$ :** See `TETEXDOC.*` and `teTeX-FAQ`:

`TEXMF/doc/tetex/teTeX-FAQ`

`TEXMF/doc/tetex/TETEXDOC.dvi`

If your  $\TeX$  system is  $\TeX$ , you can use the command

```
texdoc TETEXDOC
```

to automatically find the `.dvi` file and start up a viewer (probably `xdvi`).

When I tried `texdoc teTeX-FAQ`, however, it didn't work. You may have better luck. My attempt was with  $\TeX$  1.0.

**Recursive directory searching** For programs whose file searching is based on the `kpathsea` library,

- A 'path' is the name of a directory (aka folder), which may have several components in its name, one for each level.
- In the `texmf.cnf` file the generic separator char between components of a path is slash `/`.
- In a list of multiple paths, the generic separator char between paths is semicolon `;`. (On Unix systems you would normally see colon `:` here.)
- Trailing `//` on a directory name means to search subdirectories (recursively).
- A leading `!!` means to look only in the `ls-R` database, never on the disk.
- A leading/trailing/doubled `;` in the paths will be expanded into the compile-time default path.

## Use of filename DB to speed up file searches

The `kpathsea` library supports the use of a "filename database" to speed up file searches. (This database is the referent of `texhash`, `mktextlsr`, `ls-R` files and so on.) It is a simple lookup table that gives the location of each file in a TDS tree.

In practice, using the DB seems to cause users a lot of trouble because it is easy to forget to update the DB after adding files. Because most systems nowadays are fast enough that the search time is negligible for even a fairly large collection of files, I would recommend in most cases to freeze the `TEXMFMAIN` tree and use the DB lookup for `TEXMFMAIN` but *not for any other trees*. Whether this recommendation is a good idea in your situation depends on the following general principles:

1. Use the DB for `TEXMF` trees that don't change.
2. Use the DB for `TEXMF` trees where file access is slow (e.g., across a slow network).

For example, consider the Usenet post shown in Figure 1.

**Using an alternate `texmf.cnf` file** If you want to specify a different arrangement of `TEXMF` trees than the one in the out-of-box `cnf` file (and in practice this is almost always going to be the case), you have two choices:

1. Edit the `texmf.cnf` file in the `TEXMFMAIN` tree

---

From: Peter Neergaard <turtle@cs.bu.edu>  
Subject: Re: Why does miktex need to refresh data base  
Newsgroups: comp.text.tex  
Date: 16 Jul 2001 12:24:59 -0400

On July 16, 2001, Robin Fairbairns wrote:

RF> consider the following example from my texmf.cnf file.

```
RF> TEXMF = {$HOMETEXMF,!!$TEXMFLOCAL,!!$TEXMFMAIN,!!$TEXMFEXTRA}
```

RF> the !! indicators mean that ls-R search is obligatory.  
[...]

In some cases you might furthermore need to set TEXMFDBS which specifies where tetex looks for ls-R files.

On my work, I am actually troubled by the fact that most of the file systems are NFS mounted. I did not get a decent speed of file searching before I started building ls-R files for my private hierachies.

Figure 1: Some discussion of file searching speed

---

2. Set the environment variable TEXMFCNF to point to a different location (just give the directory name, you don't need to give the full file name)

Obviously, if you do not have privileges to change files in the TEXMFMAIN tree, you will need to resort to the latter option.

To ensure that your cnf file will be read instead of the system-wide cnf file, define the environment variable TEXMFCNF in your shell startup file (.cshrc, .profile, or the equivalent):

```
# for .cshrc:  
setenv TEXMFCNF /home/abc/texmf/web2c
```

The first three lines of the standard teTeX texmf.cnf file at the present time look like this:

```
texmf.cnf -- original runtime path  
configuration file for \fn{kpathsea}.  
(If you change or delete 'original'  
on the previous line, the  
distribution won't install its  
version over yours.)
```

If you use an alternate texmf.cnf file, outside the TEXMFMAIN tree, the comment in lines two and three is irrelevant. You don't have to change anything.

**Single user setup** For a typical single user situation, I suggest three trees: MAIN, PLUS (added-public), and PRIVATE (added-private), and using

file-database lookup only for the MAIN tree, which is kept frozen. The distinction between public and private is simply this: Put a file into the PRIVATE tree if it is not publicly released somewhere on the World-Wide Web for all to use. This could include

- packages or fonts that you created yourself
- slightly modified versions of publicly available packages, that you use yourself but that are not intended for any wider use.
- packages or fonts that were sent to you privately by a colleague
- old versions of packages that are no longer available from CTAN (but if this becomes a major component of your system you will probably want to add another OLDVERSIONS tree; see the “[Old Versions](#)” section below.

So suppose you install version 1.2 of teTeX. The MAIN tree will be the TDS tree as given in the teTeX distribution, out of the box, with absolutely no changes. The PLUS TDS tree will be for any packages that you fetch from CTAN as needed, i.e., whenever you go to use a package and find that it is not present in the MAIN tree. The PRIVATE TDS tree is for any others that are not readily obtainable from the WWW.

In the texmf.cnf file, use the !! prefix (DB lookup) only for the TEXMFMAIN tree. Suppose that you have: small

---

This example shows the top of a `texmf.cnf` file as it would look if you stripped it down to bare essentials in an attempt to understand it before beginning to add more TDS trees. The standard `texmf.cnf` file that comes with `teTeX` includes sample definitions for `TEXMFLOCAL` and `VARTEXMF` as shown here, commented out.

```
% texmf.cnf -- runtime path configuration file for kpathsea.
%
% The main tree, which must be mentioned in $TEXMF, below:
TEXMFMAIN = /usr/local/teTeX/texmf

% A place for local additions to a "standard" texmf tree. For example:
%   TEXMFLOCAL = /usr/local/teTeX/texmf.local

% If defined, teTeX's texconfig stores modifications here (instead of the
% TEXMFMAIN tree).
%   VARTEXMF = /usr/local/teTeX/texmf-var
...

TEXMF = !!$TEXMFMAIN
...

% Where to look for ls-R files. There need not be an ls-R in the
% directories in this path, but if there is one, Kpathsea will use it.
% ...
TEXMFDDBS = $TEXMF
```

**Figure 2:** A minimal `texmf.cnf` file

---

```
TEXMFMAIN = /usr/local/teTeX/share/texmf
TEXMFPLUS = /usr/local/texmf/plus
TEXMFPRIVATE = /usr/local/texmf/private
(Note the absence of "teTeX" in the latter two.)
Then you would define $TEXMF as
TEXMF =
  {$TEXMFPRIVATE;$TEXMFPLUS;!!$TEXMFMAIN}
```

The capabilities of other systems in this respect can vary, of course. The impression I have from looking at the `MiKTeX` documentation is that multiple TDS trees are supported but at the present time you do not have the option of turning the database lookup on or off for different trees. If this is true then you would have to run the `MiKTeX` command that updates the database every time you add a new file, regardless of which tree it goes into. (See <http://www.miktex.org/>.)

**Multi-user setup** For a typical multi-user situation, I suggest a set of TDS trees that is basically analogous to the single-user situation but with one or two additional considerations. If your company's name is Bingle Publishing Enterprises, then use `MAIN` and `PLUS` trees as before but define a `BINGLE` tree instead of `PRIVATE`. You may also want to consider using a `VAR` tree and, possibly, an `OLD` tree.

**TEXMFMAIN** The basic TDS tree provided by `teTeX`

**TEXMFPLUS** Additional packages not found in `MAIN` but needed for Bingle work.

**TEXMFBINGLE** Additional packages needed for Bingle work but not publicly released (items created in-house or otherwise privately obtained).

**VARTEXMF** This is the tree alluded to in the `texmf.cnf` file, which is intended to hold format files and font files generated at your site. If you use this tree it is also a good place to put the Bingle version of `texmf.cnf`.

**TEXMFOLD** A tree to hold obsolete versions of packages that may still need to be available for some of your documents. Or maybe more than one tree (see [Old Versions](#)).

Then your normal definition of `TEXMF` would be

```
{$TEXMFBINGLE,$VARTEXMF,
  $TEXMFPLUS,!!$TEXMFMAIN}
```

and when processing documents that need something from the `TEXMFOLD` tree, you would change the definition of `TEXMF` in some appropriate way (it might depend on whether or not any filenames

---

```

% Earlier entries (in the same or another file) override later ones, and
% an environment variable foo overrides any texmf.cnf definition of foo.
%
% All definitions are read before anything is expanded, so you can use
% variables before they are defined.
%
% If a variable assignment is qualified with '.PROGRAM', it is ignored
% unless the current executable (last filename component of argv[0]) is
% named PROGRAM. This foo.PROGRAM construct is not recognized on the
% right-hand side. For environment variables, use FOO_PROGRAM.
%
% Which file formats use which paths for searches is described in the
% various programs' and the kpathsea documentation.
%
% // means to search subdirectories (recursively).
% A leading !! means to look only in the ls-R db, never on the disk.
% A leading/trailing/doubled ; in the paths will be expanded into the
% compile-time default. Probably not what you want.
%
% You can use brace notation, for example: /usr/local/{mytex:othertex}
% expands to /usr/local/mytex:/usr/local/othertex. Instead of the path
% separator you can use a comma: /usr/local/{mytex,othertex} also expands
% to /usr/local/mytex:/usr/local/othertex. However, the use of the comma
% instead of the path separator is deprecated.
%
% The text above assumes the path separator is a colon (:). Non-UNIX
% systems use different path separators, like the semicolon (;).

```

Figure 3: Comments from the  $\text{tE}\text{T}\text{X}$  `texmf.cnf` file

---

in `TEXMF`BINGLE shadow filenames in the `TEXM-`FOLD tree).

### Feel free to choose your preferred tree names

There is nothing sacrosanct about the TDS tree names that I have suggested. The only thing you have to worry about is whether some particular tree names get special handling in your  $\text{T}\text{E}\text{X}$  system; this is true in  $\text{tE}\text{T}\text{X}$  for `VARTEXMF` (and `VARTEX-`FONTS), but to the best of my knowledge not for anything else. So instead of `MAIN PLUS PRIVATE`, you might prefer

```
TEXMFMAIN TEXMFEXTRA TEXMFLOCAL
```

or

```
TEXMFBASE1 TEXMFBASE2 TEXMFBINGLE
```

And I can think of a few more possible names for the first auxiliary tree, such as `TEXMFADD` or `TEXMFAUX` or `TEXMFPUBLIC` or ...

### SELFAUTOPARENT or SELFAUTODIR?

If you look at a real `texmf.cnf` file you may see that the definition of `TEXMFMAIN` refers to some puzzling variables `SELFAUTOPARENT` or `SELFAUTODIR`. There are two ideas here:

1. When  $\text{T}\text{E}\text{X}$  or another `web2c` program starts up, it should be able to decide where to look for the `cnf` file by checking its own location—i.e., if the full path for the `dvips` program is `/usr/local/teTeX/bin/dvips`, then by default `dvips` will look for the `cnf` file in `/usr/local/teTeX/texmf/web2c`.
2. The full path of programs such as `dvips` may vary according to an installation option: whether to include an extra level that indicates the architecture and operating system for which the programs were compiled.

**SELFAUTODIR** If the executable for  $\text{T}\text{E}\text{X}$  is found at

```
/usr/local/teTeX/bin/tex
```

then when `tex` first starts up it sets

- `$SELFAUTOLOC = /usr/local/teTeX/bin`
- `$SELFAUTODIR = /usr/local/teTeX`
- `$SELFAUTOPARENT = /usr/local`

These affect all subsidiary searches that refer directly or indirectly to one of the `SELFAUTO` variables. Your setting of `TEXMFMAIN` should refer to `SELFAUTODIR`, e.g.,

```
TEXMFMAIN = $SELFAUTODIR/share/texmf
```

**SELFAUTOPARENT** But there is a “multiplatform” installation option for web2c, which would have multiple tex executables in places such as:

```
/usr/local/teTeX/bin/sparc-sun-solaris2.7/tex
```

Then

- `$SELFAUTOLOC = /usr/local/teTeX/bin/sparc-sun-solaris2.7`
- `$SELFAUTODIR = /usr/local/teTeX/bin`
- `$SELFAUTOPARENT = /usr/local/teTeX`

In this case your definition of `TEXMFMAIN` should refer to `SELFAUTOPARENT`.

### Using the SELFAUTO... variables is not mandatory

If it becomes too much trouble to sort all this out, you can always fall back on giving the full directory name explicitly in the `texmf.cnf` file and don’t try to make the `SELFAUTO` idea work.

```
TEXMFMAIN = /usr/local/teTeX/share/texmf
```

It won’t make any practical difference unless and until you do something such as copy the tree to another machine, and for whatever reason don’t put it in the same location. Then you would need to edit the `texmf.cnf` file to match.

### VARTEXMF? VARTEXFONTS?

The option of generating PK files on demand, which has been around for some years now, brings with it some questions about where to put the files that are generated. On a multi-user system, ordinary users cannot normally create files in the main `TEXMF` tree. If I understand correctly, the idea of setting up a `VARTEXFONTS` area for the generated files, which was the earliest approach, was later generalized to the idea of a `VARTEXMF` tree to hold several kinds of locally generated files: `pk` files, `tfm` files, `tex` format files, and configuration files.

The evolution and documentation of this idea was not particularly easy for me to follow, and I believe that if you have a `PRIVATE` or `BINGLE` tree it can serve as the location of all the files that would otherwise go into the `VARTEXMF` tree. But I have not tested this hypothesis extensively enough to rule out the possibility that the name “`VARTEXMF`” gets special treatment either by the `kpathsea` library or by some of the `teTeX` configuration scripts.

At the present time I am using a separate `VARTEXMF` tree, setting

```
VARTEXFONTS = $VARTEXMF/fonts
```

and following the related recommendations given in the `texmf.cnf` file that comes with `teTeX`.

For some related information see also the “Filesystem Hierarchy Standard” at <http://www.pathname.com/fhs/>.

### HOMETEXMF

For individual users on a multi-user system to add personal bits and pieces, there is a convention of using a variable named `$HOMETEXMF`.

1. The sysadmin might enable use of `$HOME/texmf` for everybody by defining

```
HOMETEXMF = $HOME/texmf
```

and including `$HOMETEXMF` in the definition of `$TEXMF`, in the system-wide `texmf.cnf` file. Then anything that you put in that tree immediately becomes usable for you. (Assuming that the sysadmin did not use the `!!` prefix for the `HOMETEXMF` tree.)

[One caveat: For the sysadmin herself, it might be better *not* to have `HOMETEXMF` included in `TEXMF` when doing `TeX` system administration tasks.]

2. Or if not, you can put files into `$HOME/texmf` anyway:
  - (a) Define the environment variable `TEXMFCNF = $HOME/texmf/web2c` in your shell login script. This ensures that `tex` and other programs will use your personal `cnf` file instead of the system-wide one.
  - (b) Make a small `texmf.cnf` file in that directory that defines `TEXMF` to include `HOMETEXMF`. This ensures that all other kinds of files will be found in the specified search order. You can also copy other lines from the system-wide `texmf.cnf` file if further customization is needed.

### Upgrades

**Upgrading teTeX** Suppose that you hear `teTeX` 2.0 has been released and you are eager to install it on your system to replace your current version, 1.4. What will your plan of action be?

Naturally you will want to install version 2.0 in an entirely separate tree where it can be tested for a while, so that you feel reasonably sure that `teTeX` 2.0 will produce the same results (or, possibly, better) with your existing document base.

Suppose that you currently have:

```
TEXMFMAIN = /usr/local/teTeX/texmf
```

```
VARTEXMF = /usr/local/texmf/var
```

```
TEXMFPLUS = /usr/local/texmf/plus
```

```
TEXMFBINGLE = /usr/local/texmf/bingle
```

```
TEXMF = {$TEXMFBINGLE,$VARTEXMF,
        $TEXMFPLUS,!!$TEXMFMAIN}
```

You could install the new  $\text{tE}\text{X}$  distrib at `/usr/upgrd/teTeX-2.0` (say) and for testing purposes set one environment variable:

```
TEXMFCNF=/usr/upgrd/texmf/var/web2c
```

With luck all other adaptations can be handled by editing the new `texmf.cnf` file that you make in the indicated location, primarily defining `TEXMFMAIN` and `VARTEXMF` differently (the latter to hold format files).

```
TEXMFMAIN = /usr/upgrd/teTeX/texmf
```

```
VARTEXMF = /usr/upgrd/texmf/var
```

### Old Versions

The Comprehensive  $\text{T}\text{E}\text{X}$  Archive Network does not attempt to systematically archive old versions of software. At any given time, you may see the previous version and the current version of a particular package, or the current version and a beta version of the next release.

Therefore, if you have a need to run different documents with different versions of a given package, you will need to take it on yourself to keep a copy of all the old versions that you want.

How do you arrange that a particular document gets the appropriate version of a particular package? Well, if you thought things were complicated before . . . .

If the number of different package/version combinations remains relatively small, you can use additional TDS trees for this purpose. Let's suppose, for example, that in addition to the latest release of  $\text{L}\text{A}\text{T}\text{E}\text{X}$  you would like to be able to run older releases on demand, one for each of the years from 1996 through 2000.

Set up a separate TDS tree for each release of  $\text{L}\text{A}\text{T}\text{E}\text{X}$  (here the phrase "tree farm" comes to mind).

```
/usr/local/texmf.old/latex1996
/usr/local/texmf.old/latex1997
/usr/local/texmf.old/latex1998
/usr/local/texmf.old/latex1999
/usr/local/texmf.old/latex2000
```

Make links in the  $\text{tE}\text{X}$  bin directory so that `latex1996`, etc., can be used from the command line. Then in the `texmf.cnf` file put

```
TEXMFOLD=/usr/local/texmf.old
TEXINPUTS.latex1996=
    $TEXMFOLD/latex1996/tex/latex//
    ;$TEXMF/tex/{latex,generic}//
```

and similarly for the others.

### Searching

**Examining search paths** To check what the current value of `TEXMF` is:

```
kpsexpand '$TEXMF'
```

This shows the end result after expanding all intermediate variable references.

Try `kpsexpand '$TEXMFCNF'` and look at the `texmf.cnf` file found there if you want to examine the variable settings in detail.

**Checking the  $\text{TEXINPUTS}$  search path** This gives you the generic search path:

```
kpsexpand '$TEXINPUTS'
```

But that one is seldom used in practice. Instead, one gets a format-specific value of `TEXINPUTS` from the `texmf.cnf` file. To find out what it is:

```
kpsewhich --programe=latex
--expand-var='$TEXINPUTS'
```

This is the most accurate way to check (short of turning on the debugging flags) because it takes into account any environment variables that may be set.

**Precedence of environment variables** For a program such as  $\text{L}\text{A}\text{T}\text{E}\text{X}$ , the search path is taken from the first of the following possibilities that is discovered to be non-null:

- `$TEXINPUTS_latex` (env variable)
- `$TEXINPUTS` (env variable)
- `$TEXINPUTS.latex` (cnf file)
- `$TEXINPUTS` (cnf file)

This is for a recent  $\text{tE}\text{X}$  (1.x). Some options might not be searched for earlier versions.

The simplest way of temporarily changing the search path of a given command is still to set `TEXINPUTS`. For example (if you are using `sh/bash/ksh`):

```
TEXINPUTS=$HOME/mystyles: latex xyz
```

But remember to put the extra colon on the end that indicates "append the usual value here".

**Debugging search paths** To scrutinize in utmost detail the actual search path that is used for, say, running `latex` on a particular file:

```
KPATHSEA_DEBUG=122 latex xyz.tex
```

or equivalently, for `csh` users:

```
(setenv KPATHSEA_DEBUG 122; latex xyz.tex)
```

See `kpathsea.info` for more information on the debugging flags that are available. Some programs, for example `xdvi`, may have special debugging options in addition to the general ones available through `kpathsea`. Check their documentation.

**Slash versus dot in the TDS tree farm** In the past some people have used directory names of the form `texmf.local` for auxiliary TDS trees. My hunch is that it will tend to work better in practice to make maximal use of the filesystem hierarchy, if you have enough TDS trees to call it a tree farm. So instead of

```
/ams/texmf.plus
/ams/texmf.prod
/ams/texmf.var
```

I am using

```
/ams/texmf/plus
/ams/texmf/prod
/ams/texmf/var
```

### Fallback search paths

In the standard  $\text{\TeX}$  `texmf.cnf` file the `TEXINPUTS` path for  $\text{\LaTeX}$  currently has an interesting part at the end:

```
TEXINPUTS.latex =
    .;$TEXMF/tex/{latex,generic,}//
```

This means search the following trees in order:

```
$TEXMF/tex/latex//
$TEXMF/tex/generic//
$TEXMF/tex//
```

In other words, if a file is not found in the  $\text{\LaTeX}$  tree, and not in the generic tree, go back and search the entire  $\text{\TeX}$  tree (all formats!), including searching again in the `latex` and `generic` trees.

This is helpful if you notice that some files from one format are useful for use with other formats (e.g.,  $\text{\LaTeX}$  graphics stuff or certain  $\text{\ConTeXt}$  files). But it also makes you vulnerable to other problems, and ultimately I think it is better to look for a different solution. So I would say:

- Don't put that final comma in.
- And by the way it probably should be a more generic semicolon.
- And by the way the slashes don't really come out right unless you do as I say and omit the comma. (Although `kpathsea` manages anyway by ignoring extra slashes if there are too many.)

### Expanding a path string

```
kpsewhich --expand-path=
    '$TEXMFMAIN/tex/{latex,generic}'
```

yields:

```
/usr/local/texmf/tex/latex:
/usr/local/texmf/tex/generic
```

### Recursive expansion

```
kpsewhich --expand-path=
    '$TEXMFMAIN/tex/{latex,generic}'//
```

yields all the `latex` subdirectories first, then all the `generic`:

```
/usr/local/texmf/tex/latex:
/usr/local/texmf/tex/latex/base:
/usr/local/texmf/tex/latex/config:
/usr/local/texmf/tex/latex/amsfonts:
    ...
/usr/local/texmf/tex/generic/thumbpdf:
/usr/local/texmf/tex/generic/ukrhyph
```

### Expansion with two TDS trees

```
kpsewhich --expand-path=
    '${VARTEXMF,$TEXMFMAIN}/tex/{latex,generic}'
```

yields:

```
/bingle/texmf/var/tex/latex:
/usr/local/texmf/tex/latex:
/bingle/texmf/var/tex/generic:
/usr/local/texmf/tex/generic
```

### Recap

Suppose your company name is Bingle. Here in a nutshell is a typical approach you could use.

Environment:

```
export TEXMFCNF=/usr/local/texmf/bingle/web2c
texmf.cnf file:
```

```
TEXMFMAIN = /usr/local/teTeX/texmf
TEXMFPLUS = /usr/local/texmf/plus
VARTEXMF = /var/texmf
TEXMFBINGLE = /usr/local/texmf/bingle
TEXMF = {$TEXMFBINGLE,$VARTEXMF,
        $TEXMFPLUS,!!$TEXMFMAIN}
...
TEXINPUTS.context =
```

- ```
    .;$TEXMF/tex/{context,generic}//
```
- Put all upgrades and new packages that you get from elsewhere into the `TEXMFPLUS` tree.
  - Put packages in `TEXMFBINGLE` if they are not publicly available. (This might include old versions of public packages that you want to keep.)
  - Put format files and config files in `VARTEXMF`.
  - Debug by setting `$KPATHSEA_DEBUG=122`.

### References

Most of these links are already mentioned earlier in the text, but I gather them here for convenience.

## Appendix M: Some MiKTeX notes

To give a flavor of the differences between TeX systems when it comes to search paths, I show some fragments from the MiKTeX documentation in Figures 4 and 5.

---

```
[LaTeX]
Input Dirs=.;c:\users\me//;%R\tex\latex//;%R\tex\generic//
The direct TeX equivalent would be
TEXINPUTS.latex=!/users/me//;!!$TEXMF/tex/latex//;!!$TEXMF/tex/generic//
although the latter two parts would more usually be combined using brace notation:
TEXINPUTS.latex=!/users/me//;!!$TEXMF/tex/{latex,generic}//
```

**Figure 4:** From the MiKTeX documentation: the search path for L<sup>A</sup>T<sub>E</sub>X.

---

### Which is the best directory to keep .sty files where MiKTeX can find them?

Each new L<sup>A</sup>T<sub>E</sub>X package should be installed in the folder `tex\latex\package` relative to the local TEXMF folder (usually `C:\localtexmf\`).

Example: suppose you want to install the package `thesis-ex` which consists of the style file `thesis-es.sty`: Create a new folder:

```
C:\> mkdir C:\localtexmf\tex\latex\thesis-ex
Copy all files (only one in our case) to the new folder:
C:\> copy thesis-ex.sty C:\localtexmf\tex\latex\thesis-ex
Refresh the file name database so that MiKTeX finds the new files:
C:\> initexmf -u
```

**Figure 5:** From <http://www.miktex.org/faq/index.html>

### TeX Directory Structure

- <http://www.tug.org/tds/>
- <http://www.ctan.org/tex-archive/tds/standard/tds/tds.html> (.pdf, .dvi, etc)

### Filesystem Hierarchy Standard

<http://www.pathname.com/fhs/>

### Web2c documentation

- `/usr/local/teTeX/info/web2c.info` (command-line options!)
- <http://www.tug.org/web2c/manual>

### Kpathsea documentation

- `/usr/local/teTeX/info/kpathsea.info` (search path syntax!)
- <http://www.tug.org/kpathsea/>

**teTeX:** See `TETEXDOC.*` and `teTeX-FAQ`:

```
TEXMF/doc/tetex/teTeX-FAQ
TEXMF/doc/tetex/TETEXDOC.dvi
```

**dvips:** `dvips.info`

**xdvi:** man page for `xdvik`

### Obtaining a TeX system for your computer

See <http://www.tug.org/interest.html>.

- **fpTeX** is a free system for Windows 95/98/ME/NT/2000/XP computers that is based directly on **web2c**. <http://www.fptex.org/> (Thank you, Fabrice Popineau.)
- **MiKTeX** is another one often used on Windows computers. <http://www.miktex.org/> (Thank you, Christian Schenk.)
- **Join the TeX Users Group.** Then you will get a TeX Live CD in the mail every year containing several free TeX systems, including the two above, in a ready-to-run or easy-to-install form. (This is also true for most of the other Local User Groups such as Dante, UKTUG, etc.) See <http://www.tug.org/texlive.html>.

**Appendix R: Some Miscellaneous Remarks**

- These programs use `kpathsea`: `TEX META-POST`, `BIBTEX`, `dvips` [`dvipsk`], `xdvi` [`xdvik`], and quite a few others.
- Some programs that don't use `kpathsea`: `ghostscript`, `ghostview`, `latex2html`, `info`, and `man`.
- Drawback of finely subdivided trees as recommended in the TDS spec: It's harder to do group operations across the boundaries. Advantage: It's easier to do operations on the natural subsets.
- I like to put the `.tex`, `x.tex`, etc., files from `latex` into the generic tree.
- `kpathsea` changes the generic `;` path separator into the Unix equivalent `:` when printing messages to the screen, if running on a Unix system.
- If a directory is listed in `TEXMFDBS` in the `texmf.cnf` then the disk is never searched—I think. Need to be wary of complications here.

**Appendix X: xdvi**

**xdvi search paths** The following variables are significant for `xdvi`. General setup:

`TEXMFCNF` to find `texmf.cnf` and read search path info  
`TEXMFDBS` to search for `ls-R` database files

`MIMELIBDIR` for mime types

`MAILCAPDIR` for mailcap file

For dealing with a particular font (e.g., `cmtt10`):

`VFFONTS` search for `cmtt10.vf`

`MKTEXPK` use `mktexpk script?` 0 false, 1 true

`PKFONTS` search for `cmtt10.300pk`

**Generating PK files** Here is my weak and superficial understanding of the process, in case it may be of some small help. I have not done a great deal of experimentation with PK file generation.

The config files that affect the generation of PK files are

```
...teTeX/texmf/web2c/texmf.cnf
```

and

```
...teTeX/fontname/*
```

Note that the latter is *outside of the texmf hierarchy*.

If the MF sources were found in one `texmf` tree, the PK files are written into that same `texmf` tree. If that `TEXMF` tree is not writable, the generated fonts go into `VARTEXFONTS`. The files in the `fontname` directory control which subdirectory of the PK tree the PK files go into.

# Mirroring CTAN

Michael Doob

Department of Mathematics, The University of Manitoba, Winnipeg, Manitoba, Canada

## Abstract

The plummeting cost of disk space has made it perfectly reasonable to keep a copy of some or all of the CTAN archive on your local network or even on your own desktop computer. Keeping it up to date without creating unbearable traffic on your network is the job of mirroring software. We'll look at some software available that does this job (almost) painlessly.

## The name of the game

Over the past decade CTAN (Comprehensive T<sub>E</sub>X archiving network) has become a standard tool for T<sub>E</sub>X users to obtain updates and even full installations of software and documentation. It is not only easy to access, but it also contains the latest version of (sometimes frequently) updated software. The only downside is that sometimes the Internet can be congested and the hosting sites are quite heavily used. In addition, those not graced with high speed connections to the Internet must show great patience when trying to download some of the larger packages. This situation can be ameliorated somewhat by using mirroring software.

The purpose of mirroring software is to keep a copy (mirror) of part or all of some authoritative site. This means that whenever a file is created, modified or deleted from that site, the same files is created, modified, or deleted in the mirror. This implies that the mirroring software must contact the site being mirrored on a regular basis to synchronize the file systems.

The current size of CTAN is just under 5 gigabytes, a size easily accommodated even on a personal desktop computer. In the next two sections we'll look at software using two different approaches to mirroring.

## The `wget` program from the `gnu` collection

The program `wget` is a relatively small C program developed as part of the `gnu` project of the Free Software Foundation. The home site for the `gnu` project is <http://www.gnu.org>; the source code is available there. For most UNIX systems, it is simply a matter of going through the `./configure` → `make` → `make install` cycle.

In addition, precompiled versions are available for the appropriate package manager for various Linux distributions (Red Hat, Debian and Slackware) and `wget.exe` is on CTAN. Once the executable is available, using it for small sections of CTAN is almost trivial. Suppose you want to get the macros for TUGBoat. Then using the command `wget -npm ftp://ctan.tug.org/tex-archive/macros/latex/contrib/supported/tugboat` will download all the files in the given directory on the (authoritative) site <http://ctan.tug.org> to your own computer. Of course you have to know the name of the directory you want (which can be pretty long) so it is natural to use `wget` in script or batch files. The `-m` option is for mirroring. This means that a record will be kept of the files downloaded, and calling `wget` again will result in only new or changed files being sent.

Once `wget` completes its download, you'll have a new directory `ctan.tug.org` which will have a subdirectory `tex-archive` and so forth so that the downloaded files will be in the same place in the directory tree as on the original site. Other pieces can be filled in as needed. In principal the whole site could be downloaded, but this would take days, even with a high-speed connection to the Internet (see the following section for an alternative).

The `wget` program will automatically do recursive downloads of directories. The command `wget -npmq ftp://ctan.tug.org/tex-archive/macros/latex/contrib/supported` would download all of the supported L<sup>A</sup>T<sub>E</sub>X macros (this makes for a pretty big transfer!) The `-mq` option is for mirroring quietly so no informative messages come to the screen. Rerunning the command would download only the macros that changed or were added

since the last download, so this could be done periodically (with a cron job in UNIX) and not incur long download times.

It should be noted that if files are deleted on the site being mirrored, they are not deleted from your computer (so in a sense the mirror is half-silvered). There are also many other options that can be used with `wget`, and they are given in detail in the excellent manual available from the GNU site.

### The mirror program from Imperial College, London

The `mirror` program is a perl script, so there is nothing to compile. It is available from <http://sunsite.org.uk/packages/mirror> or <ftp://ftp.cs.columbia.edu/archives/perl/mirror>. The main program is `mirror.pl`; it uses an auxiliary file `mirror.defaults` to configure itself. This is a text file pretty straightforward (and well documented) and allows, for example, for some files to be skipped. Here is a fragment to indicate the type of instructions that appear (in this example the CD images of the `texlive` distribution and the unneeded `00Contents` files are skipped):

```
package=ctan
comment=Comprehensive TeX Archive Network
      (CTAN) at St. Marys College
site=ctan.tug.org
timeout=400
exclude_patt+|\.texlive\*|00Contents|
local_dir=/mirror/ftproot/tex-archive
remote_dir=/tex-archive/
max_days=0
max_delete_files=20%
max_delete_dirs=20%
```

When the `mirror` program is run, it downloads the current directory tree of the authoritative site and compares it with the mirror directory tree. It then downloads new or revised files from the authoritative site and deletes files on the mirror as necessary.

This program is robust enough to mirror the full CTAN collection. It might be preferable to copy the CTAN image from the available CD and then update from an authoritative site rather than download 5 gigabytes across the net. Once it is set up, a nightly run using `mirror` will normally update the mirror in less than half an hour.

### Authoritative sites

There are three equivalent fully authoritative sites from which CTAN may be mirrored:

- <ftp://ftp.dante.de> (Hamburg, Germany)
- <ftp://ftp.tex.ac.uk> (Cambridge, UK)
- <http://tug.ctan.org> (Colchester, Vermont, USA)

In addition there are several dozen mirror sites listed in the file `/tex-archive/CTAN.sites` on CTAN itself (which themselves can be mirrored).

With only a small amount of effort, the areas of CTAN of interest can be mirrored on a personal computer; the latest software is then always available in a quick and easy fashion. It makes it easy to go forth and multiply!

# Installing TeXShop

Richard Koch

Mathematics Department, University of Oregon, Eugene, Oregon, USA

[koch@math.uoregon.edu](mailto:koch@math.uoregon.edu)

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

## Abstract

TeXShop is a  $\text{\TeX}$  previewer for MacOS X, Apple's new Unix-based operating system for the Macintosh. It uses  $\text{teTeX}$  as an engine, typesetting primarily with  $\text{pdfTeX}$  and  $\text{pdflatex}$ . The pdf output is displayed using Apple's internal pdf display code.

## $\text{\TeX}$ Implementations for MacOS X

MacOS X is Apple's new operating system for the Macintosh. Four features of this operating system interest us. First, the operating system is Unix at the ground level, so all standard  $\text{\TeX}$  programs run. Second, the command line interface is optional throughout the system, so programmers cannot assume that users understand Unix. Third, the system has a wonderful programming environment inherited from NeXT. This environment, OpenStep on the NeXT, is called Cocoa on the Mac. Finally, the graphic system on MacOS X is based squarely on pdf. For example, the Cocoa environment includes classes to display pdf images; low-level graphic primitives on the system correspond directly to pdf commands.

There are several ways to run  $\text{\TeX}$  on MacOS X. The operating system has a mode called "classic" which runs old Macintosh programs; Textures runs under Classic. The X-windows environment runs on MacOS X, so  $\text{emacs}$  and  $\text{xdvi}$  can be used. To obtain X, go to <http://www.apple.com/x11>. Go to <http://fink.sourceforge.net> for  $\text{emacs}$  and  $\text{xdvi}$ . Several native systems run directly on the new operating system, including C $\text{MacTeX}$  by Tom Kiffe, Oz $\text{TeX}$  by Andrew Trevorow, and TeXShop by my group. A number of these systems use the same  $\text{teTeX}$  distribution, which was compiled and configured for MacOS X by Gerben Wierda. The site <http://www.esu.psu.edu/mac-tex> lists all of these systems, with links to appropriate web sites. See <http://www.uoregon.edu/~koch/texshop> to obtain TeXShop and  $\text{teTeX}$ . TeXShop is free.

## Installing TeXShop

In the TUG conference program, 45 minutes were allotted to installing TeXShop; we'll see if that was

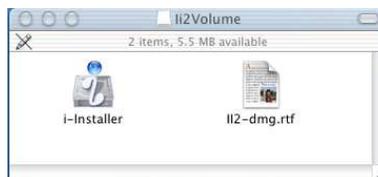
enough time. To install, double click on a file named **texshop.dmg** downloaded from my web site.



A window opens containing a TeXShop icon. Drag the TeXShop icon to the Applications folder. Done. TeXShop is installed.

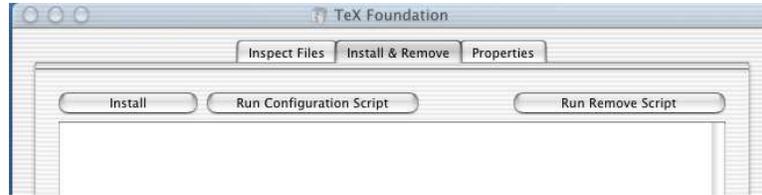
## Installing teTeX

We also need to install  $\text{teTeX}$ . Portions of TeXShop use  $\text{ghostscript}$ , so we install it at the same time. Double click on the file named **I12.dmg**.



A window opens containing the  $\text{teTeX}$  installer. Double click on this installer. A panel opens, as shown on the next page.

If we click "Install," the  $\text{teTeX}$  distribution is installed in `/usr/local/teTeX` without further input from us. We must also install two other packages; the process takes about five minutes, not counting download time. I refused to install  $\text{teTeX}$  at the conference because I once watched an Apple Demo



at the University of Oregon in which the demonstrator decided on the spur of the moment to install MacOS-X right before our eyes. Installation was almost complete when the two hour demo ended.

In all, three packages must be installed: TeX Foundation, TeX Programs, and Ghostscript 7. Optional packages are available through i-Installer for graphic conversion, cm-super fonts, etc.

You may have heard that Apple's installer has a bug which sometimes causes it to erase all programs in /Applications while installing new material. For that reason, Gerben Wierda wrote his own installer. You can find details, including full source code for `teTeX` and for Wierda's installer, on his site at <http://www.rna.nl/tex.html>.

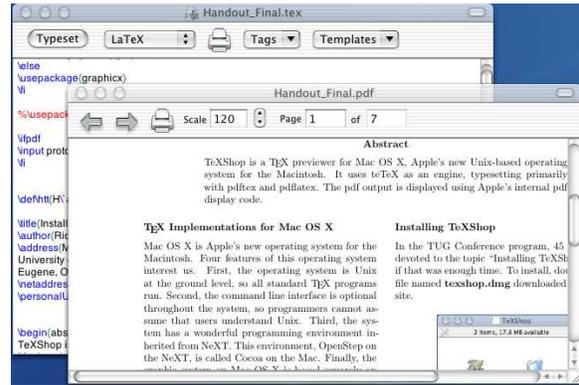
## Running TeXShop

Let's see what we've got. To the right, TeXShop is shown running. The window in the background is a source editor; the window on the front shows the resulting typeset document. The editing window has features we certainly expect: parenthesis matching, syntax coloring, tags. The preview window has buttons which increase or decrease magnification and page forward or backward.

Notice the button named "Typeset" at the extreme left of the editing window. Hitting this button saves the document and calls `latex` (actually `pdflatex`) to typeset. If the source code has an error, it can be fixed and typeset again without quitting the original `TeX` job. The popup menu labeled "LaTeX" next to the "Typeset" button changes the behavior of the first button, allowing users to call `TeX`, `LATeX`, `BIBTeX`, `MakeIndex`, `METAPOST`, `ConTeXt`, or `METAFONT`.

TeXShop can open multiple documents at the same time. Some of these might be `TeX` documents and others might be `LATeX` documents. When a document is first opened, the button labeled "LaTeX" will default to a choice set in preferences. If a user usually works with `LATeX` but happens to open a `TeX` document, it is easy to switch to `TeX` for that particular file.

By default, TeXShop calls `pdftex` and `pdflatex` to typeset, but `pdfetex`, `pdfelatex` and other programs can be chosen in the preferences dialog.



Notice the "Templates" menu. Choosing an item from this menu inserts appropriate text into the document. For instance, I use standard templates to begin `LATeX` documents and to insert graphic commands. The menu is configurable. Templates are stored in `~/Library/TeXShop/Templates`. When TeXShop starts, the names of files in this directory are listed in the Templates menu; the contents of a file will be inserted when the item is chosen. To modify the menu, simply change the programs in this folder.

## Behind the Scenes

In normal operation, TeXShop does not produce `dvi` files at all. Instead it calls Hàn Thế Thành's wonderful `pdftex` and `pdflatex` to directly output `pdf` files, which are rasterized by Apple's `pdf` software. The `pdf` file is also used in printing. MacOS X's printing architecture always prints using a two pass process: in the first pass the printing job is converted to a `pdf` file, and in the second pass this `pdf` file is processed for a particular printer. In TeXShop, the first pass has already been done, so the same `pdf` file is used for displaying to the screen and printing to the printer. TeXShop has no special code for particular printers.

The basic philosophy behind TeXShop is simple: if software already exists to do a job, use that software rather than inventing your own. TeXShop carries this philosophy to extremes — it doesn't do any hard work itself. Editing is done by the

text object written at NeXT, typesetting is done by the software invented by Knuth and many others, and pdf display is done by Apple's pdf rendering routines.

Here's how TeXShop came about. I bought a NeXT when that machine was introduced because I thought it would be fun to demonstrate to others. Wrong. Friends who used Unix pushed the NeXT windows aside and opened a large terminal window. Mac friends said, "I've got that stuff on my Mac and why are the scroll bars on the left?"

But the NeXT was a wonderful working machine, and I came to rely on Tomas Rokicki's `TeXView`, a  $\TeX$  previewer which made use of the NeXT's Display PostScript. When Apple bought NeXT, I celebrated and began pestering my Apple representative to make sure `TeXView` was ported. The representative said "we understand that  $\TeX$  is important and we're working on it." But later he said "who's this Tomas Rokicki anyway."

Around that time someone ported `teTeX` (minus `xdvi`) to the MacOS X server. Shortly after that I learned about `pdftex` and `pdflatex` and discovered that they were present in the `teTeX` port. What a revelation. Thank you, Hàn Thê Thành!

But would Apple's software display the pdf? According to the html documentation for Cocoa's `NSPDFImageRep` class, the class had only a few methods, and none to change the page displayed. "Obviously," I thought, "Apple is using a very primitive form of pdf for graphic display." Then on a fluke I looked at the header files and found other undocumented methods. Done!

Constructing TeXShop was easy. Cocoa contains a powerful text editing object. When I wanted to run  $\LaTeX$ , I looked at the html list of classes and found `NSTask`, an object which runs other Unix processes. Later I wanted to obtain  $\LaTeX$  errors through some sort of pipe, so I glanced through the html list of classes. "Aha. Here's one called `NSPipe`." It was all about that simple.

During the beta period for MacOS X, Apple's pdf routines refused to recognize embedded fonts, so TeXShop had the following mild restrictions: you couldn't use symbols and your font had to be Times Roman! It was a thrill to run TeXShop on the release version of MacOS X and see  $\TeX$  fonts for the first time.

The Cocoa classes provide so much functionality that it sometimes seems they work automatically. After using TeXShop for a while, I needed to find something and hit command-f, but no Find panel appeared. "What sort of lousy software is this

Cocoa?" I thought. "It didn't even give me a Find panel." TeXShop has one now.

A user wrote asking if we could implement a specific emacs key binding. I replied that TeXShop had no key bindings, but we might add some in the future. "What do you mean?" he wrote back. "Most emacs bindings are there." Sure enough . . .

## The Team

TeXShop is a free program, released under the GPL. Source code is available on my web site. Shortly after TeXShop was released, Dirk Olmes offered to contribute to the program. At least that's how I remember it; perhaps he really wrote "buster, you need help."

When Olmes began, the program code was in a single source file! Olmes broke the code into pieces, rationalized it, and completely rewrote the preferences system while teaching me about the defaults database on OS X. He is now part of our team. Shortly afterward, Jérôme Laurens contributed the TeXShop icon, which I like a lot. Jérôme's signature reads "Dijon, France, where the mustard grows," so I have a romantic image of his home.

Gerben Wierda has become a more and more crucial player in the team. He carefully crafted a version of `teTeX` which can be used by Mac folks who don't understand Unix. Several scripts used by TeXShop were written by Wierda. He fixed `BIBTeX` so it can deal with files with Mac line feeds. I'll have more to say about his `teTeX` in a moment.

Above all, Wierda constructed the `teTeX` installer; Unix no longer confronts a user who wants to install TeXShop.

It is interesting that the members of this team come from different countries and speak different languages; I have never met any of them. I don't know how old they are, or how they talk, or where they work. For all I know, one of them is a ten-year old kid.

A great many other users have suggested features which are now in TeXShop. Their names and addresses can be found on my web site under the "Credits" link.

## Advanced TeXShop Features

Of course eps illustrations must be converted to pdf form for `pdflatex`. One hopes that future Macintosh programs will create pdf directly, but that may not happen soon; the MacOS X version of Mathematica offers to "Save Selection" as eps, but not pdf.

A few months ago, Sean Luke suggested that TeXShop should be able to typeset using the sequence `latex` → `dvips` → `ps2pdf` to convert source to `dvi`, and then to PostScript, and finally to `pdf`. He sent a shell script to do that. The advantage of this typesetting approach is that old projects with `eps` illustrations and PostScript special commands typeset directly without modification. TeXShop now includes an option to typeset that way; the option calls an expanded version of Luke's script carefully constructed by Gerben Wierda. The `teTeX` distribution has been configured to make the resulting `pdf` file use Type 1 fonts when possibly so it will display rapidly and crisply with Apple's software. Notice that the option requires `ghostscript` for the final conversion.

The user interface for this option has confused some users. TeXShop is designed so users can have several documents open at once. We imagine that some of these documents will contain `pdf` illustrations and be typeset by `pdflatex`, while others will contain `eps` illustrations and be typeset with `TeX` and `ghostscript`. There is a default typesetting command set in preferences; when documents are first opened they are assigned the default typesetting method. But a command in the typesetting menu allows users to change the typesetting method for a particular file after it is open. Users who don't understand this design tend to change the typesetting method in preferences and then discover that the change doesn't affect documents already open.

The latest version of TeXShop has a feature stolen from Rokicki's `TexView` to simplify choosing a typesetting method. If the first line of a document is `%&pdflatex`, then `pdflatex` is used. If the first line is `%&latex`, then `latex` and `ghostscript` are used. Similar remarks hold for `%&pdftex` and `%&tex`. If no such line exists, TeXShop reverts to the previous method to determine the typesetting engine.

When working with `TeX` projects, users often need to examine other files briefly. TeXShop can open many such files itself. This includes text files; log files can be opened and inspected, `BIBTeX` files can be opened and edited, etc. TeXShop can also open `jpg`, `tiff`, and `pdf` illustrations; indeed it can open any `pdf` file. The latest version of TeXShop is also able to open `dvi` files, PostScript files, and `eps` files. Obviously this is done by converting the file to `pdf` and displaying the `pdf`.

In particular, TeXShop can be used to convert `eps` illustrations to `pdf` format, since the process of opening an `eps` also converts it. The conversion indirectly uses `epstopdf`, but since Macintosh `eps` files sometimes have Macintosh line feed conven-

tions, the conversion script is able to deal with `eps` files regardless of the line feed used. In particular, TeXShop users can now convert `eps` illustrations from Mathematica to `pdf` without first modifying the file.

When `TeX` is used to write a book, it is common to organize the project using a controlling "root" file. The chapters of the book and their illustrations are put in separate directories inside the project directory. TeXShop can deal with this organization. In the TeXShop "File" menu, there is an item named "Set Root File." This item allows users to assign a root file to any particular `TeX` source file; when the source file is typeset, the file is saved, but TeXShop typesets and displays the root file. This root file can be in any directory.

I'm not certain this design is optimal because a root file must be set for each separate book chapter. It would be nice to configure the project just once. So the root file syntax may be modified at some later time.

### Configuring teTeX

Although `teTeX` is the engine underlying TeXShop, users may choose to ignore it completely; `teTeX` will do the hard work silently.

In MacOS X, Unix directories are hidden from the user. In particular, the directory where `teTeX` is installed, `/usr/local/teTeX`, is not visible from the finder. Gerben Wierda automatically adds a symbolic link to this directory to the system Library folder, which is visible. To inspect `teTeX` in the Finder, go to `/Library/teTeX`.

The `teTeX` documentation can be found in `/Library/teTeX/share/texmf/doc/tetex`. In particular, see the files `TETEXDOC.pdf` and `teTeX-FAQ`. More documentation for Wierda's MacOS X compilation is in `/Library/teTeX/README.macosx`.

It is possible to store additional style files, fonts, etc., in `teTeX` itself. But that is not a good idea; it is better to keep the `teTeX` directory pure so it can be upgraded when later versions are released. Instead, users should construct a mirror image of the `teTeX` directory structure inside their `~/Library` folder and store personal files there.

Gerben Wierda has configured `teTeX` to look for `TeX` files in four locations, in the following order:

- `~/Library/texmf`
- `/usr/local/teTeX/share/texmf.local`
- `/usr/local/teTeX/share/texmf.macosx`
- `/usr/local/teTeX/share/texmf`

The first of these spots is where individual users will put extra files. The second is where the system

administrator will put files for all users on the computer. The third is where Gerben Wierda will put his own configuration files for `teTeX`. The final spot is the original `texmf` tree maintained by the `TeX` community and kept as default as possible.

When new upgrades to `teTeX` are introduced, the installer will overwrite files in the last two directories. But files in the other directories will be left alone.

The `teTeX` system uses a directory structure invented by the TUG working group. When users want to add files to the system, they need to inspect `/Library/teTeX/share/texmf` to find a likely spot for the file, and create a mirror image directory in `~/Library/texmf`.

I'll show how this works by answering three common email questions about TeXShop. As you'll see, we are only partway along the path leading to Mac-like ways to administer `TeX`. The answer to the first question below is satisfactory, but the remaining answers require more Unix than we'd like.

### How Do I Install REVTeX?

At the time of the conference, REVTeX was not included in Gerben Wierda's distribution. It is now, but I'll still use it to explain how to install extra packages in the system. I'll simulate a typical installation, including an error which must be debugged.

Visit <http://publish.aps.org/revtex4/>. At the bottom of this page our package is available as `revtex4.tar.gz`. Unpack this package.

The folder contains seven input files:

```
10pt.rtx 11pt.rtx 12pt.rtx aps.rtx
revsymb.sty revtex4.cls rmp.rtx
```

It also contains two `BIBTeX` files: `apsrev.bst` and `apsrmp.bst`. We look inside the `teTeX` directories and find that most `LATeX` include files are in folders inside `tetex/tex/latex`. Therefore, we create the folder `~/Library/texmf/tex/latex/revtex` and place all nine REVTeX files in that folder.

Next we try REVTeX on some sample files. Everything works until we use `BIBTeX`; the system complains that it cannot find `apsrev.bst`. We conclude that `BIBTeX` include files belong in a different directory.

Looking again at `teTeX`'s `tetex` directory, we find a subdirectory named `bibtex`. Inside this directory is a directory named `bst`. Probably `BIBTeX` include files go into this directory. So we create `~/Library/texmf/bibtex/bst/` and place the two `bst` files from REVTeX inside. Then everything works.

### How Do I Typeset TeX Files Created by Mathematica?

Mathematica can save notebooks in `TeX` format. This `TeX` source requires special include files and fonts, which we must install. The required files are packaged with Mathematica. The instructions which follow refer to the MacOS X version of Mathematica. Click on the Mathematica icon while holding down the control key. A dialog allows us to open Mathematica as a folder.

Inside, we find many files, including `TeX` files. Go to `SystemFiles/IncludeFiles/TeX` and copy files as follows:

- `texmf/tex/latex/wolfram`  
→ `~/Library/texmf/tex/latex/wolfram`
- `texmf/fonts/tfm/wolfram`  
→ `~/Library/texmf/fonts/tfm/wolfram`
- `texmf/fonts/vf/wolfram`  
→ `~/Library/texmf/fonts/vf/wolfram`
- `texmf/dvips/init/wolfram.map`  
→ `~/Library/texmf/dvips/config/`
- `/SystemFiles/Fonts/Type1`  
→ `~/Library/texmf/fonts/type1/wolfram`

So far the instructions are Mac-like, if somewhat complicated. Unfortunately, Wolfram supplies type 1 fonts in `pfa` format, and `teTeX` requires fonts in `pfb` format. So we must convert each Wolfram font from one format to the other. The program required to do this is called `tblbinary`; luckily it exists in `teTeX`. To convert, say, `Mathematical.pfa` to `Mathematical.pfb`, type the following command

```
tblbinary --output=Mathematical.pfb
Mathematical.pfa
```

Convert all forty-six fonts in this way.

When Mathematica outputs notebooks in `TeX` format, it writes graphic content as a series of `eps` files. Hence the notebook should be typeset using TeXShop's "TeX and Ghostscript" option.

### How Do I Use the Lucida Bright and MathTime Fonts?

These are commercial fonts, so `teTeX` does not contain them. But `teTeX` has all required supporting files, so it is only necessary to obtain the Type 1 fonts themselves and add them to the system.

Many users already own these fonts for earlier Mac operating systems. I'll describe how to use them for users who bought the fonts from Blue Sky Research for Textures. There is just one problem. The Textures fonts are in a Mac format used by Adobe Type Manager. But `teTeX` uses the `pfb` format required in the Windows world and elsewhere.

|                |      |                   |      |                   |      |
|----------------|------|-------------------|------|-------------------|------|
| LucidBri       | lbr  | LucidSan          | lsr  | LucidSanTyp       | lstr |
| LucidBriDem    | lbd  | LucidSanDem       | lsd  | LucidSanTypBol    | lstb |
| LucidBriDemIta | lbdi | LucidSanDemIta    | lsdi | LucidSanTypBolObl | lsbo |
| LucidBriIta    | lbi  | LucidSanIta       | lsi  | LucidSanTypObl    | lsto |
| <hr/>          |      |                   |      |                   |      |
| LucidFax       | lfr  | LucidNewMatAltIta | lbmo | LucidCalIta       | lbc  |
| LucidFaxDem    | lfd  | LucidNewMatArr    | lbma | LucidHanIta       | lbh  |
| LucidFaxDemIta | lfdi | LucidNewMatExt    | lbme | LucidBla          | lbl  |
| LucidFaxIta    | lfi  | LucidNewMatIta    | lbmi | LucidBriObl       | lbsl |
|                |      | LucidNewMatSym    | lbms |                   |      |

Luckily, free conversion software is available on the internet. Running this software is the unpleasant part.

The software needed is called `t1unmac`. At the conference I had to explain how to retrieve this software and compile it. Luckily, the software is now part of Gerben Wierda's `teTeX` distribution.

The `t1unmac` Unix program which will do the conversion. But Unix does not understand the resource fork used by Macintosh fonts, so we must first pack these fonts into a flat file using the old Mac utility `BinHex4`. As an example, let us convert the Lucida fonts. Run `classic`, run `BinHex4`, and use the menu `Application` → `Upload` to convert `LucidBri` to `LucidBri.Hqx`. Convert the other font files similarly.

Move these files to your home directory. Open Terminal and execute the command

```
t1unmac --binhex --pfb
--output LucidBri.pfb LucidBri.Hqx
```

to convert the first font to `pfb` format. Convert the other files the same way.

The `pfb` files must now be renamed; for example, `LucidBri.pfb` to `lbr.pfb` (see the table at the top of the page). Place the renamed `pfb` files in `~/Library/tetex/fonts/type1/yandy/Lucida`. To use these fonts instead of the `cm` fonts, add the following line to your  $\LaTeX$  document:

```
\usepackage{lucidbry}
```

### Coexisting with Fink

MacOS X uses its own graphic system rather than the X Window System. But X-windows can run on MacOS X. When it does, the user has two views of the same file system and can switch between these views with a keystroke.

It is still necessary to obtain and compile X-windows software like `ghostview`, `xdvi`, and `xpdf`. A remarkable system called Fink has been invented to ease this task. Suppose Fink has been installed and suppose you want to obtain `ghostview`. Type “fink install gv” in the Terminal. Fink will consult its

database and discover that `gv` requires `ghostscript` and `ghostscript-fonts`. It will ask permission to install these as well, and will then retrieve the source files, retrieve patches for MacOS X, compile the code, and install it, all automatically.

To avoid overwriting files, Fink installs software in its own location, `/sw/bin`. So far, so good. But when Fink is setup, it modifies the `PATH` variable to search `/sw/bin` first. Thus Fink versions of software will be used even if other versions are also installed.

At the time of the conference, this caused problems. Most of these problems have since been fixed, but I'll discuss them briefly for amusement.

Originally TeXShop used `ghostscript 6` rather than `ghostscript 7` because Apple's pdf rendering software could not interpret  $\TeX$  fonts in pdf files created by `ps2pdf` in `ghostscript 7`. So it was important that TeXShop call the version of `ps2pdf` in `/usr/local/bin` rather than the Fink version in `/sw/bin`. Unhappily, `ps2pdf` is a script which ultimately calls `ghostscript`, so even if we called the correct version, the wrong version of `ghostscript` would run.

Just before the conference, we released new versions of TeXShop and `teTeX` which were inoculated against Fink. TeXShop now calls scripts which temporarily reset the `PATH` variable before calling `ps2pdf`. Even if users installed the fink version of `ghostscript`, TeXShop will run with no problems. We also modified TeXShop so it will use Gerben Wierda's distribution even if a second version of `teTeX` is installed.

Later Apple fixed the pdf rendering bug, and our system now uses `ghostscript 7`.

Since the conference, the Fink distribution of `teTeX` was modified. The fink installer now asks if it should install its own version or make symbolic links to a version already installed in `/usr/local/bin`. Users should choose the “symbolic link” option.

Those who install the fink version of `teTeX` are asking for trouble, because any  $\TeX$  program called from the command line will use the Fink version,

which is not configured to look for extra files in `~/Library`. Thus, include files will mysteriously be found sometimes and not at other times. There is little reason to install the Fink version of `teTeX` because X-windows software should work fine with Gerben Wierda's distribution. Certainly that is true of the `xdvi` installed by Fink.

### Missing Features

If you are using TeXShop, please report bugs and request extra features. Our email addresses are listed at the end of this article.

Four requests were often made at the time of the conference. We have responded to some of these requests since then. Here are the requests:

**Better spell checking:** Recent versions of TeXShop set the Macintosh type of files to `TEXT`. Therefore, these files can be opened with Excalibur, a wonderful  $\LaTeX$  spell-checker. The spell checker `cocoaAspell` by Anton Leuski was introduced after the conference and also works with TeXShop.

**Magnifying glass in the preview window:** This was introduced in version 1.27, using code provided by Mitsuhiro Shishikura. Shishikura added other important features in 1.27.

**Edit using other editors:** Some users want to edit their  $\TeX$  files with BEdit or xemacs. This can be done with versions introduced after the conference.

**Coordinate the source and pdf windows:** Textures has a feature called Synchronicity. We are often asked to provide it for TeXShop. But the Apple pdf classes have no methods for searching pdf files or clicking in the image and finding the corresponding spot in the file. So providing this feature would be hard work.

Samuel Eilenberg was a great topologist. One day a student said to him "Professor Eilenberg, I have decided to work in algebraic topology." Eilenberg replied "Young man, don't let me stand in your way."

Similarly, the TeXShop source code is available at my web site. "Young man, . . ."

### Email Addresses

Richard Koch [koch@math.uoregon.edu](mailto:koch@math.uoregon.edu)

Dirk Olmes [dirk@xanthippe.ping.de](mailto:dirk@xanthippe.ping.de)

Gerben Wierda [Gerben\\_Wierda@rna.nl](mailto:Gerben_Wierda@rna.nl)

# Font Installation: Agfa Eaglefeather to Linotype Zapfino

William F. Adams

ATLIS, 75 Utley Drive, Suite 110, Camp Hill, PA 17055

[wadams@atlis.com](mailto:wadams@atlis.com)

## Abstract

Here's how I managed to get Agfa Eaglefeather and Linotype Zapfino installed in  $\TeX$  and some of the things which I learned, from A to Z.

**Beginning** First, acquire the fonts. I'd purchased Agfa Eaglefeather back when it first came out to do the signage for a Frank Lloyd Wright exhibition at the Longwood Center for the Visual Arts. I've always wanted Zapfino since first hearing of it, so was thrilled to learn it was bundled with Mac OS X.

Choose your font vendor carefully. Considerations include:

Are they on the "short list" of font vendors which are listed in Fontname (<http://www.tug.org/fontname/>)? This eases the process of installation, Agfa for example isn't, and I was having trouble getting the install of Eaglefeather started until I hit upon the idea of lumping it in with Monotype Octavian which I'd installed previously, on the theory that assimilation works both ways (Monotype was bought out by Agfa).

Are `.afm` (adobe font metric) files provided/available? Not all vendors do this, especially for fonts provided in Mac format. If `.afm` files are not available, one can make one's own, with certain caveats. The typical case is a Mac `lwf` printer font and bitmap screen font pair, in which case the `bmp2afm` utility from CTAN and other file archives will extract the metric and kerning information. For the Windows `.pfm` (postscript font metric) format files similar utilities are available. Additionally, most font editors can access this information and write `.afm` files. `pfaedit` is an open source font editing/creation program recently announced on SourceForge (<http://www.sourceforge.net>).

Who made the font and why? Does the font vendor acknowledge the font's designer at least? Make royalty payments? Provide information which helps in learning how the font was first used (or ideally tutorial exemplars such as the lovely booklets which accompanied the Adobe Originals), where it was first used (a few fonts are still named for the book in which they were first used. . .) and when so that one may write a decent colophon?

As implied above, it is the font's vendor which, in the Fontname scheme, determines the beginning

of the font name. For Agfa-Monotype Eaglefeather, "M", for Apple-Linotype Zapfino, "E". the appropriately named sub-directories (`mef` and `ezo`) for the fonts are placed in directories for the relevant vendor (`monotype` and `apple`). If you are using a font from a vendor to which no letter or number is assigned, I would suggest using "z" for misc instead of attempting to assign something oneself—not only does this raise the spectre of incompatibility should the number later be formally assigned, it seems not to work without reconfiguration of the  $\TeX$  Directory Structure settings.

**Conversion** As a graphic designer by trade, I've always purchased Mac format fonts, working on the theory it's easier to convert from Mac to PC/Unix than in other directions. (This habit was formed, of course, before `.pdf`) There are a number of tools for this, I use a combination of: Ares FontHopper (to get from Mac to PC format), and Type Designer by Manfred Albracht (to regularise the encoding and generate a `.afm`), and of course the afore-mentioned `bmp2afm` and other programs. Of notable interest for those with older  $\TeX$  systems, `ttf2afm`, a part of the FreeType project allows `.dvi` processors to make use of TrueType outline fonts by converting them into bitmap fonts.

Zapfino for MacOS X is provided in the new `.dfont` resource fork-less format, which is as yet unique to MacOS X. Fortunately, the font tools group at Apple has created "Fork Switcher", a small utility program which switches a font from resource to data fork. Zapfino is also available from Linotype in a more prosaic Type 1, multiple font format, but the Apple version has many additional characters (a total of 1,286) and promises to eventually provide elegant access to all them and more which are yet to be created. . .

Fork Switcher is available from <http://fonts.apple.com>. Unfortunately, Ares FontHopper has been discontinued since Adobe purchased Ares, and DTP Software became the German distributor

for Yuri Armola’s FontLab program, (<http://www.pyrus.com> in the U.S., or <http://www.fontlab.com>) which subsumed the code and features of Type Designer. (Note: This latter Type Designer is *not* the font editor which once upon a time was sold by Letraset.)

**Decisions** Eaglefeather was rather straightforward once I’d worked out a solution to its lacking an Expert font set and having “merely” a Small Caps font. (I made my own Expert fonts. This is probably a variant of the “If all one has is a hammer…” maxim.)

I chose “ef” for the name, since it seems unlikely I’d need Egyptienne (the canonical font for this letterpair). Arguably, I should use an unused letterpair, since T<sub>E</sub>X insists on making a folder for `egyptien.tfms`.

Zapfino has proven more complex, and even now, I’ve not reached an equitable solution. Unfortunately, despite its having over 1,000 unique glyphs, very few of them are actually wired up (for example, there’s no way to access the old-style figures) for usage in TextEdit.app or WorldText.app (née GXWrite from Apple’s QuickDraw/GX)—this program is available in the Extras folder of the Developers’ Applications (if you didn’t get a Developer CD with your copy of MacOS X, the Tools are available as a download with (free) membership in the Apple Developers’ Connection) in MacOS X at this time (10.1 has not yet been released as of this writing). Sadly, this is confirmed by my usage of Zapfino in T<sub>E</sub>XGX. Although installation there was almost painfully easy (switch font fork with Fork Switcher, drag to System Folder:Fonts, start T<sub>E</sub>XGX, use font), GX does not provide access to font features for which layout tables have not been written, and at this time these are limited to the binary options (to ligature or not to ligature) which Apple’s TextEdit.app and other programs which use the `nstext` object make available. One additional consideration here is that the font after conversion bears special handling since it is set for installable embedding which seems *not* to be covered by Apple’s software license. Correspondence with Apple asking for guidance or clarification on this issue has thus far gone unanswered—it’s worth noting though that Apple removed font tables from their system fonts so as to preclude the conversion and installation of same in other operating systems.

**Eaglefeather** David Siegel, the designer of this font under license from the Frank Lloyd Wright Foundation has written “The Story of Eaglefeather” which is available from his website, <http://www.dsiegel.com>.

By the time this is printed a .pdf version set in the font itself should be available from my personal web site <http://members.aol.com/willadams>.

**Font Installation:** `fontinst` I must confess that I use this amazing macro package by Alan Jeffrey as a “black box” and merely feed it appropriately named collections of .afms and a T<sub>E</sub>X file consisting of `\input fontinst.sty`, an appropriate `\latinfamily` command, and `\bye`. In addition to the manual, I would especially recommend reading Tako Hoekwater and Ulrik Vieth’s presentation for EuroT<sub>E</sub>X ’99 “Surviving the T<sub>E</sub>X font encoding mess” (filename `et99-font-tutorial.pdf`) which is included in the pre-release `fontinst` documentation directory on CTAN.

It is my understanding after extensive reading of the manual and the `weaved` source of the package itself (included with the pre-release version on CTAN, filename `fisource.pdf`) that it can be controlled by .etx files, and hope to eventually find the time to experiment with that. Alan Hoenig’s wonderful (but sadly out-of-print) book *T<sub>E</sub>X Unbound* has extensive tutorials on this as well. One observation before I begin a description of low-level direct manipulation which a better understanding of `fontinst` might obviate: .etx files are created in the course of font installation by `fontinst`, and if a second (tweaked) install is attempted, they will be used in lieu of (updated) .afm files which have the same filename.

**General Tweaking** Making the O FLOAT.

This is specifically suggested for adjustment in “The Story of Eaglefeather”, and I did so—couldn’t resist. For those with a similar weakness, we start with the `vp1` for the roman font, and construct an LO ligature First, find an empty character slot. Since there’s no “Eng” at D 141, we’ll use that. (My apologies to those whose native tongue requires this character.)

```
(CHARACTER D 141 (COMMENT Ng)
  (CHARWD R 5.0)
  (CHARHT R 5.0)
  (CHARDP R 0.0)
  (MAP
    (SETRULE R 5.0 R 5.0)
    (SPECIAL Warning:
      missing glyph ‘Eng’)
  )
)
```

and add in the small cap “o” we’ll be inserting.

Find a character to replace and grab the replacement character:

```
(CHARACTER D 111 (COMMENT o)
```

```
(CHARWD R 5.81995)
(CharHT R 4.8999)
(CharDP R 0.10999)
(MAP
  (SETCHAR D 111) (COMMENT o)
  )
}
```

Merge the two entries:

```
(CHARACTER D 141 (COMMENT small cap o)
  (CHARWD R 5.81995)
  (CHARHT R 4.8999)
  (CHARDP R 0.10999)
  (MAP
    (SETCHAR D 111) (COMMENT o)
    )
  )
```

and then tell  $\TeX$  from where to draw the character:

```
(CHARACTER D 141 (COMMENT small cap o)
  (CHARWD R 5.81995)
  (CHARHT R 4.8999)
  (CHARDP R 0.10999)
  (MAP
    (SELECTFONT D 1)
      (COMMENT mefr8x at 10.0pt)
    (SETCHAR D 111) (COMMENT o)
    )
  )
```

Unfortunately,  $\text{KRN}$  only allows us to move characters horizontally, so a character slot must be dedicated to this as far as I know. So, we create the ligature character:

```
(CHARACTER D 173 (COMMENT L_o)
  (CHARWD R 6.875)
  (CHARHT R 6.39999)
  (CHARDP R 0.0299)
  (MAP
    (PUSH)
    (MOVERIGHT R 1.75)
    (MOVEUP R 2.2)
    (SETCHAR D 111) (COMMENT o)
    (POP)
    (SETCHAR D 76) (COMMENT L)
    )
  )
```

Then we go back and insert a reasonable  $\text{KRN}$  for A...

After doing this, we must note that even though it was missing, “ng” was kerned (with the values for ‘n’ I suppose), so we want to delete such kerns. Similarly, we delete any kerns for where we put the “o”. This part could be automated by `fontinst`, I believe...

Kerning after the  $\text{L}_o$  pair hopefully won’t need too much adjusting.

Next, noting that “L” is CHARACTER D 76, we modify the `LIGKERN` entry for o which is already there, changing it from a kern to a ligature; from: `(KRN D 111 R -0.48999) (COMMENT o)` to: `(LIG D 111 D 173) (COMMENT o L_o)`.

**Hyphenation** One of the things which `fontinst` doesn’t do for the fonts which it installs is a zero-width hyphen. Fortunately, as Don Hosek of Quixote <http://www.quixote.com>, publisher of the magazine *Serif: The Magazine of Fine Type and Typography* noted in a post to `comp.text.tex` ages ago, this is quite easy to do, just:

- open up the appropriate `.vpl` (e.g., `mefr9d.vpl`)
- find the “hyphenchar” if it exists, e.g.,

```
(CHARACTER D 127 (COMMENT hyphenchar)
  (CHARWD R 2.90991)
  (CHARHT R 3.05994)
  (CHARDP R 0.0)
  (MAP
    (SETCHAR D 45) (COMMENT hyphen)
    )
  )
```

- zero out `CHARWD` (or adjust it to the desired value if having the hyphen hang all the way out is not desirable)
- if “hyphenchar” doesn’t exist, copy the real hyphen into a blank character slot. Note: `CHARDP` was `-2.5` for the real thing...not sure why it should be zeroed out for this.

Then, in one’s  $\TeX$  file, one must set the hyphenchar in each font definition macro to the correct slot, e.g., `\hyphenchar\efroman = 127`.

**Installation** Once `fontinst` has turned out the basic font property list files (both normal, `.pl` for regular fonts and `.vpl` for virtual fonts), one must convert these into the `.tfm` ( $\TeX$  Font Metrics) and `.vf` (virtual fonts) files which  $\text{TeX}$  itself will use by using `pltotfm` and `vptovf`. Doing this is specific to one’s system/ $\text{TeX}$  implementation and is well-described in the `fontinst` manual. There are comments which I copy which describe the appropriate command for each file, but the executable is misnamed in them (`pltotfm` vice `pltotf` and `vpltovf` vice `vptovf`).

Then, the files should be stored away in the proper places, the fonts added to the `dvips config.ps` file by way of an added map file and one’s  $\text{TeX}$  filename database refreshed.

**Justification** A thing which bears noting on T<sub>E</sub>X’s method of justification is that it does not alter inter-character spacing—this ensures that ligatures will be visually compatible with the text in which they reside. Readers with access to a copy of MacOS X can graphically explore this by setting “office” in the Snyder Bold font bundled with MacOS X in TextEdit, selecting it and turning on “all ligatures”.

**Kerning** A font installed in T<sub>E</sub>X will inherit the kerning from the relevant .afm files (if any). Kerning should be tested with a text which contains all of the letter triplets used in the intended language, as well as as sensible Cap/lowercase pairs. Such a file for English which makes use of Webster.app, Digital Shakespeare and the *King James Version of the Bible* is available at <http://members.aol.com/willadams/typography/textsamples>

**Ligatures** Only the “basic” f-ligatures are installed by fontinst for a typical font since those are the only ones normally available. Eaglefeather only has f-ligatures as actual glyphs, while others can be created by careful kerning of select letterpairs.

Zapfino on the other hand, has a plethora of ligatures (if only one could get at them).

#### Quote

TeX started out with a highly irrational and non-composable design, which was partly helped and yet a bit furthered by the fine L<sup>A</sup>T<sub>E</sub>X NFSS. Making sense out of it all is most difficult.

One concludes that human language and their alphabets cannot be coldly reduced to integer codes. The letter is kept; the spirit is lost.

*Dr. Richard Kinch* responding to Javier Bezos in the thread “T<sub>E</sub>X and Unicode = where are we?” in `comp.text.tex`  
Wednesday 25 April 2001 12:42:01 PDT

**Samples and Testing** The canonical way to test T<sub>E</sub>X fonts is “`tex testfont`”. This provides a number of options for providing various character charts and text samples which are fairly rigorous, and quite good at exposing problems (e.g., missing accents, but only for those using plain T<sub>E</sub>X-like methods of accessing accented characters) with a font which might otherwise be left until crunch time.

Stephen Moyer has also done some wonderful macros for generating specimen pages – `typespec01-10` – which are available on CTAN where plain T<sub>E</sub>X-contributed macros are stored.

**Zapfino** Zapfino had its origins in Prof. Zapf’s 1944 sketchbook, when he was a mapping officer during World War II. A previous attempt to render those letterforms as type, Virtuosa Script for D. Stempel had been rather compromised by the limitations of hot metal matrices, especially the swash letters.

This design was revived when David Siegel in 1993, after working on the Euler project with Prof. Zapf, and after graduating approached him about a chaotic calligraphic typeface based upon an example done for the Society of Typographic Arts in Chicago. Remembering the page from his sketchbook, Prof. Zapf saw the chance for a design without compromises due to the advantages afforded by digital type technology.

Digitization was done by Gino Lee, but production halted due to personal problems, and languished until Prof. Zapf showed the design to Linotype. It was then rendered as a traditional, multiple alphabet typeface family.

Apple’s having John Hudson of Tiro Type re-encode these multiple fonts as a single entity brings this full circle for their new MacOS X and its “ATSUI” (Apple Typographic System for Unicode Information), and Zapfino is, or rather will be, a full-featured hand-writing font...



## Calendar

### 2001

- Sep 8 WDA'2001: First International Workshop on Web Document Analysis, Seattle, Washington. For information, visit <http://www.csc.liv.ac.uk/~wda2001>.
- Sep 17–20 XML World 2001, San Francisco, California. For information, visit <http://www.xmlworld.org/>.
- Sep 23–27 EuroT<sub>E</sub>X 2001, “T<sub>E</sub>X and Meta: the Good, the Bad and the Ugly Bits”, Kerkrade, Netherlands. For information, visit <http://www.ntg.nl/eurotex/>.
- Sep 29 29th Annual General Meeting of the Danish T<sub>E</sub>X Users Group (DK-TUG), Århus, Denmark. For information, visit <http://sunsite.dk/dk-tug/>.
- Oct 24–26 4<sup>th</sup> International Conference on The Electronic Document, Toulouse, France. For information, visit <http://www.irit.fr/CIDE2001/>.
- Nov 9–10 ACM Symposium on Document Engineering, Atlanta, Georgia. For information, visit <http://www.documentengineering.org>.
- Feb 19–22 Seybold New York, New York City. For information, visit <http://www.key3media.com/seyboldseminars/ny2002/>.
- Feb 20–23 DANTE 2002, 26<sup>th</sup> meeting, Universität Erlangen-Nürnberg, Germany. For information, visit <http://www.dante.de/dante2002/>.
- Mar 13–Apr 23 The Best of the Best: A traveling juried exhibition of books by members of the Guild of Book Workers. San Diego State University Malcolm A. Love Library, San Diego, California. Sites and dates are listed at <http://palimpsest.stanford.edu/byorg/gbw>.
- Apr 29–May 3 EuroBachoT<sub>E</sub>X 2002, 13<sup>th</sup> meeting of European T<sub>E</sub>X Users and 10<sup>th</sup> annual meeting of the Polish T<sub>E</sub>X Users' Group (GUST), “T<sub>E</sub>X and beyond”, Bachotek, Brodnica Lake District, Poland. For information, visit <http://www.gust.org.pl/BachoTeX/2002/>.
- May 7–Jun 27 The Best of the Best: A traveling juried exhibition of books by members of the Guild of Book Workers. San Francisco Public Library, San Francisco, California. San Diego, California. Sites and dates are listed at <http://palimpsest.stanford.edu/byorg/gbw>.

### 2002

- Jan 16 – Feb 20 The Best of the Best: A traveling juried exhibition of books by members of the Guild of Book Workers. Swarthmore College, Swarthmore, Pennsylvania. Sites and dates are listed at <http://palimpsest.stanford.edu/byorg/gbw>.
- May 29 Journée GUTenberg, “Distributions”, Paris, France. For information, visit <http://www.gutenberg.eu.org/>.
- May 29–31 Society for Scholarly Publishing, 24<sup>th</sup> annual meeting, Boston, Massachusetts. For information, visit <http://www.sspnet.org>.
- Jul 12–14 TypeCon 2002, Toronto, Canada. For information, visit <http://www.typecon2002.com>.

*Status as of 1 July 2002*

For additional information on TUG-sponsored events listed above, contact the TUG office (+1 503 223-9994, fax: +1 503 223-3960, e-mail: [office@tug.org](mailto:office@tug.org)). For events sponsored by other organizations, please use the contact address provided.

Additional type-related events and news items are listed in the Sans Serif Web pages, at <http://news.serifmagazine.com/>.

Owing to the lateness of this issue, please consider that all events shown for 2001 are included only “for the record”.

Jul 21–26 SIGGRAPH 2002, San Antonio, Texas. For information, visit <http://www.siggraph.org/calendar/>.

---

### TUG 2002

International Convention Centre, Trivandrum, India. For information, visit <http://www.tug.org.in/tug2002/>.

Sep 1–3 Tutorials: L<sup>A</sup>T<sub>E</sub>X; L<sup>A</sup>T<sub>E</sub>X to XML; METAPOST; the Text Encoding Initiative; T<sub>E</sub>X macro expansion.

Sep 4–7 The 23<sup>rd</sup> annual meeting of the T<sub>E</sub>X Users Group, “Stand up and be proud of T<sub>E</sub>X!”. For information, visit <http://www.tug.org.in/tug2002/>.

---

Sep 19–22 Association Typographique Internationale (ATypI) annual conference, Rome, Italy. For information, visit <http://www.atypi.org/rome2002/>.

Sep 24–25 First Annual Conference, Friends of St. Bride Printing Library, London, England. For information, visit <http://www.stbride.org/conference.htm>

Oct 12 UK TUG Autumn meeting, Nottingham University. For information, contact Dick Nickalls, [dicknickalls@compuserve.com](mailto:dicknickalls@compuserve.com).

Oct 14–17 Book History Workshop, Institutue d’histoire du livre, Lyons, France. For information, visit <http://ihl.enssib.fr>.

Nov 8–9 ACM Symposium on Document Engineering, McLean, Virginia. For information, visit <http://www.sdml.cs.kent.edu/doceng2002/>.

---

### 2003

#### TUG 2003

**Outrigger Waikoloa Beach Resort, Big Island, Hawai’i.**

---

**Jun 24-27: EuroT<sub>E</sub>X 2003: Back to typography. For more information see** <http://omega.enstb.org/eurotex2003>.

Jul 15–18 Beginning/Intermediate L<sup>A</sup>T<sub>E</sub>X class. For more information see <https://www.tug.org/tug2003/latexclass.html>.

Jul 20–24 The 24<sup>th</sup> annual meeting of the T<sub>E</sub>X Users Group. For information, visit <http://www.tug.org.in/tug2003/>.

---

Jul 27–Aug 1 SIGGRAPH 2003, San Diego, California. For information, visit <http://www.siggraph.org/calendar/>.

# TUG 2003: the Silver Anniversary – 25 years! – of T<sub>E</sub>X

The 24th Annual Meeting and Conference of the T<sub>E</sub>X Users Group

tug2003@tug.org

## Themes

- Resurgence of T<sub>E</sub>X & L<sup>A</sup>T<sub>E</sub>X
- pdfT<sub>E</sub>X, ConT<sub>E</sub>Xt, MetaPost, MetaFont
- T<sub>E</sub>X–XML Symbiosis, T<sub>E</sub>I, Digital Archiving
- Fonts & Graphics
- Installations & Management, CTAN
- Publisher & Prepress Dilemmas
- MacOS X TeX: New Kid on the Block

## Important Dates

| 2002                    |        | 2003                     |       |
|-------------------------|--------|--------------------------|-------|
| Abstracts due           | 18 Nov | First draft of paper due | 9 Feb |
| Abstracts accepted      | 18 Dec | Registration deadline    | 9 Apr |
| Early-lion registration | 31 Dec | Final paper due          | 9 Jun |

## Links

|                           |                                                                                                           |
|---------------------------|-----------------------------------------------------------------------------------------------------------|
| Homepage                  | <a href="http://www.tug.org/tug2003/">http://www.tug.org/tug2003/</a>                                     |
| News Mailing List         | <a href="http://www.tug.org/tug2003/news/">http://www.tug.org/tug2003/news/</a>                           |
| T <sub>E</sub> X Heritage | <a href="http://www.tug.org/tug2003/heritage/">http://www.tug.org/tug2003/heritage/</a>                   |
| Call for Abstracts        | <a href="http://www.tug.org/tug2003/callfor.html">http://www.tug.org/tug2003/callfor.html</a>             |
| Registration/Donations    | <a href="https://www.tug.org/tug2003/registration.html">https://www.tug.org/tug2003/registration.html</a> |



T<sub>E</sub>X Enthusiasts worldwide are invited to join us for a grand reunion to celebrate the accomplishments of T<sub>E</sub>X  
Polish up the old and learn about the shiny new directions in TeX!

July 20–24, 2003, Outrigger Waikoloa Beach Resort, Big Island, Hawaii  
[ Kona International Airport at Keahole (KOA) ]

## Institutional Members

- American Mathematical Society,  
*Providence, Rhode Island*
- Center for Computing Science,  
*Bowie, Maryland*
- Cessna Aircraft Company,  
*Wichita, Kansas*
- The Clarinda Company,  
*Clarinda, Iowa*
- CNRS - IDRIS,  
*Orsay, France*
- CSTUG, *Praha, Czech Republic*
- Florida State University,  
School of Computational Science  
and Information Technology,  
*Tallahassee, Florida*
- IBM Corporation,  
T J Watson Research Center,  
*Yorktown, New York*
- Institute for Advanced Study,  
*Princeton, New Jersey*
- Institute for Defense Analyses,  
Center for Communications  
Research, *Princeton, New Jersey*
- Iowa State University,  
Computation Center,  
*Ames, Iowa*
- Kluwer Academic Publishers,  
*Dordrecht, The Netherlands*
- KTH Royal Institute of  
Technology, *Stockholm, Sweden*
- Marquette University,  
Department of Mathematics,  
Statistics and Computer Science,  
*Milwaukee, Wisconsin*
- Masaryk University,  
Faculty of Informatics,  
*Brno, Czechoslovakia*
- Max Planck Institut  
für Mathematik,  
*Bonn, Germany*
- New York University,  
Academic Computing Facility,  
*New York, New York*
- Princeton University,  
Department of Mathematics,  
*Princeton, New Jersey*
- Springer-Verlag Heidelberg,  
*Heidelberg, Germany*
- Stanford Linear Accelerator  
Center (SLAC),  
*Stanford, California*
- Stanford University,  
Computer Science Department,  
*Stanford, California*
- Stockholm University,  
Department of Mathematics,  
*Stockholm, Sweden*
- University College, Cork,  
Computer Centre,  
*Cork, Ireland*
- University of Delaware,  
Computing and Network Services,  
*Newark, Delaware*
- University of Oslo,  
Institute of Informatics,  
*Blindern, Oslo, Norway*
- Università degli Studi di Trieste,  
*Trieste, Italy*
- Vanderbilt University,  
*Nashville, Tennessee*

## T<sub>E</sub>X Consulting & Production Services

Information about these services can be obtained from:

**T<sub>E</sub>X Users Group**  
 1466 NW Naito Parkway, Suite 3141  
 Portland, OR 97209-2820, U.S.A.  
 Phone: +1 503 223-9994  
 Fax: +1 503 223-3960  
 Email: [office@tug.org](mailto:office@tug.org)  
 URL: <http://www.tug.org/consultants.html>

### North America

#### Loew, Elizabeth

President, T<sub>E</sub>Xniques, Inc.,  
 675 Massachusetts Avenue, 6th Floor,  
 Cambridge, MA 02139;  
 (617) 876-2333; Fax: (781) 344-8158  
 Email: [loew@texniques.com](mailto:loew@texniques.com)

Complete book and journal production in the areas of mathematics, physics, engineering, and biology. Services include copyediting, layout, art sizing, preparation of electronic figures; we keyboard from raw manuscript or tweak T<sub>E</sub>X files.

#### Ogawa, Arthur

40453 Cherokee Oaks Drive,  
 Three Rivers, CA 93271-9743;  
 (209) 561-4585  
 Email: [ogawa@teleport.com](mailto:ogawa@teleport.com)

Bookbuilding services, including design, copyedit, art, and composition; color is my speciality. Custom T<sub>E</sub>X macros and L<sup>A</sup>T<sub>E</sub>X<sub>2 $\epsilon$</sub>  document classes and packages. Instruction, support, and consultation for workgroups and authors. Application development in L<sup>A</sup>T<sub>E</sub>X, T<sub>E</sub>X, SGML, PostScript, Java, and BC++. Database and corporate publishing. Extensive references.

#### Veytsman, Boris

2239 Double Eagle Ct.  
 Reston, VA 20191;  
 (703) 860-0013  
 Email: [boris@lk.net](mailto:boris@lk.net)

I provide training, consulting, software design and implementation for Unix, Perl, SQL, T<sub>E</sub>X, and L<sup>A</sup>T<sub>E</sub>X. I have authored several popular packages for L<sup>A</sup>T<sub>E</sub>X and `latex2html`. I have contributed to several web-based projects for generating and typesetting reports. For more information please visit my web page: <http://users.lk.net/~borisv>.

#### The Unicorn Collaborative, Inc, Ted Zajdel

115 Aspen Drive, Suite K  
 Pacheco, CA 94553  
 (925) 689-7442  
 Email: [contact@unicorn-collab.com](mailto:contact@unicorn-collab.com)

We are a technical documentation company, initiated in 1990, which time, strives for error free, seamless documentation, delivered on time, and within budget. We provide high quality documentation services such as document design, graphic design and copy editing. We have extensive experience using tools such as FrameMaker, T<sub>E</sub>X, L<sup>A</sup>T<sub>E</sub>X, Word, Acrobat, and many graphics programs. One of our specialties is producing technical manuals and books using L<sup>A</sup>T<sub>E</sub>X and T<sub>E</sub>X. Our experienced staff can be trained to use any tool required to meet your needs. We can help you develop, rewrite, or simply copy-edit your documentation. Our broad experience with different industries allows us to handle many types of documentation including, but not limited to, software and hardware systems, communications, scientific instrumentation, engineering, physics, astronomy, chemistry, pharmaceuticals, biotechnology, semiconductor technology, manufacturing and control systems. For more information see our web page <http://www.unicorn-collab.com>.

### Outside North America

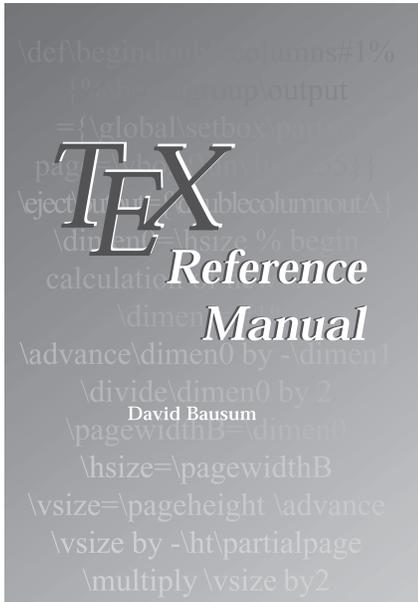
#### DocuT<sub>E</sub>Xing: T<sub>E</sub>X Typesetting Facility

43 Ibn Kotaiba Street,  
 Nasr City, Cairo 11471, Egypt  
 +20 2 4034178; Fax: +20 2 4034178  
 Email: [main-office@DocuTeXing.com](mailto:main-office@DocuTeXing.com)

DocuT<sub>E</sub>Xing provides high-quality T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X typesetting services to authors, editors, and publishers. Our services extend from simple typesetting and technical illustrations to full production of electronic journals. For more information, samples, and references, please visit our web site: <http://www.DocuTeXing.com> or contact us by e-mail.

# TEX Reference Manual

by **David Bausum**, *Lighthouse & Associates, Beloit, WI, USA*



The *TeX Reference Manual* is the first comprehensive reference manual written by a programmer for programmers. It contains reference pages for each of TeX's 325 primitive control sequences. Over 80% of its reference pages contain examples that range from simple to challenging. Each example is typeset verbatim in a style which is easy to read and experiment with. *TeX Reference Manual* also just typesets the example, so you can see what it makes, and explains how the example works. The description on each primitive's reference page is an annotated discussion of *The TeXbook's* treatment of the primitive. That means a TeX user will find it natural to move back and forth between the two books. One of *TeX Reference Manual's* innovative features is families. They simplify the search for the primitive which performs a particular task.

| Primitive Control Sequences |                |      |                          |
|-----------------------------|----------------|------|--------------------------|
| Family Name                 |                | Type | Description              |
| Box (29)                    | Logic (20)     | C    | Command (163)            |
| Character (16)              | Macro (20)     | D    | Derived Command (17)     |
| Debugging (25)              | Marks (4)      | IQ   | Internal Quantity (42)   |
| File I/O (13)               | Math (69)      | PI   | Parameter (integer) (55) |
| Fonts (5)                   | Page (13)      | PD   | Parameter (dimen) (21)   |
| Glue (12)                   | Paragraph (30) | PG   | Parameter (glue) (15)    |
| Hyphenation (11)            | Penalties (12) | PM   | Parameter (muglue) (3)   |
| Inserts (8)                 | Registers (11) | PT   | Parameter (token) (9)    |
| Job (11)                    | Tables (9)     |      |                          |
| Kern (7)                    |                |      |                          |

## CONTENTS

Preface

1. Families and Primitive Control Sequences.

2. Reference Pages for the Primitives.

**Appendix A.** Typesetting Verbatim Material.

**Appendix B.** Working with PostScript Fonts.

**Appendix C.** Typesetting Material in Two Columns.

Bibliography. Index.

February 2002 Hardbound, ISBN 0-7923-7673-0 390 pp.

EUR 108.00 / USD 99.00 / GBP 68.00

**Special Price offered to TUGBOAT subscribers:**

**EUR 97.00 / USD 90.00 / GBP 61.00**

*TeX Reference Manual* has appendices which provide a comprehensive discussion of: verbatim material, PostScript fonts, and two-column material. In particular, one word describes its font macros, elegant. The *TeX Reference Manual* is an invaluable tool for both the experienced and new users of TeX.



**ORDER TODAY!**

**ONLINE: WWW.WKAP.NL**

**Fax your order:**

USA: 781-681-9045

Rest of World: +31 78 6546 474

**Phone:**

USA: +781-871-6600

Rest of World: +31 78 6392 392

**Email:**

USA: [kluwer@wkap.com](mailto:kluwer@wkap.com)

Rest of World: [services@wkap.nl](mailto:services@wkap.nl)

*introducing*  
**TEXTURES<sup>®</sup> 2.0**

W I T H   S Y N C H R O N I C I T Y



AGAIN THE MACINTOSH DELIVERS A NEW T<sub>E</sub>X WITH A REVOLUTION IN HUMAN INTERFACE.

As computer power has advanced, the Macintosh has consistently been the leader in the human and humane connection to technology, and Textures has consistently led in bringing ease of use to T<sub>E</sub>X users.

First with Textures 1.0, the first truly

integrated T<sub>E</sub>X system. Then with Lightning Textures, the first truly interactive T<sub>E</sub>X system. Now, with Textures 2.0 and Synchronicity, Blue Sky Research again delivers a striking advance in T<sub>E</sub>X interactivity and productivity.

With Synchronicity, your T<sub>E</sub>X input documents are reliably and automatically cross-linked, correlated, or "synchronized" with the finished T<sub>E</sub>X typeset pages. Every piece of the finished product is tied directly to the source code from which it was generated, and vice-versa. To go from T<sub>E</sub>X input directly and exactly to the corresponding typeset characters, just click.

It's that simple: just click, and Textures will take you instantly and precisely to the corresponding location. And it goes both ways: just click on any typeset character, and Textures will take you directly to the T<sub>E</sub>X code that produced it. No matter how many input files you have, no matter what macros you use, Synchronicity will take you there, instantly and dependably.

*Improve YOUR performance:*

G E T   S Y N C H R O N I C I T Y

BLUE SKY RESEARCH  
317 SW ALDER STREET  
PORTLAND, OR 97204 USA



800 622 8398  
503 222 9571  
WWW.BLUESKY.COM