TEXniques

Publications for the TEX Community
Number 7

Conference Proceedings

TEX Users Group
Ninth Annual Meeting
Montréal, August 22–24, 1988

TEX Users Group
P. O. Box 9506
Providence, R. I. 02940, U.S.A.

TeX Users Group

The TeX Users Group (TUG) is a nonprofit association dedicated to the dissemination of information to the TeX community and to the education of the general public about TeX. These objectives are accomplished through the publication of a newsletter, *TUGboat*, and the conduct of conferences and TeX courses. TUG has about 3,000 members throughout the world: approximately 2,150 in the U.S., 150 in Canada, 475 in Europe, and 175 in other countries.

President
Bart Childs
Texas A&M University

Vice President
Richard Furuta
University of Maryland

Secretary
Alan Hoenig
John Jay College, CUNY

Treasurer
David Ness
TV Guide

Executive Director
Raymond Goucher
TeX Users Group

Program Committee Chairman
Dean Guenther
Washington State University

Program Committee
Shawn Farrell
McGill University

Dean Guenther

Christina Thiele

Proceedings Editor
Christina Thiele
Carleton University

Address:
TeX Users Group
P.O. Box 9506
Providence, R.I. 02940
USA

Phone:
(401) 751-7760

Electronic Mail:
TUG@SEED.AMS.COM

Printed and Bound by: 1-800-USA-BOOKS

Conference Proceedings

TeX Users Group
Ninth Annual Meeting
Montréal, August 22–24, 1988

© 1988 by the TEX Users Group. Copying any paper within is permitted as long as credit is given to the source, and copies are not made or distributed for direct commercial advantage. Authors retain their individual copyrights.

TEX Users Group
P.O. Box 9506
Providence, R.I. 02940, USA

The following trademarks appear in this publication:

AMPRO LB-186 is a trademark of AMPRO Computers, Inc. AMS-TEX is a trademark of the American Mathematical Society. Apollo is a trademark of Apollo Corp. APS μ5 is a trademark of Autologic, Inc. AutoCAD is a trademark of Autodesk, Inc. Bitstream is a trademark of Bitstream, Inc. CAP, CP-6, MULTICS, COMPOSE, and TEXT are trademarks of Honeywell Bull, Inc. CEO is a registered trademark of Data General. Compugraphic MCS 8400 is a trademark of Compugraphic Corp. Cricket Draw is a trademark of Cricket Software, Inc. DB-Library is a trademark of Sybase, Inc. dBaseIII is a trademark of Ashton-Tate, Inc. DEC, Digital, LN03, LN03-Plus, Rainbow, Runoff, VAX and MicroVAX are trademarks of Digital Equipment Corp. DVIHP, DVIPS, DVILASER/PS, and Preview are trademarks of ArborText, Inc. Epson is a registered trademark of Seiko Epson Corp. Excelerator is a trademark of Index Technology, Inc. FASTEX, FTEX and QKEY are trademarks of Norman Paul Consultants, 920 Commercial Street, Palo Alto, CA 94303. GEM is a trademark of Digital Research, Inc. Hercules is a trademark of Hercules Computer Technology. HiJaak is a trademark of InSet Systems, Inc. HP and LaserJet are registered trademarks of Hewlett-Packard Company. IBM and PC-DOS are registered trademarks of International Business Machines Corp. IDM and Intelligent Database Machine are trademarks of Britton Lee, Inc. IMAGEN, ImageStation, and IMAGEN Innovator are trademarks of IMAGEN Corp. Lasergrafix and QMS are trademarks of Quality Micro Systems, Inc. Lightning is a trademark of Borland International, Inc. Macintosh and QuickDraw are trademarks of Apple Computer, Inc. MacLink*Plus* is a registered trademark of DataViz, Inc. MacPaint is a trademark of Claris Corp. MAXview is a copyright of Aurion Tecnología, SA de CV. METAFONT and MicroTEX are trademarks of Addison Wesley Publishing Compan y. MS-DOS, Microsoft WORD, and Microsoft Windows are trademarks of Microsoft Corp. NBI System 8 is a trademark of NBI, Inc. Olivetti is a trademark of Olivetti Corp. PageMaker is a trademark of Aldus Corp. PanaScan is a trademark of Matsushita Electric Industrial Company, Ltd. Panasonic FX505 is a trademark of the Matsushita Electric Company. PC Paintbrush is a trademark of Z-Soft Corp. PCTEX is a registered trademark of Personal TEX, Inc. Peder is a trademark of Micro Programs, Inc. Personal-REXX is the copyright of Mansfield Software. POSTSCRIPT and Adobe are trademarks of Adobe Systems, Inc. PreTEX is a trademark of Robert L. Kruse. Prime 750 is a trademark of Prime Computers, Inc. ProKey is a trademark of RoseSoft, Inc. PSL/PSA is a trademark of Meta Systems, Inc. Pyramid is a trademark of Pyramid Technology Corp. Sanyo is a trademark of Sanyo Corp. Scan-Do is a trademark of Hammerlab, Inc. Stratus is a trademark of Stratus Computer, Inc. Sunview and Sun-3 are trademarks of Sun Microsystems, Inc. Tandy is a trademark of Tandy Corp. TEX is a trademark of the American Mathematical Society. UNIVAC is a registered trademark of Sperry Rand Corp. UNIX is a trademark of AT&T Bell Laboratories. Wang is a trademark of Wang Laboratories, Inc. WordPerfect is a registered trademark of WordPerfect Corp. WordStar is a trademark of MicroPro International Corp. Wyse is a registered trademark of Wyse Technology. X Windowing system is a trademark of MIT. Zephyr is a trademark of Zentec Corp.

Table of Contents

Introduction ... v

Production Notes ... vi

Mary McCaskill
Producing NASA Technical Reports with TEX ... 1

J. Tom Renfrow
Use of TEX in an Integrated System Development Environment ... 11

David Ness and James Slagle
TEX and Databases ... 25

Laurie Mann
Producing Manual Sets Using Single-Sourcing ... 31

Jean Pollari
Using TEX to Produce Government Standard Documentation ... 41

Eric Jul
Implementing TEX in a Production Environment: A Case Study ... 53

Peter Tonkin and Alex Warman
How and Why a Trade Typesetter Chose TEX ... 61

James D. Mooney
An Experience in Textbook Production ... 69

Robert L. Harris
Using TEX to Produce Kennel Club Yearbooks ... 83

Elizabeth Barnhart and David Ness
Layout for TEX ... 97

Bart Childs et al
Syllabi for TEX and METAFONT *Courses* ... 117

Berkeley Parks
TEX Tips for Getting Started ... 129

Alan Wittbecker
The Art of Teaching TEX for Production ... 149

Shawn Farrell
Choosing Between TEX and LATEX ... 155

Kazuhiro Kitagawa and Nobuo Saito
Mathematics Textbook Publishing with Japanese TEX ... 165

Jacques J. Goldberg
Approximate TEX for Semitic Languages 171

Michael J. Ferguson
TÊX is Multilingual 179

Kauko Saarinen
Experiences with TEX in Finland 189

Stephan von Bechtolsheim
Using the Emacs Editor to Safely Edit TEX Sources 195

Lynne A. Price
Using SGML and TEX for User Documentation 203

Ken Yap
DVI Previewers 211

Robert L. Kruse
PreTEX: Tools for Typesetting Technical Books 219

Mike Schmidt
CapTEX: Industrial Strength TEX 227

Paul M. Muller
FASTEX: A PC Text Editor and Front-End for TEX 235

Participants, 1988 TEX Users Group Meeting 255

List of Exhibitors 262

Introduction

The 1988 Annual Meeting of the TEX Users Group was held in Montreal, Canada, at McGill University, and was attended by some 180 people from as far away as Finland, Israel, England and Japan. There were 24 presentations this year, representing almost 250 pages in these *Proceedings*.

There was a very clear emphasis on TEX in a production environment: in-house document preparation, commercial applications, with case studies in both. Several presentations dealt with non-English applications, describing the attendant problems and solutions encountered. The integration of TEX with other programs is fast becoming a new and important aspect of the program, especially with respect to source material stored in databases. How to use TEX and how to teach TEX was the subject of two presentations; using TEX in an SGML context was also outlined in some detail. A presentation on previewers was the sole discussion on the more technical side of the program, illustrating the feeling that TEX the program has now achieved stability, and the focus is now shifting to its integration into already existing operations, and with other non-typesetting programs. Preprocessors to TEX also indicate a growing interest in making TEX more accessible, and therefore more easily integrated into environments where TEX output is desirable, but TEX expertise is not as likely to be aimed for.

Indeed, in presentation after presentation, the recurring themes were the high quality of TEX output, the program's flexibility, the fact that it allowed users to continue working on their own computers, with no concern for compatibility. Almost all the production/application presentations raised these points.

There remain some problems, most often in the area of font and printer compatibilities; table making seems to be a recurring area of concern. Several presentors spoke of the need for management to be firmly committed to the use of TEX, both in terms of support, and investment in training time. Interestingly, in conjunction with concerns about adequate and proper training, there were calls for TUG courses themselves to address the question of applied TEX courses, particularly to typography, font design, and the production process. Thus, as the user focus shifts, the user group is also being asked to help deal with this shift.

All in all, this year's conference, and these *Proceedings*, mark a significant change in direction, hinted at during last year's meeting in Seattle: TEX is coming of age, and is now being stretched and adapted to deal with a much wider range of applications than were perhaps initially envisaged by Professor Don Knuth. And the demands being made of it are being addressed, are being solved; the program not only works, but it works very well and seems to rise to each new leap forward or outward in scope and use. Something which we can all participate in, and enjoy.

Christina Thiele
Editor

Production Notes

Apple LaserWriter II
Berkeley Parks, *TEX Tips for Getting Started.*

Apple LaserWriter Plus
Ken Yap, *DVI Previewers.*
Robert Kruse, *PreTEX: Tools for Typesetting Technical Books.*

Autologic APS μ5 (Almost Computer Modern)
Mary McCaskill, *Producing NASA Technical Reports with TEX.*

IBM 3820
Participants, 1988 TEX Users Group Meeting.

IMAGEN ImageStation
Preliminary pages to the Proceedings.
David Ness, *TEX and Databases.*
Laurie Mann, *Producing Manual Sets from the Same Source.*
Eric Jul, *Implementing TEX in a Production Environment.*
Peter Tonkin and Alex Warman, *How and Why a Trade Typesetter Chose TEX.*
Elizabeth Barnhart, *Layout for TEX.*
S. Bart Childs et al, *Syllabi for TEX and* METAFONT *Courses.*
Alan Wittbecker, *The Art of Teaching TEX for Production.*
Shawn Farrell, Choosing Between TEX and LATEX
Kauko Saarinen, *Experiences with TEX in Finland.*
Stephan von Bechtolsheim, *Using the Emacs Editor to Safely Edit TEX Source.*
Lynne Price, *Using SGML and TEX for User Documentation.*
Mike Schmidt, *CapTEX: Industrial Strength TEX.*
Kazuhiro Kitagawa, *Mathematics Textbook Publishing with Japanese TEX.*

IMAGEN Innovator
Paul Muller, *FASTEX: A PC Text Editor and Front-End for TEX.*

LN03A
Jacques Goldberg, *Approximate TEX for Semitic Languages.*

LN03-Plus
James Mooney, *An Experience in Textbook Production.*

NEC Silentwriter LC890
Robert Harris, *Using TEX to Produce Kennel Club Yearbooks*

QMS 810
Tom Renfrow, *Use of TEX in an Integrated System Development Environment.*

QMS-1200
Michael Ferguson, *TÊX is Multilingual.*

QMS Lasergrafix 2400
Jean Pollari, *Using TEX to Produce Government Standard Documentation.*

Producing NASA Technical Reports With TEX

MARY K. MCCASKILL

Mail Stop 149
NASA Langley Research Center
Hampton, VA 23665-5225
mary@teb.larc.nasa.gov

ABSTRACT

Since 1984, the Technical Editing Branch (TEB) at NASA Langley Research Center has been formatting NASA research reports with TEX. These highly technical reports contain complicated tables and numerous mathematical equations that are difficult to format. TEX was chosen as the basis for Langley's in-house typesetting system because of its excellent mathematical typography and its device independence; its excellence in typography in general, for example, the line-breaking algorithm, was an added bonus.

TEX was initially installed on a Prime computer at Langley and TEB personnel input files remotely using word processing workstations as dumb terminals; no preview capability was available. User adaptation to TEXing documents on a computer as opposed to using a WYSIWYG word processor and development of expertise with TEX were difficult problems to surmount. With improved hardware, experience, and perseverance, we are now typesetting our reports in a highly efficient manner. Production figures (minutes per page) are presented. Discussed are the successes and problems during implementation of the TEX-based system, how problems with hardware, training, and technical expertise were solved, and how word processing personnel were converted to typesetters. The paper concludes with a "wish list," what we would like to see in the near and distant future.

Introduction

The Langley Research Center is one of several installations of the National Aeronautics and Space Administration, which is responsible for government-sponsored aerospace and aeronautics research. Since its inception in 1917, as the National Advisory Committee for Aeronautics (NACA), the NASA Langley Research Center has published results of its research—in NACA or NASA reports and in journals or other external literature. As early as 1935, Langley had

a technical editing department responsible for editing and production of manuscripts for printing. Equipment for production has included simple typewriters, proportional-spacing typewriters with changeable keys, word processors in conjunction with daisy-wheel printers, and most recently, a digital phototypesetter and laser printer.

The Langley Technical Editing Branch typically produces 150–160 research reports per year authored by members of 25 research divisions. The technical complexity of these reports renders them difficult to produce, whatever system is used. The typical report contains multilevel equations, complex tables, and many illustrations. Word processing aided greatly in allowing quicker production and correction of manuscripts, but the quality of the daisy-wheel printer output was only marginally satisfactory for printing.

In the early 1980's, in-house typesetting was initiated to improve manuscript quality from what was available with word processors. An extensive investigation of available typesetting systems for technical manuscripts resulted in the choice of a digital phototypesetter driven by a TEX-based system. TEX was chosen as the basis for Langley's typesetting system because it offered the only hope of typesetting the NASA reports without losing productivity or quality. Neither the editors nor the technical typists had to become expert in mathematical typography; no other system that we evaluated automated mathematical typography as well as TEX does. TEX could produce the equations written by Langley researchers with the least difficulty and time. At this time we were just beginning to explore transferring draft manuscripts electronically from the author for correction and formatting. The 25 Langley research divisions that submit manuscripts for publication each choose their own document-processing system. TEX's device independence offered the potential for simplifying electronic transfer between all the disparate systems at the center.

A TEX-based system was also the least expensive to acquire. The only hardware initially purchased was the digital phototypesetter itself; input terminals and a computer were already available.

Initial Implementation

The Technical Editing Branch approached the installation of a TEX-based typesetting system with some naiveté. TEX had been chosen on the basis of Knuth (1979) (the preliminary version of the *TEXbook*) without actual experience using it. In addition, the lack of maturity of the laser printer and terminal technology made purchase of proofing and screen preview devices difficult. Thus the initial implementation of TEX was somewhat crude; it was also a pioneering achievement. A management-oriented office with little experience with typesetting or computers were the first to install and use TEX routinely at Langley. We were also one of the first to install it in a production environment. Recognition of the advantages of TEX for technical typography and commitment

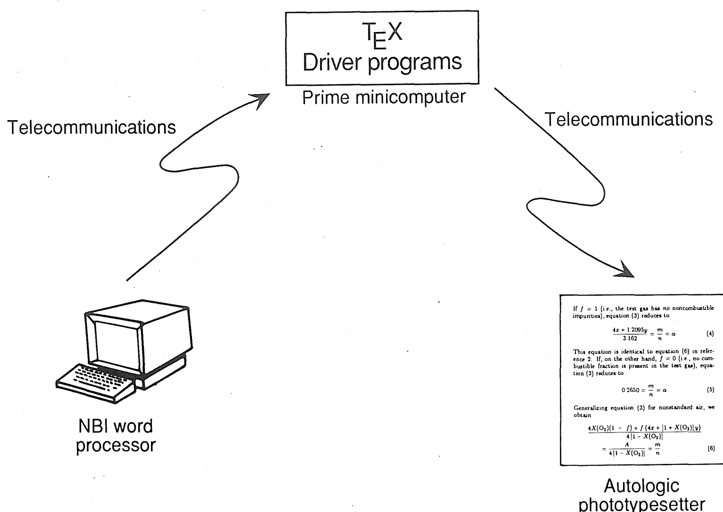


Figure 1. Initial implementation of TeX at Langley Research Center.

to device independence has resulted in a typesetting system well-suited to Langley's needs.

Hardware

TeX was installed on an existing minicomputer, a Prime 750, at the central computing facility at Langley. A device driver for the phototypesetter, an APS-Micro 5, was also written and installed on this minicomputer. The phototypesetter and the minicomputer were connected via a telecommunications network. An existing word processing system, an NBI System 8, was interfaced with the minicomputer also over the telecommunications network. The word processor workstations served as dumb terminals creating ASCII input files, which were sent to the central computing complex, TeXed, and returned to the phototypesetter over the network.

This somewhat simplistic system, shown in Figure 1, had some severe drawbacks. The transmission of files via telecommunications was slow (1200 baud) and error prone. After the input files arrived at the central computing complex, TeX ran very slowly, particularly at times of peak usage, on the shared minicomputer. Then, because no proofing device or screen preview was available, the only way to see the pages was to typeset the dvi file from the phototypesetter, a time-consuming, expensive, and laborious process. The typesetters used the excellent text editor available on the word processing system for initially formatting the document file, but for minor corrections and debugging on the minicomputer, a line editor was available. They found this line editor very unattractive and consequently preferred to edit the file on the word processor and re-send it to the minicomputer.

User Adaptation and Training

The hardware difficulties described above made adaptation to the new production system even more difficult for users, who were technical typists accustomed to a menu-driven word processor. The lack of interactive WYSIWYG display was particularly hard to adapt to. Also, to format the complex Langley reports, a higher level of expertise with TEX was required than had been anticipated. These technical typists not only had to learn TEX but also had to adapt to using a computer for the first time.

The system analyst who installed TEX and set up the system made the interface between the word processor and the computer as user-friendly and foolproof as possible. She wrote a macro set to produce the preferred double-column text format and ruled tables. She also taught a beginning TEX course to the technical typists. In early 1984, we began typesetting technical reports.

Development of Expertise

Teaching the technical typists to use TEX on a computer was not the only training requirement in implementing the new typesetting system. Editors, typists, and proofreaders all had to learn basic typography.

Very early we encountered the problem of combining computer, publishing, and TEX expertise. It was difficult for our systems analyst to understand why the editors and proofreaders would not tolerate widow and orphan lines, unconventional page breaks, and so forth. Likewise it was difficult for the editors to understand why all the fonts available on the phototypesetter were not available through the TEX system. It became obvious that some liaison between the three areas of expertise needed to be established. The editors and proofreaders needed familiarity with the capabilities of TEX in order to communicate their requirements to the typesetters and also to ensure that they were not requesting the impossible. Being a technical editor with interest in typography and computers, I became that liaison.

An in-depth understanding of TEX was developed slowly. At first no one in the organization had experience using TEX. Such experience was difficult to find in those days. The systems analyst wrote the initial macro set and taught the beginning TEX course without formal training. She learned by reading the *TEXbook* (Knuth 1984) and studying information that came with the program.

As each new formatting problem came up, the typesetters had no source for a solution. Either the systems analyst or the liaison editor had to educate herself (again from the *TEXbook*) to solve the problem, perhaps write or adjust a macro, and teach the typesetters the proper approach. Lack of screen preview made this trouble-shooting very slow and difficult. This situation slowed training; productivity statistics showed that the typesetters were still on a learning curve a full year after implementing TEX.

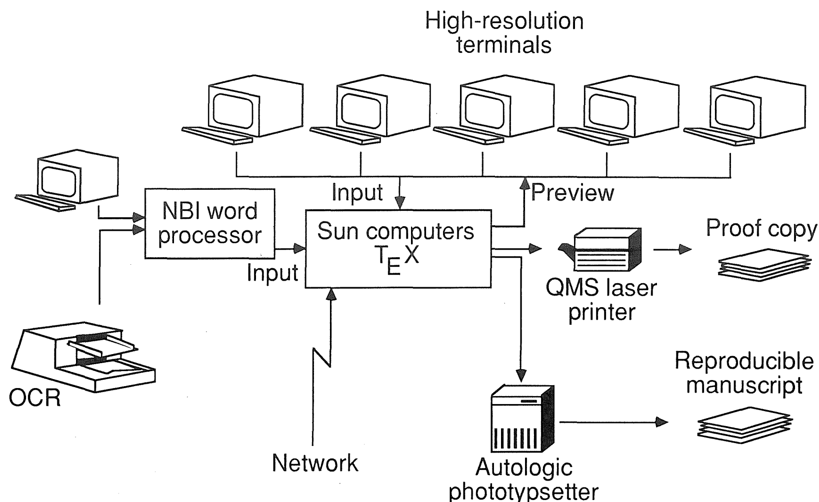


Figure 2. Present implementation of TEX in the Langley Technical Editing Branch.

The result of these difficulties with hardware, user adaptation, and training was an alarming decrease in productivity and a severe backlog of work, an intolerable situation in a production environment.

Solutions to Difficulties

Since 1985 the problem of productivity has been solved. With the present typesetting system shown in Figure 2, the four typesetters are typesetting reports faster than they typed them prior to 1984. The problems associated with hardware, that is,

1. Slow transmission of input files
2. Slow execution of TEX
3. No screen preview or quick proofing
4. Awkward file manipulation

have been solved by acquiring a laser printer for proofing and a dedicated computer system for typesetting in the Technical Editing Branch. The UNIX computer system consists of Sun-3 workstations, with high-resolution screens. Moving to the UNIX operating system has necessitated further training of users, and having our own computer has required more expertise in computer system administration.

A particularly important enhancement has been a preview program that displays the output from TEX on the screen, so that "what you see" is almost

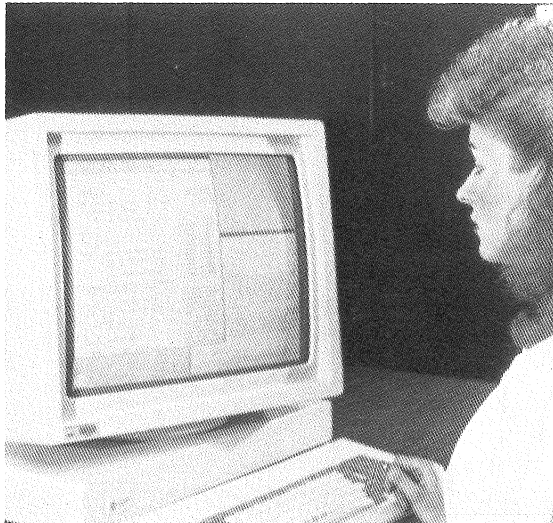


Figure 3. TEX ASCII file and typeset page displayed together on high-resolution screen.

"what you get" from the digital phototypesetter. The windowing environment also allows input files to be displayed alongside the typeset results, as shown in Figure 3.

The users adapted to the new computer system and learned TEX, UNIX, and so forth primarily through perseverance. They began using the system with only minimal formal training. When a new challenge or question came up, they met it as best they could, sought assistance when necessary, and learned from the experience. The original technical typists have become typesetters and have developed a body of knowledge of TEX and experience TEXing NASA technical reports. These employees have been able to pass on this knowledge and experience very quickly to new typesetters, who have become proficient and productive in approximately 2 months.

When novices are being trained on a system, whether a publishing program or some other system, they need an expert to whom they can turn for answers and help. Initially, we did not have such a resource and thus training was unacceptably slow.

The Technical Editing Branch computer system is going to continue to require significant technical knowledge and experience in system administration, networking, TEX, and publishing systems so that the productivity of the branch can be maintained and enhanced. Computer technology obviously has great potential to enhance the efficiency and productivity of publishing professionals, but only people who know how to use the technology and how to produce high-quality mechanicals for printing can make it work.

Present Status

As mentioned before, the research divisions at Langley each use text-processing systems that suit their individual needs. Thus numerous document-processing systems exist on various computers at Langley. The Technical Editing Branch (TEB) has been actively encouraging the use of TEX, since it can be installed on nearly any computer for very low cost. Although we are receiving many inquiries concerning TEX, its use, and its availability, we are not yet receiving many drafts that have been formatted with TEX. However we can accept ASCII files from several sources (network, PCs, word processors). Thus the typical draft is transferred to the TEB system without TEX formatting. The text of a document is rarely re-keyed, but the mathematics and tables almost always must be re-keyed.

For the typical rough draft, the standard procedure is to

1. Make changes marked in the draft either by extensively editing the file that has been transferred from the author's system or by entirely re-keying the draft
2. Enter TEX formatting, math, and alignment macros into the file (a standard macro set[1] has been defined on our system to produce the preferred double-column format)
3. Print out a proof copy from the laser printer
4. Correct the document twice, once after proofreading in TEB and once after author review
5. Typeset reproducible copy of text, tables, and captions from the phototypesetter
6. Paste up the illustrations

Table 1 lists the productivity statistics for the 128 reports produced by the four typesetters during the past year. The numbers in the table are the average time in minutes required to typeset the material from an 8½-inch by 11-inch page of double spaced, typed (usually) draft. All these reports have technical material within the text, that is, Greek symbols, mathematical characters, subscripts, and superscripts. Of the 128 reports, 20 had from one to four displayed equations per page. To prepare the text of these 20 reports required an average of 28 min/page, only 6 min/page more than the average for all 128 reports. As with most typesetting systems, tables offer the most room for productivity improvement. Typesetting a text page requires less than one-third of the time required to typeset a table page.

[1] This macro set will be documented in a forthcoming NASA Technical Memorandum.

Table 1. Productivity Statistics During Past Year

	Average time required to prepare rough draft page, min
Text (typeset)	22
Table (typeset)	73
Figure (captions typeset and pasted up)	5
Correct all pages (two correction cycles)	16
All page types	21

Future

At present we are evaluating TEX-based desktop publishing software in an effort to lower the cost of producing technical illustrations for printing. Our goal is to import graphics electronically from as wide a variety of computer graphic systems as possible, embed these graphic files in text which has been initially formatted with TEX, and print out the finished document at a resolution sufficient for high-quality printing.

In participating in the implementation of the TEX system in TEB, I have developed several concerns:

1. I have been known to say, "If I had never heard the word *font*, it would have been too soon." Publishers, and certainly graphic designers, are going to be happy only with a choice of typefaces. Setting up font families other than Computer Modern has been extraordinarily difficult. If we had known the trials of using the fonts available from the Autologic phototypesetter with TEX, we may not have chosen TEX. This is a severe problem.

2. With the new TEX-based software that is coming out (TEX translators, TEX math packages, desktop publishing packages), are document files produced on one TEX-based system going to be portable to other TEX-based systems?

3. How are technical editors going to fit into this publication system? Figure 4 illustrates how much of the Langley publication system is automated. The editing function, totally unautomated at present, exists in the midst of the other functions. Computerized editing tools (for example, prose analyzers) are now available, but will they work with TEX-formatted files? Some work on the part of a programmer was required to allow the UNIX spell command to work with TEX files.

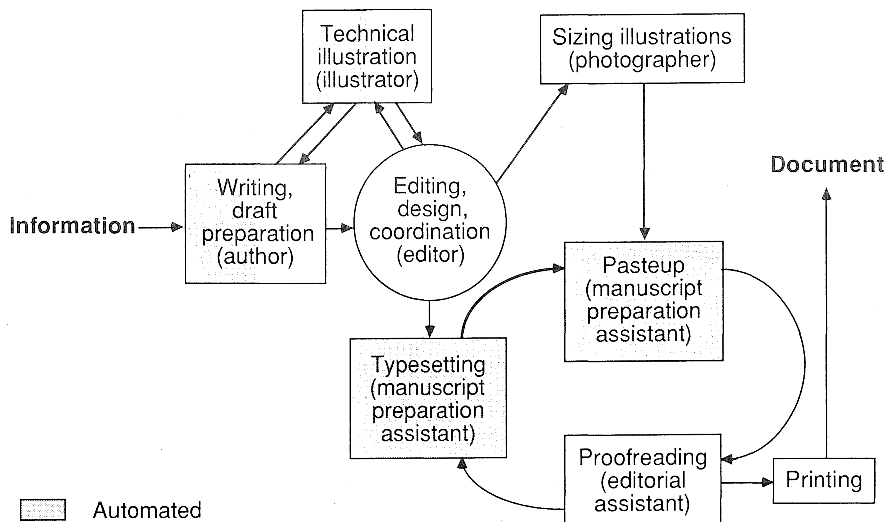


Figure 4. Langley publishing system—functions and personnel.

Concluding Remarks

Installation of a TEX-based typesetting system at NASA Langley Research Center has resulted in a system well-suited to the technical typesetting needs within the Langley Technical Editing Branch. TEX has offered the following advantages:

1. When adequate sources of TEX training and expertise became available, personnel were able to learn to typeset technical material quickly; Langley typesetters become productive in approximately 2 months.

2 The productivity possible with the system is very satisfactory. The appearance of Langley reports has been greatly improved over the output from daisy-wheel printers, while the reports are actually being produced in less time than they were previously with a word processing system.

3. The automation of mathematical typography has simplified the tasks of editors and proofreaders in marking and checking mathematics as well as the task of typesetters in formatting it. Not only does TEX produce outstanding mathematical typography, but also it excels in other typographic areas, for example, line breaking, hyphenation, and justification.

4. Device independence offers hope of attaining a standard text formatting system at Langley, where 25 research divisions each have a document-processing system of their choice.

The only disadvantage found in using TEX at Langley has been the difficulty with using font families other than Computer Modern.

On the basis of our experience in using TEX in a production environment, three additional comments are in order:

1. Because TEX is so device independent, one can scrimp on hardware and still attain a working system. However, such a system is not necessarily a productive one.

2. To install and maintain a TEX typesetting system requires expertise in computer systems, TEX, and publication practices. A real effort must be made to coordinate knowledge in all three areas.

3. TEX works best in an environment where the same format is used in most documents. When page layout changes often, too much time and expertise are required to constantly set up new formats.

Wish List

Imagine a bottle containing a TEXnical genie who could make all TEX wishes come true. Were I to rub the bottle, my three wishes would be

1. A program that provided a self-paced tutorial on using TEX. Short courses are not as effective as prolonged study with much practice mixed in.

2. An easy way of setting up the TEX-required files for any font.

3. Software that would assist technical editors in their tasks and that could be easily used on files formatted with TEX.

Actually, what I really want is a publication system that uses all the potential capability of computers to assist in (1) writing a document, (2) editing, analyzing, and revising it, (3) beautifully formatting it, even the difficult technical parts, in the typeface of my choice, (4) producing precise, artistic illustrations, (5) combining all its elements—text, tables, and illustrations—in the most useful and attractive layout, and (6) printing it precisely for distribution. TEX has a definite place in this system, but much work lies ahead in combining technology in the various areas to produce a truly complete computerized publication system.

Bibliography

Knuth, Donald E. *TEX and Metafont—New Directions in Typesetting.* Bedford, Mass.: Digital Press and American Mathematical Society. 1979.

Knuth, Donald E. *The TEXbook.* Reading, Mass.: Addison-Wesley. 1984.

Use of TEX in an Integrated System Development Environment

J.T. Renfrow

Jet Propulsion Laboratory
California Institute of Techology
4800 Oak Grove Drive
Pasadena, CA 91109
trenfrow@jplpds.jpl.nasa.gov

ABSTRACT

The Planetary Data System (PDS) is a software-intensive system being developed to archive planetary science data from all NASA missions. TEX is used as one of the tools in PDS' Integrated System Development Environment. The documentation produced using TEX includes system (i.e., software, database, user interface) development documents, user manuals, status reports, configuration management reports, and visual presentations. The generation of all this material is accomplished primarily by means of a variety of automated document generation techniques—database management systems, CASE tools (e.g., PSL/PSA and Excelerator), text manipulation languages (awk), and graphics editors. This environment is composed of heterogeneous hardware (IBM PCs, Macintoshes, Apollos, VAXes) connected via a national network. A set of methodologies has been implemented to allow the smooth integration of all these tools across this distributed development environment. The programs, procedures, and macros used are also discussed.

Introduction

The National Aeronautics and Space Administration (NASA) has sent many spacecraft to various planets in our solar system, in addition to launching numerous satellites to study our planet Earth. The Jet Propulsion Laboratory (JPL), a part of the California Institute of Technology, has the unmanned exploration of the solar system as one of its main missions. The engineering, scientific, and technological challenges to send a satellite to a distant planet, to have that satellite make meaningful measurements of the state of the planet, and to return that data to earth are extremely significant and complex, but the final value to the mission is achieved only if scientists can get access to, understand, and use the data acquired from a planetary exploration mission.

The goal of the Planetary Data System is to develop a cost-effective means to catalog, archive, and distribute useful, easily interpretable planetary science data, and the supporting data and computer programs for the planetary science community. This goal is being realized through three efforts—technology evaluation, standards development, and system development. The project is managed by JPL but involves the work of eight other academic institutions and government agencies. A distributed architecture is the model for the system, with JPL serving as the central node and the other nodes functioning as remote discipline-specific nodes. The institutions are all connected via several national electronic networks and communication between nodes is almost exclusively via these networks.

This paper focuses primarily on the requirements and design phases of the system development life cycle. The project recently completed its Critical Design Review and Version 1.0 is scheduled to be operational in November 1989. One system prototype was introduced in November 1987, and another prototype will be introduced in November 1988.

In order to develop a cost-effective system, the PDS uses a very automated approach to system development. TeX is used to produce all official PDS documentation. Various methodologies and other tools have been combined into an integrated system development environment to allow the efficient development and documentation of the PDS. Further information on these methodologies or machine-readable copies of any PDS-developed macros or software discussed in this article can be obtained by writing to the author.

The next section summarizes the guiding philosophy for this project as it relates to document production. The next four sections describe the integrated system development environment and explain the function TeX serves in that environment. The section on the system development environment describes the components of the environment: hardware, software, standards, methodologies, and staff. The section on TeX support discusses how and why TeX was introduced into the project and how it is currently supported and used. The section on document production processes describes the various classes of documents that are produced by the PDS and the methodologies/production strategies associated with documents in each class. The penultimate section presents the conclusions of our work as it relates to TeX. The last section of the paper presents the bibliography for the paper.

Project Philosophy

The following philosophical assumptions or guidelines have been used in developing the document production approach for the project:

1. The appearance of documentation as well as the content of documentation is important in conveying the project's message to its audience.
2. Tools should be developed when appropriate. "If you do a task less than five

times, don't automate it. If you do a task five or more times, build a tool to help you. If you build a tool more than five times, build a tool to help you build tools."

3. Not all staff on the project need the same level of training or expertise with all the tools. Try to make the tools as simple to use as possible.
4. Advance planning is essential to build databases that will contain all the information that will be needed in all the documents.
5. Since the project involves many geographically distributed components the creation of a comprehensive set of development and documentation standards is essential.

System Development Environment

The PDS system development environment is highly distributed. The work is done at nine nodes, one at JPL and the remainder at eight other institutions.[1] The node at JPL consists of two development groups and three science groups, in addition to groups working on standards and technology. The nodes at the other sites are principally science groups, developing science databases, and science data retrieval and display software.

1. Hardware

There are different types of computer hardware at the various nodes. Every node has a VAX computer and this is the target architecture for the operational PDS system. At the nodes (including JPL) there are local networks involving IBM PCs, Apple Macintoshes, and Apollo workstations. Not all the computers are connected via these institutional local networks but all have access to the national networks connecting PDS (Telenet and SPAN). Some of the nodes have PDS-purchased Britton Lee Intelligent Database Machines (IDM). All the nodes have laser printers and most of the nodes have printers which can interpret PostScript code.

2. Software

The software used in system development includes the usual compilers (the PDS is using FORTRAN and C) and text editors/word processors. In addition to these standard development tools, there are tools which have been used extensively during the requirements and design phases of the project. These include:

---

[1] Washington University (St. Louis), U. S. Geological Survey (Flagstaff), University of California at Los Angeles, University of Iowa (Iowa City), MIT, Applied Physics Labs (Johns Hopkins University), University of Colorado, and University of Hawaii.

2.1 Document production tools

TEX is the principal tool used for document production. While a few of the remote nodes produce some documents using automated systems other than TEX, all official PDS documents are produced using TEX.

2.2 Computer aided software engineering (CASE) tools

PSL/PSA (Problem Statement Language/Problem Statement Analyzer) is a software package used for automated analysis, design, and documentation. PSL/PSA supports a flexible model of information systems and allows one to capture information about the various aspects of an information system (information about processes, inputs, outputs, user interfaces, data items, etc.). While PSL/PSA produces many standard reports for use in analysis and design and documentation, PDS has been using a companion tool to PSL/PSA, Text Generator (TG), which allows the production of customized documentation and reports. These two packages are used to store the entire system model for the PDS. They run on the VAX and Apollo systems at JPL.

Standard software-based and hardware-based database management systems (dBase III and the Britton Lee IDM) are also used to keep track of the data for certain parts of the PDS system model. In particular the PDS Data Management Team maintains the entire data dictionary for all science and system data on the Britton Lee IDM. Conversion routines have been developed to share this data with the PSL/PSA processor.

Excelerator is a PC-based tool which can be used for automated drawing, printing, and analysis of data flow diagrams, structure charts, and data modelling diagrams. It has been used to produce all data flow diagrams and structure charts for our development work. Excelerator does not currently produce these diagrams in PostScript code. A program has been written to translate the Excelerator output metafile to PostScript code. Another program has been obtained from a custom software house to translate the text-based structural information from Excelerator to PSL/PSA input. In this way the Excelerator and PSL/PSA databases can be kept consistent. A software package is used to send PostScript and textual files between the PC and Apollo environments.

2.3 Utilities

Two graphics drawing programs, Cricket Draw and Adobe Illustrator, are used to produce graphics which are not produced by Excelerator. These packages are used both because they contain powerful graphics tools and because they can easily be made to output very intelligible PostScript code. The programs are run on a Macintosh and the results communicated to the PC environment using a software product, MacLink*Plus*, which does all necessary file format conversions.

Text manipulation languages, such as `awk` and `sed`, are used to convert text between non-TEX formats and TEX formats. These two languages are used for

frequently re-occurring and standard types of text modifications. ProKey, a PC-based keyboard macro package, can be used in conjunction with a word processing program such as WordPerfect to perform these same types of format changes when they are done on an *ad hoc* or interactive basis.

3. Standards

The PDS has adopted or created a number of system/data/software development standards based on the JPL software development standards. These have been documented in the *PDS Software Management Plan.* The standards for the work at the central node are more stringent than those required of the remote nodes, and the requirements on each are carefully delineated in the document. The reason for more complex standards at the central node is that the code being developed for the central node falls within a higher standards level by the JPL classification of software. Because of the widely distributed nature of this project, the systematic application of the standards has been extremely important. It has tended to ensure uniform and compatible software development and software products.

4. Methodologies

The PDS has adopted three methodologies for use with the requirements and design phases of the system development. During the requirements phase of the project Yourdon's Structured Analysis methodology was used (DeMarco 1979). During the design phase of the project, Yourdon's Structured Design methodology was used (Page-Jones 1980). The database was developed using extended Entity Relationship modeling techniques as well as internal PDS-developed methodologies.

5. Staff

The PDS staff consists of engineers, development engineers and programmers, document production personnel, and scientists. The core development team at the central node has had experience with the systematic application of development methodologies. All other members of the project have gradually come to realize the advantages (as well as the extra effort) associated with these automated systematic design approaches. The management of the PDS has actively supported and encouraged the use of these methodologies.

TEX Support

TEX was chosen for this project for several reasons, the principal being that the PDS project manager was familiar with its power and greatly encouraged its use. A second reason arose from the distributed nature of the project. Textual information had to be sent from one node to another (generally in the form

of electronic mail messages) to generate reports and documents. A variety of word processors and text editors were being used. There was no common set of control sequences associated with these word processors and even if there were, these control sequences could not be easily incorporated into electronic mail messages. Thus the only standard that could be required was the transmission of pure ASCII text. A third reason for the adoption of TEX was that very large documents (500 to 1,000 pages) would have to be produced based on an automated extraction of text from databases and there would not be enough time for manual manipulation and adjustment of text. The UNIX-based typesetting tools were not considered because the use of UNIX is not widespread in the PDS community. A fourth reason was that TEX was a very inexpensive (relatively speaking) yet extremely powerful product.

TEX has been installed on almost all the PCs at the central node (JPL has a site license for a micro-based version of TEX) and exists on many of the workstation/minicomputers at the remote nodes. It has also been installed on the VAX at the central node and the Apollo workstation. We have TEX supplied by Kellerman and Smith (VAX), Arbortext (Apollo), Addison-Wesley (MicroTEX), and Personal TEX (PCTEX). We use Preview from Arbortext. For print drivers we have the Imagen driver from Kellerman and Smith, and PostScript and HP drivers (DVIPS and DVIHP) from Arbortext.

There are various levels of TEX expertise among the project staff. At most of the remote nodes at least one member of the programming staff has become somewhat familiar with TEX. At the central node most members of the staff follow directions for document preparation and use of TEX conventions and macro packages supplied to them by other PDS staff. Another group of staff members have learned enough TEX so that they can generate the appropriate TEX commands as they create their document generation programs. There is a small staff on the project at the central node which is in charge of document production. They maintain the documentation databases and actually perform the production runs, which means that they have become very good in dealing with TEX's error messages. Finally there is one TEX wizard who writes widely-used macros, gives classes in TEX for the staff, and consults on TEX as necessary. Some of the documents require some sophisticated uses of TEX (e.g., modifying the output routine) and he helps in the development of these uses.

Training on TEX has evolved. Initially the classes consisted of lectures in which the instructor explained the principles of TEX and showed examples of TEX code to produce documents. There were nine classes, one a week, and each lasted two hours. No homework was given or required. Based on the instructor's experiences at a course from Adobe Systems on PostScript, the method of instruction was changed. The classes are now very interactive, with each student having a computer terminal with TEX on it as the class is taught. The class size is usually restricted to five or six students. A TEX concept or technique is explained and several examples are explained by the instructor. The students

are then asked to complete several exercises based on the examples. The raw material for the exercises (text and macros to be manipulated and changed) are supplied to minimize the data entry delays. The instructor works with the students so that they actually complete the exercises. Answers for all the exercises are always provided after the lecture. By making the instructional style interactive and immediate the learning process is active rather than passive.

The topics covered in the course are only those that are needed to create most of the TeX macros needed for PDS documents. The topics include: basic TeX wisdom (e.g., many spaces collapse to one space), handling of special characters (&, %, #, etc.), fonts, glue (how to skip vertically and horizontally), hboxes and vboxes, shaping of paragraphs, tables, and the creation of simple macros. The work on hboxes and vboxes is considerable since these are used extensively in PDS documents. Topics such as TeX for math, complicated macros, output routines, marks, and the reading and writing of files are not discussed.

Document Production Processes

There are several categories of documents that are produced by the PDS. Some of these have different development processes and these will discussed separately. There are, however, some widely-used document development processes which will be discussed first. Figure 1 presents the documentation tree for the project.

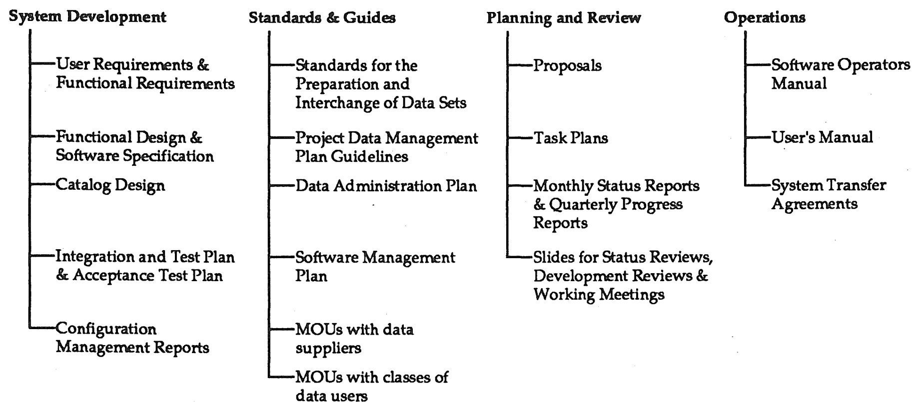


Figure 1: PDS Project Documentation Tree

Many of the documents produced by the PDS have the following characteristics—a title page, a table of contents, several chapters with each chapter containing sections, subsections, etc., and zero to many appendices, again with sections, subsections, etc. Since this form of documentation is so ubiquitous, a set of TeX macros was created to produce them. These macros were

designed to "word wrap" arbitrarily long titles, section headings, figure and table titles, and table of contents entries. Additional macros can be added to handle special formatting needs. The features of this standard package are:

1. Creation of chapters or appendices – Each chapter or appendix begins on an odd-numbered page. An extra page is inserted at the end of the previous chapter if necessary. The title of the chapter is included in the footer line. An author for the chapter can be included also. Title and page number information is written to a table of contents file.
2. Creation of sections, subsections, etc. – Up to 15 levels of sections can be given. Section numbering is automatically calculated. Table of contents entries are entered.
3. Figures and tables – Captions and numbers for figures and titles are automatically generated. Table of contents entries for each are created.
4. Verbatim text – Verbatim text can be entered. This allows for blank lines, lines starting with spaces, etc.
5. Automated list management – Lists with up to four levels of indentation are handled. Numbering is done automatically so rearrangement of lists is easy.
6. Table of contents generation – After the document has been TEXed, another macro package is called which generates the complete table of contents from files generated by TEX in the document production process. The table of contents includes separate parts for the main chapters of the document, the appendices, the list of figures, and the list of tables.

The PDS has developed a document describing the writing conventions to be used in PDS documents. This includes capitalization conventions, word use conventions (e.g., "data base" versus "database"), TEX conventions, and editing conventions.

If someone is producing a document and these macros are sufficient, then he can produce the document and "TEX" it on his own system. If it is a PDS official document, however, then a documentation production schedule for the document is prepared, including a review schedule. The document production plan clearly identifies the people involved in the production process (i.e., writers, programmers, producers, reviewers).

1. Project Development Documents

The automated production of system development documents (i.e., producing documents from databases) is done principally at the central node. The approach used in producing these documents is to extract information from a database and build a TEX file from this information.

The development of a system model which can be implemented via a database and then used to hold all information associated with the system being con-

structured is a very complex process. The details of this process will not be described in this article; it is described elsewhere (Childs 1988). The model must store all information as it is created, and must capture all the relationships between the various pieces of information. In building the model, one has to understand all the information needed from the model either for analysis of the requirements or design, for the generation of prototyping software, or for the documentation of various components of the system.

For example, consider a data item which is a part of the PDS Catalog (a catalog of scientific datasets). This data item has attributes such as data type, length, default value, value constraints, and measurement units. In addition it may be related to other entities and elements of the relational database schema and these relationships will have to be documented in the *PDS Catalog Design Document.* The data item may be called by software and this relationship must be specified in the *PDS Software Specification Document.* The presentation of this data item to the user via the PDS user interface system and the menus and display screens which use this item must be documented in the *PDS User's Guide.* All these relationships are captured for this data item. Other items of the model have equally complex relationships. By having all this information stored in only one model we help to ensure the consistency of the system and its documentation. As the items making up the system model change, this change can be reflected in all documentation since all are generated from the same model.

The production of the system development documentation goes through several stages:

1. The organization and content for the documentation are specified. The contents of the document are based on the JPL software documentation requirements, the development methodology being used, and the system documentation requirements found in the *PDS Software Management Plan.*

2. The information that will be needed to populate each section is determined. If some relationships among the data items are missing from the model, then the model and the database are augmented and the missing information is acquired.

3. The format for the presentation of the information (e.g., lists, tables, and diagrams) is determined and the TEX macros are written to format the information.

4. Programs are written (in C, TG code, or in the database system language) which will extract the information from the database and insert it properly into TEX macros. The programs are able to detect when pieces of information have not yet been entered into the database and handle this situation as the document is being generated.

5. The programs are run against the database and the documentation can be produced in several preliminary versions for inspection and review.

6. The final documentation is produced, reproduced and sent out for review and use.

The first two steps in this process are performed by development engineers; the third and fourth steps are done by document programmers; and the last two by the document production staff.

The graphics generated from Excelerator, Adobe Illustrator, and Cricket Draw must be included in the document. Initially these graphics items were produced in PostScript format, printed separately, and manually pasted into the documents. Currently these graphics are being stored in PostScript format and placed in the document directly via the `\special` command. Naming conventions are used to keep the graphic file names consistent with those found in the document.

This automated placement of graphics has shortened the time in which a document can be "turned around"—which was already remarkably short. For example, once all the edits to the database have been made, a new copy of any document can be produced in one half to one full day, and all the documents produced after the edits will be consistent with respect to contents.

2. User Manuals and Program Documentation

User manuals and program documentation are produced not only by the central node's system development staff but also by the science development staffs at the various nodes. Two nodes, in addition to the central node, have made significant and novel uses of TEX in producing this form of documentation.

The central node uses TAE (Transportable Applications Executive, a software package created by NASA's Goddard Space Flight Center) for its user interface. TAE allows the user to create menu definition files (MDFs) and procedure definition files (PDFs) and basically these define the user interface. All the associated help messages, parameter specification and passing, process initiation and control are handled by TAE. Loosely speaking, TAE builds a user interface system based on a set of tables. Through a rather clever bit of programming, the PDS staff builds all these tables using programs that draw the needed information from the system model. The menu hierarchy, components, help messages, descriptions and software procedure names are all contained in the system model. The User's Manual for this system is also produced from the same system model. This ensures consistency between the actual user interface system and the User's Manual.

Two of the science nodes have developed mechanisms for documentation of actual programs that involve TEX. These have some of the characteristics of the `WEB` system but have been designed for slightly different purposes. They have also been designed for languages other than Pascal, namely FORTRAN and C.

The node at UCLA has developed a system to produce online and printed

User's Manuals from the same source. They have also developed a mechanism to produce programmer's reference manuals from program headers. Help information for the programs is maintained online in VAX help file formats. No "foreign" TEX commands are allowed to appear in this material. However, natural looking but syntactically exact constructs are used which can be converted to material that includes the necessary TEX macros. For example, a list can be started by ending the previous line with a ":". Key caps are made by enclosing material between "<" and ">" inside a rectangle. For example <CR> is translated into CR. Tables presented in the online help as

```
+--------------------------------------+
| word         |  description          |
|--------------------------------------|
| help         | a request for assistance|
+--------------------------------------+
```

will be translated into a genuine table using the \halign command (multiple entries need to be set via vboxes/vtops). The number of dash symbols is used to determine how wide to make the tables. The use of these conventions allows the online material to appear natural and yet allows a correct translation into material that can be processed by TEX. The conversion of this material is handled by the awk and sed utilities.

The UCLA programming staff find it acceptable to maintain program header information inline with the program code. They have written awk and sed programs which extract these comments as well as actual code from the programs and then prepare material in a standard format for a reference manual. Information on functional declarations, entry requirements, returned values, development history, include files, functions accessed, common variables, and local variables can be captured and formatted into tables.

The node (located at JPL) in charge of navigation/planetary geometry for all satellite data has developed a different approach for producing consistent sets of program documentation. They produce all their program documentation using a special subset of TEX commands. These documents can either be run through TEX and typeset or they can be run through a special translator, SUBTEX (for SUBset of TEX), which will produce a pure ASCII version of the text. This ASCII version attempts to mimic the format that TEX would produce—indented lists, word wrap within tables, etc.—and can be used in help files or in program headers. SUBTEX allows the creation of tables, lists, and verbatim text so that the usual material needed in online helps can be produced. The node is developing a "reverse" translator to convert this ASCII material back to a form that has TEX macros in it. The utilities that do these translations are written in FORTRAN.

3. Status Reports

Monthly and quarterly progress reports are prepared for all the work on the PDS. The monthly reports are prepared by the PDS Administrative Assistant based on information received from the PDS managers. A set of macros has been written to produce these reports so that the managers have to use only a few simple TEX macros when they send in their text. The material is sent to the Administrative Assistant via electronic mail. Resource data (financial and manpower) is graphed using an Apple Macintosh and pasted in. Schedules are prepared using a special widely-used program at JPL and are pasted in (they are not produced in PostScript format).

Quarterly status reports are sent in (via electronic mail) by twelve different people to the Administrative Assistant. Each report forms one chapter of the *PDS Quarterly Report.* The macros for documents with multiple chapters discussed earlier are used for this document. A simple set of formatting conventions has been distributed to the preparers of chapters of the document and they try to follow these formatting conventions. The Administrative Assistant must sometimes make editorial changes to the documents and for this, the use of a keyboard macro package, such as ProKey, has proven invaluable. The combination of ProKey and WordPerfect commands allows the systematic and rapid transformation of non-TEX constructs into ones containing the appropriate syntax and TEX macros. Where additional formatting is required, as in the conversion of ASCII material into correctly formatted tables, the combined use of ProKey and WordPerfect can be used very effective and is sometimes much easier than using `awk` or `sed` utilities.

4. Presentation Material

Oral presentations of PDS material are usually accompanied by 8 1/2 × 11 inch overhead transparencies. The material for these overhead transparencies is prepared using a powerful and flexible set of transparency preparation macros created at JPL by Peter J. Scott. The PDS TEX expert has added some macros which basically restrict the options available to someone using these preparation macros. There are only six macros which anyone needs to use to prepare transparencies. Thus, this macro package is widely used and the preparation of presentation material is very easy, yet the results are very professional.

5. Other Documents

Other documents produced include telephone directories, action item status reports, and form letters. These documents have been produced from dBase III files, and special macro packages have been written to format these documents. This allows very professional reports to be prepared as often as necessary and with minimal impact on PDS personnel resources.

Conclusions

TeX has been used extensively on the PDS since 1985 and by a wide group of people with varying degrees of expertise in document preparation systems. Some conclusions can be drawn from this experience based on well-understood experiments and processes. These are:

1. People are more receptive to reading documents prepared using a professional typesetting system such as TeX than if a standard word processing system had been used. The content of the document is very important of course but we have shown that the format and appearance of the document are also important.

2. The engineering of a document production process is worthwhile but definitely non-trivial. By using databases and TeX macros, information can be formatted in ways that enhance its interpretation and usefulness. Complex information can be integrated correctly. The consistency and correctness of document contents can be enhanced by producing documents from one source. In order to achieve these benefits, however, careful planning must be done. Issues such as how to build and control the database, how to ensure consistency of written material, how to map between a database and a document, and how to construct programs to build documents must be addressed during the early stages of the engineering effort. The building of tools to help in the documentation process can be expensive initially but can pay big dividends when these tools are used repeatedly.

3. Care must be taken to consider the person who will be reading a document produced from a database. The material extracted from a database can begin to look very formal and very repetitious if one is not careful. Proper introductions and explanations of the extracted material are necessary to enhance readability. An enormous amount of information can be extracted from the database and the document can become quite large. Decisions must be made carefully regarding how much material to include in such a document.

4. Documents with a consistent format can be produced at different sites by the use of a common set of TeX macros. Differences in document "source codes" (e.g., spacing, indentations, line lengths) disappear when the final product is produced. Electronic exchange of document "source code" is very feasible since TeX uses only standard ASCII characters.

5. People must receive training in the use of TeX. Most people will not learn as complicated a language as TeX without some training. Different groups of people need training for different levels of TeX expertise. Writing and TeX formatting conventions must be well documented and examples of their use prepared. Commonly used macros must be written to simplify the implementation of standard document formatting conventions.

6. Total document production costs using TeX and databases can be lower than

if an outside documentation preparation service is used. Different versions of a document can be produced rapidly and with little additional cost.

Bibliography

Childs, D. B. *Planetary Data System System Engineering Environment.* PDS internal document. 1988.

DeMarco, T. *Structured Analysis and System Specification.* Englewood Cliffs, New Jersey. Prentice-Hall. 1979.

Page-Jones, M. *The Practical Guide to Structured Systems Design.* New York, New York. Yourdon Press. 1980.

Acknowlegement

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

TEX and Databases

DAVID NESS AND JAMES SLAGLE

TV Guide Magazine
Radnor, PA 19088

ABSTRACT

Sometimes several documents form a *family*, or group, because they share some common characteristics. It is often useful to be able to extract these characteristics from the documents in order to put them into a database that can be used for other purposes, including automated generation of summary documents. This article describes some facilities that allow us to do this with families of TEX documents.

What TEX Facilitates

TEX is flexible to the extreme. This makes it possible—through the careful organization of information—to accomplish more than the rendering of the documents themselves. A common example of this kind of *meta*-use is to prepare lists and summaries of documents of various kinds.

Some Common Problems

We have found the technology described in this article to be of use to us in several different problem domains:

- project control documents
- program documentation
- general memos
- computer purchase 'dockets'

All of these areas share an important common characteristic:

> There is a requirement for at least *two* ways of looking at the documents: (1) as individual documents themselves; and (2) as information to be incorporated into a summary of the documents relevant to a particular purpose.

Dockets as a Prototype

The Docket in our organization is a document that is presented to the Steering Committee requesting an allocation of funds for the purchase of computer

hardware or software (or occasionally, services). All dockets share a number of characteristics, which include:

- Title
- Requester
- Author (often a different person from the requester)
- Amount
- Date Submitted
- Date Decided Upon
- Purpose
- Nature, etc.

Some of these data items are a few words or a sentence (i.e., Title, Author, Dates, Amount). Others can be many paragraphs in length (i.e., Economic Justification, Purpose).

Dealing with these documents led us through some problems to a comfortable solution. In fact, these rather sophisticated documents are now routinely produced by an employee with only introductory-level TEX skills.

An Early Try

In the early days of preparing Dockets and Docket Summaries, we attempted to maintain this information in two separate-but-related documents: a text file and a facts file. The text file grew to be a very long document containing all the text describing the verbose sections of dockets, such as Nature and Purpose. The facts file, on the other hand, was stored in a comma-delimited file format that we borrowed from BASIC.

With this early try, all the information necessary to produce a Docket Summary was contained in the facts file. However, to produce an individual docket, it was necessary to pull data from both the text and facts files.

The database became difficult to maintain. Changes to a docket or dockets required that two different documents be updated concurrently, and sometimes we ended up with inconsistent entries.

The Solution

The solution—which has also worked well for other databases—is to maintain each docket as a self-contained document, with facts stored as `\def` values. The documents have been given similar filenames, e.g., `dock0100.tex`, `dock0101.tex`, to facilitate batch processing.

A batch process extracts these definitions and compiles them into a summary file that is similar to the facts file formerly maintained by hand. This summary file is always built on demand in order to produce the Docket Summary, thus ensuring that the summary data is consistent with the actual dockets.

The process requires some general purpose sorting software, and some software specifically developed to handle TEX definitions in useful ways. We are comfortable with this mix of technology, although it may offend some TEX purists.

`GETDEFS` passes through the list of dockets and gathers macro definitions into an output file. Every time that the sequence `\def\macname{Definition}` is encountered, a line is written into the output file consisting of:

```
FILENAME: macname="Definition"
```

`REMAPDEFS` is then executed to replace the names of the macros in the file produced by `GETDEFS` by sequence numbers that are specified in an auxiliary remap file. This fixes some field lengths and makes the file amenable to sorting and further processing. Definitions not required in the output database are removed at this point by excluding them from the remap file.

A `SORT` program then organizes the definitions into the desired order, typically descending order, so that the most recent dockets will head the docket summary.

Finally, `COMBINE` is run to group all the definitions for an individual docket, as in:

```
DOCKET1{Arg1, Arg2, Arg3, ...}
```

A Sample Docket

A fragment of a docket follows. The definitions are entered one per line in a format we consider the header. In practice a new header is virtually always picked up and edited from an existing docket, minimizing omissions and making it easy to compare one document to another. Some of the `\defs` that are not of any use in a summary file will only print in 'Docket 0185'; others will print only in the 'Docket Summary', and many will be used in both.

```
 \def\dockno{0185}
 \def\requestby{J.Slagle}
 \def\dept{REO}
 \def\amt{151,575}
 \def\state{Proposed}
 \def\name{Telex}
 \def\type{Hardware}
 \def\writer{JFS}
 \def\firstdisc{25 May 88}
 \def\lastdisc{}
 \header
 \nature{
This is a request for equipment to upgrade three classes of PCs in
the field . . .
        }
\purpose{
```

```
This is a follow-up to Docket 0170, in which the need for faster
processing . . .
        }
```

Producing a Docket Summary

Anyone fluent in TEX should be able to produce 'Docket 0185' from the above data. The `\defs` can be called upon as needed, and the form determined by a dockets style file.

Producing a summary of all the dockets is less straightforward. The first step is to collect the `\defs` from each of the dockets into a facts file. As outlined above, this is accomplished by a batch process of several home-grown programs. The process passes through each document collecting the 'facts' and sorting them into an order defined in an auxiliary file, then reformatting the results into a useful file format:

```
*DOCKNO,FIRSTDISC,     REQBY, START,     FDATE,        AMT,     STATE,    NAME,
   0185,  25May88, J.Slagle,    ---, 88-06-07, "151,575", Proposed,   Telex,
   0184,  15Apr88,   D.Ness,    ---, 88-06-07,          ?,       ---, Mitron,

   [...  ...]
```

Compare this fragment to the document for docket 0185 above. The first row labels the columns that follow. Facts such as `DOCKNO` and `FIRSTDISC` were pulled unaltered from the original dockets. `FDATE` (file date) was obtained from `DOS`. `START` was not only renamed from `\lastdisc` in the original, but a default value of '---' was supplied, since the original `\def` was found empty. The value for `AMT` was quoted for docket 0185, since it contained a comma—our delimiter character.

Once the facts file is prepared, certain facts are selected and dumped into a file that will be presented to TEX:

```
\def\mac{\xxx{A}
\fact<0185=J.Slagle=25May88=151,575=Proposed=Telex>
\fact<0184=D.Ness=15Apr88=?=---=Mitron>
\tablerule}
\input DOCSUM.FIL
```

`DOCSUM.FIL` is essentially a style file, containing basic table-building TEX. Here is a look at the output:

Steering Committee Docket Status					
No.	Req.By	Date	Amt	State	Item
0185	J.Slagle	25May88	151,575	Proposed	Telex
0184	D.Ness	15Apr88	?	—	Mitron

Knuth, Speed & Flexibility: Some Appreciation

Working on a project like this makes one appreciative of both the flexibility and efficiency with which Knuth implemented TeX. By worrying about efficiency inside TeX, Knuth left us free to make heavy use of facilities—such as macro definitions and calls—which otherwise might prove expensive and cumbersome. Without hesitation we can encapsulate crucial notions in macros and know that we will never be concerned about the computer time that will be eaten up by the implementation.

Moreover, the flexibility of TeX makes it easy to use macro definitions to store information. If we wanted to define some macros, but not have them recognized by our database management software, we would just provide an alternative form for `\def` and its allies. Our database software would never see the definitions while TeX, of course, would.

Definitions and Calls

So far we have accomplished our objectives by using `\def` and `\gdef` to mark 'facts' for later use. We have not found a need to keep track of our `\xdef` and `\edef` commands, as these would be unusual in the context we are describing here, although the process could be easily adapted to accommodate these related commands. At the moment we normally recognize only `\defs`, but a run-time switch on our software allows `\gdefs` to be recognized also.

Relationship to SGML

At present we are disinclined to pursue Standard Generalized Markup Language, since we have been able to achieve what it promises and more through TeX. With some clever re-definitions of the active character, it may even be possible to cause TeX to rather directly interpret SGML. As has been demonstrated above, TeX is able to accomplish some of the sought-after separation of form and content, and it is able to handle database functions as well, something not generally touted for this kind of language.

Managing the Database

The database—a collection of documents—is stored on the server of a local area network. The utility program `ARC` is used to compact them into a single archive, which is rigorously backed up. Any time a facts file is needed, it is built on demand to incorporate the latest revisions. Because several people work on the dockets, we must be careful to avoid simultaneous updates.

The facts file uses a very simple database format, in which each line represents a record and fields are comma-delimited. This format is very similar to that used by an off-the-shelf database manager called InfoScope that performs standard data-management tasks (sorts, totals, averages, etc.). In fact, to load

our database into InfoScope, we simply split off the format line into a separate file, since InfoScope reads data and format from two separate files.

Producing Manual Sets Using Single-Sourcing

LAURIE MANN

Stratus Computer, Inc.
M22PUB
55 Fairbanks Blvd.
Marlboro, MA 01752
508-460-2610
uucp: harvard!anvil!es!lmann

ABSTRACT

Maintaining sources for manual sets can be made easier by exploiting TEX's powerful conditional capabilities. This paper describes how Stratus has developed a system for producing multiple manuals from the same basic source files, and other ways this system can be used.

Introduction

In documentation departments, it is common to write and revise manuals that are similar. At Stratus Computer, we publish some manual families where related manuals contain much of the same text. For example, we publish six manuals that document operating system subroutines—one for each of the six programming languages Stratus supports. Text describing individual subroutines is the same from language to language; usually the only difference from language to language is the subroutine declaration. Rather than writing separate descriptions for each subroutine in each language, we write and maintain a single source file containing the subroutine description. This paper describes *single-sourcing*, a system which simplifies the writing and maintenance of manual families.

The History of Single-Sourcing

Stratus Computer was founded in 1980, and designs and builds fault-tolerant computers for on-line transaction processing. In the early 1980s, computer manuals at Stratus were written in-house, but produced out-of-house. TEX was first used at Stratus to produce manuals in 1983. By early 1984, we had converted old manual source files to TEX, and coded all new material in TEX.

When the *VOS Service Subroutine Manual* (*R005*) was first published in 1981, it was 370 pages long and documented how to call 152 subroutines in PL/I. During 1982 and 1983, Stratus added a number of new subroutines to the operating system. In addition, we needed to document subroutine support for

BASIC, FORTRAN, Pascal, and COBOL. Because of the number of subroutines being added, we were faced with three problems:

1. How could we best maintain the source files for nearly 200 subroutines in five different languages?
2. How could we manage documentation sources to allow for future expansion (i.e., new language support and new subroutines)?
3. How could we make the *VOS Service Subroutine Manual* (*R005*) easy for the reader to use, in light of the amount of material we needed to present?

We decided to document the subroutine set for each language in a separate manual. The five manuals would share the same text with the exception of the subroutine declarations. Since the subroutine declaration for each language was different, it was coded in a separate input file. Here is part of the source file for the `s$read` subroutine. Flags appear in the left margin of the source code, and the related explanation follows the code:

```
\begincrs{s$read}
\INDEX{s$read subroutine}

\begintitle{Purpose}
 The subroutine |s$read| reads a record in the file or I/O
 device attached to the caller's |default_input| port.

\begintitle{Usage}
```

A `\input read.\language.tex`

A `\language` is a macro which appears as an infix in the name of a file containing a subroutine declaration. For example, the name of the file containing the usage information for `s$read` in COBOL is `s$read.cob.tex`. For Pascal, the name of the input file is `s$read.pas.tex`.

The `s$read` PL/I usage input file, `s$read.pl1.tex`, is coded as follows:

```
\setbox\codebox=\vbox{\begintt
declare buffer_length binary(15);
declare record_buffer char(@N@) varying;

declare s$read entry( binary(15),
                      char(@N@) varying);

          call s$read( buffer_length,
                       record_buffer);
\endtt}\code
```

The s$read BASIC usage input file, s$read.bas.tex, is coded as follows:

```
\setbox\codebox=\vbox{\begintt
dimension           buffer_length%=15
dimension           record_buffer$<=@N@

subprogram s$read( buffer_length%=15, &
                    record_buffer$<=@N@) external

       call s$read( buffer_length%,    &
                    record_buffer$)
\endtt}\code
```

To pull together all the source files and macros for one manual, we wrote a unique wrapper file for each version of the manual. The following is a portion of the wrapper file, used for the second edition of the *VOS PL/I Subroutines Manual* (*R005*):

```
A  \def\language{pl1}
   \def\languagename{PL/I}
   .
   .
   \beginmanual{\vosplisubrsmanual}

B  \input subintro.\language.tex
   \startpart{Introduction to the \vos\ Service Subroutines}

   \input subguide.\language.tex
   \input briefsubsdesc.\language.tex

C  \input abbrev
   \input addepiloguehandler
   \input additem
   .
   .
   \input writewrapindent
   \input writewrappartial
   .
   .
   \endmanual
   \end
```

A This is the definition for the `\language` infix.

B The files in the wrapper with the `\language` infix are unique to each manual; these files are different for each language documented.

C Each subroutine description is documented in a separate file. As described above, each source file contains another input file, to call in the appropriate usage file. Since each manual has a unique wrapper file, it is easy to modify the list of subroutines to be included for each manual. A number of VOS subroutines supported under PL/I, are not supported under BASIC or FORTRAN. For example, the epilogue handler subroutines are not documented in the *VOS BASIC Subroutines Manual* (*R018*), but they are documented in the *VOS PL/I Subroutines Manual* (*R005*).

To document the `s$read` subroutine in every supported language, the following files are needed:

`read.tex`	TEX source file for the text of the `s$read` subroutine
`read.bas.tex`	BASIC usage for the `s$read` subroutine
`read.cob.tex`	COBOL usage for the `s$read` subroutine
`read.for.tex`	FORTRAN usage for the `s$read` subroutine
`read.pas.tex`	Pascal usage for the `s$read` subroutine
`read.pl1.tex`	PL/I usage for the `s$read` subroutine.

When the wrapper file for *VOS PL/I Subroutines Manual* (*R005*) is compiled, TEX reads in the appropriate files. When that edition of the *VOS PL/I Subroutines Manual* (*R005*) was published, the printed description of the `s$read` subroutine looked as follows:

s$read

Purpose

The subroutine `s$read` reads a record in the file or I/O device attached to the caller's `default_input` port.

Usage

```
declare buffer_length  binary(15);
declare record_buffer  char(N) varying;

       declare s$read  entry( binary(15),
                              char(N) varying);

         call s$read(  buffer_length,
                       record_buffer);
```

In contrast, the description of the **s$read** subroutine in the *VOS BASIC Subroutines Manual* (*R018*) looked like this:

s$read

Purpose

The subroutine **s$read** reads a record in the file or I/O device attached to the caller's **default_input** port.

Usage

```
dimension                buffer_length%=15
dimension                record_buffer$<=N

   subprogram s$read(    buffer_length%=15, &
                         record_buffer$<=N) external

         call s$read(    buffer_length%,&
                         record_buffer$)
```

Other language-specific information is also put in special files. This information generally consists of tables, which sometimes vary from language to language.

While this system works well for most of the subroutines, there is a limitation. A few subroutines function differently from language to language. We had to write different subroutine descriptions for each language. As a result, we had to maintain different descriptions for some subroutines.

1. Debugging the `\language` Macro

The `\language` files required little debugging, and worked as we expected. We ran into one problem. Occasionally, some information is needed for subroutine descriptions in some languages, but not in others. It was clear that the `\input` statement had to have a corresponding file in each language we were documenting, so we sometimes needed to create empty files. However, we soon discovered that TEX would stop in its tracks when it found an empty input file. To prevent this, we had to create a few files with one line:

```
\vskip\leadreg
```

This method allowed TEX to continue processing the manual.

The primary limitation of the input files was the inability to insert text of less than one paragraph in a subroutine description.

The \languagecase Macro

Over the next year, the subroutine manuals were revised. New subroutines, and a new language (C), were added. The manuals continued to grow, and averaged 800 pages for the spring 1985 release.

For the following release, we took an additional step to make the project a little more manageable. We decided to develop a way to permit brief insertions of language-specific information into paragraphs. To help handle this problem, we wrote a new macro, `\languagecase`. `\languagecase` is a powerful conditional macro that inserts brief amounts of language-specific text into manuals without having to use input files. The `\languagecase` macro takes six arguments:

```
\long\def\languagecase#1#2#3#4#5#6{%
          \iflanguageall [BASIC: #1] [C: #2] [COBOL: #3]%
                         [FORTRAN: #4] [Pascal: #5] [PL/I: #6]\fi%
          \iflanguagebasic#1\relax\fi%
          \iflanguagec#2\relax\fi%
          \iflanguagecobol#3\relax\fi%
          \iflanguagefortran#4\relax\fi%
          \iflanguagepascal#5\relax\fi%
          \iflanguagepli#6\relax\fi}
```

The `\languagecase` macro uses a series of "language switches." These switches "turn on" or "turn off" the appearance of language-specific text in the output. Their definitions follow:

Definition	**Description**
`\newswitch{languageall}`[1]	Show all languages in text
`\languageallfalse`	Disregard the `\languagecase` macro
`\newswitch{languagebasic}`	Show BASIC in text
`\languagebasicfalse`	Disregard BASIC in text
`\newswitch{languagec}`	Show C in text
`\languagecfalse`	Disregard C in text
`\newswitch{languagecobol}`	Show COBOL in text
`\languagecobolfalse`	Disregard COBOL in text
`\newswitch{languagefortran}`	Show FORTRAN in text
`\languagefortranfalse`	Disregard FORTRAN in text
`\newswitch{languagepascal}`	Show Pascal in text
`\languagepascalfalse`	Disregard Pascal in text
`\newswitch{languagepli}`	Show PL/I in text
`\languageplifalse`	Disregard PL/I in text

1

[1] The plain files we have developed at Stratus diverge substantially from the plain file described in the *TeXbook*. `\newswitch` is our version of the `\newif` control sequence.

The `\languagecase` macro is used in text when a writer describes a subroutine that has different values in each language. For example, subroutines take different numbers and types of scalar arguments, depending on the language. Here is how `\languagecase` is used in text to differentiate language-specific information:

```
The operating system subroutines take
\languagecase{three}{five}{five}{three}{five}{five}
general types of scalar arguments:  arithmetic,
character string,\languagecase{}{ pointer, entry,}{ pointer,
entry,}{}{ pointer, entry,}{ pointer, entry,} and logical.
```

The wrapper for the *VOS FORTRAN Subroutines Manual* (*R020*) contains the `\languagefortrantrue` switch. This switch tells TeX to pick and print the material in the fourth set of braces after the `\languagecase` macro. In the *VOS FORTRAN Subroutines Manual* (*R020*), the paragraph shown above reads:

> The operating system subroutines take three general types of scalar arguments: arithmetic, character string, and logical.

In the *VOS PL/I Subroutines Manual* (*R005*), the paragraph reads:

> The operating system subroutines take five general types of scalar arguments: arithmetic, character string, pointer, entry, and logical.

One particular strength of the `\languagecase` macro is the ability to show **multiple** languages for the purposes of review. In this way, we are able to show reviewers each language on the same page. This makes technical review easier:

> The operating system subroutines take [BASIC: three] [C: five] [COBOL: five] [FORTRAN: three] [Pascal: five] [PL/I: five] general types of scalar arguments: arithmetic, character string,[BASIC:] [C: pointer, entry,] [COBOL: pointer, entry,] [FORTRAN:] [Pascal: pointer, entry,] [PL/I: pointer, entry,] and logical.

The addition of the `\languagecase` macro means that we can place short amounts of language-specific information in single-source files. Material can be inserted in the middle of paragraphs without having to resort to input files or separate description source files. We do not have to create new input files for brief differences from language to language.

2. Debugging the `\languagecase` Macro

The `\languagecase` macro is used extensively in text. However, `\languagecase` does not work in all cases. We are running TeX Version .99,[2] so we have problems

[2] This is, fortunately, no longer the case! We're now up to 2.0, and have ported TeX from Pascal to PL/I.

with macro memory space. When `\languagecase` is used to include more than two or three paragraphs, we sometimes see the following log message:

```
TeX capacity exceeded, sorry
If you really absolutely need more capacity,
you can ask a wizard to enlarge me.
```

In order for TeX to process this type of insertion, we put language-specific text of more than two paragraphs into separate input files.

The `\languagecase` macro must be used with caution within certain macros. For example, `\languagecase` can create problems within macros that write to output files. One such macro (`\leveltwohead`) writes material out to the table of contents file. A `\leveltwohead` was coded like this:

```
\leveltwohead{Using the SQL/2000 SQL Server with the
\languagecase{}{}{C}{COBOL}{}{}{PL/I} DB-LIBRARY}
```

As a result, the table of contents line in the *SQL/2000 C DB-LIBRARY Reference Manual (R150)* looked like this:

Using the SQL/2000 SQL Server with the CDB-LIBRARY

To recapture the lost space, we need to insert a final \ after the end of the `\languagecase` macro:

```
\leveltwohead{Using the SQL/2000 SQL Server with the
\languagecase{}{}{C}{COBOL}{}{}{PL/I}\ DB-LIBRARY}
```

Here is how the table of contents line should look:

Using the SQL/2000 SQL Server with the C DB-LIBRARY

We found it best to insert entire index hits inside the `\languagecase` macros, rather than put the `\languagecase` macro inside the `\index` macro:

```
\languagecase{\index{basic limitations}}{\index{c limitations}}
{\index{cobol limitations}}{\index{fortran limitations}}
{\index{pascal limitations}}{\index{pl1 limitations}}
```

Finally, we found we could not put `\input` statements inside `\languagecase` macros.

Other Possible Applications for Single-Sourcing

Over the last few years, we have adopted this system of single-sourcing for a variety of related manuals. Our set of Transaction Processing Facility Manuals were all written and produced using generic text files with language-specific inputs. Additionally, a number of manuals on programming for database systems have also been written in this manner. However, this system can be adapted for a variety of other purposes.

1. Documentation for internal versus external users. Documentation for in-house users can contain information that might not be appropriate for customers to have, but would be beneficial for the in-house users to know.
2. Different levels for different users. Some manual sets are very similar structurally. The main difference is that examples are aimed at different types of users, and that additional explanation is added for beginning users.
3. Teachers' editions of textbooks. Teachers' editions invariably contain more examples, answers to problems, ideas for class discussion, etc., which don't appear in students' editions,
4. Maintaining the same documentation for different versions of the same software. Some companies maintain many releases of software, and the software runs a little differently from release to release and from machine to machine. By using single-sourcing, changes can be inserted in input files or in conditional macros without having to write multiple revisions.

In conclusion, TEX's conditional capabilities can make maintaining and revising manual sets easier. By using a variety of input files and conditional macros, it is possible to generate a family of manuals from the same basic sources, and to maintain less source.

Using TEX to Produce Government Standard Documentation

JEAN J. POLLARI

Rockwell International
M.S. 153–100
400 Collins Road, NE
Cedar Rapids, Iowa 52498

ABSTRACT

A military program requires substantial documentation, and these documents must be formatted according to strict government standards. Rockwell-Collins in Cedar Rapids, Iowa, has developed a set of TEX macro packages to support these documentation standards. The documents are typically very large (1000+ pages) and involve complex mathematics, tables, and figures. The hardware environment consists of a VAX 785, a VAX 8800, a MicroVAX II, QMS Lasergrafix 800 and 2400 laser printers, and a Digital LN03 laser printer (with upgrades).

Introduction

There are five primary types of documentation that must be produced for any government contractor software project: Requirements Specification, Design Specification, Test Plan, Test Procedure, and Test Report. The Requirements Specification documents the requirements that software must satisfy in order to fulfill the government contract. The Design Specification documents software design; it describes, in detail, every module of the software, how these modules interact with each other, and how the software is controlled. The Test Plan describes the method of verifying that the software satisfies the requirements described in the Requirements Specification. Detailed instructions to execute the test are provided in the Test Procedure, and the final results of the testing are reported in the Test Report. The format and content of these documents are very specifically defined in Military or Department of Defense Standard publications (i.e, MIL-STD-490A, DoD-STD-2167A).

The Problem

For our project, we had to create unique documentation for every software deliverable that we produced, even though much of the text was common among documents. Some of these documents were very large (over a thousand pages each), and many contained complex mathematics.

We were using a Wang word processing system to create our documents, AutoCAD to generate graphics, and a lot of engineering time. We wanted a new system that would automate and simplify the creation of these documents; our first step was to identify the problems that we had with our old system and compile a wish list of things that we wanted our new system to do.

1. Some of our documents contained over a hundred pages of complex math equations, including matrices, integrals, etc. Our existing system would only support a limited subset of mathematical expressions. Our new system had to allow us to typeset these equations.

2. Since we have many different engineers writing sections, we wanted a system that would accept source files from different computers (i.e., VAX or IBM PC). Our existing word processing system did not have this flexibility; all source files had to be created and maintained on one system by the word processing staff.

3. The new system had to support all Military and Department of Defense Standard format requirements.

4. We wanted to use this system for many different types of documentation, so we wanted a system that would let us decide upon the format of the document, rather than forcing us to use predefined layouts.

5. The new system had to automatically generate a table of contents, list of figures and list of tables with page numbers.

6. The new system had to automatically number sections, figures, and tables, and allow the author to use symbolic references to the section, figure, or table number which would be replaced with the correct number when the document was generated.

7. The new system had to allow graphics to be merged with the document text. Our old system would not allow this; figures were created using AutoCAD, and then cut and pasted onto blank pages reserved in the document.

8. The new system had to automatically number and indent lists according to Military and Department of Defense Standard requirements.

9. Several portions of our documents share common text. Our existing word processing system required that we maintain separate documents, with the common text stored in different places. Our new system had to allow common files to be shared among documents.

10. The Requirements Specification contains a Test Verification Matrix (TVM) which lists each requirement that must be formally tested. The number of the section which contains the requirement, a description of the requirement, and the method of testing the requirement are all listed on a multi-page table. The TVM must be kept consistent with the requirements. If a section number changes, it must change in the TVM; if a requirement changes, the

description in the TVM must change; if a requirement is deleted, it must be deleted from the TVM. We wanted our new system to generate the TVM automatically from the requirements, ensuring that the section numbers and requirements to be tested were correct.

11. The Test Plan also contains a Test Verification Matrix. The Test Plan TVM is identical to the Requirements Specification TVM, except that it has two additional columns which identify where each requirement is tested. We wanted a way to easily tailor the Requirements Specification TVM so that it could be used in the Test Plan.

12. Our documents often go through several revision cycles. We wanted our new system to support revisions to a baseline document, including change bars in the margins and modified page numbers.

13. We needed a document processing system that would accept input from outside sources. We were developing Requirements and Design Specification support systems which would help us to track data and control flow information, and we had to have a document processor which could use files generated by these systems.

14. Government projects sometimes require that the contractor develop software using classified information. This classified information must be included in the documentation, and, as a result, the documentation itself becomes classified. Classified documents must contain special markings. Every paragraph must be marked with the classification level: (S) for secret, (C) for confidential, (U) for unclassified. The top and bottom of the page must be marked with the highest classification level on the page. For example, if a page contained unclassified, confidential, and secret information, the top and bottom of the page must be marked "SECRET."

    Using our old system, we had been typing in the (S), (C), or (U) for every paragraph, and then using a hand stamp to mark the page classification. We wanted our new system to automatically determine the page classification markings from the individual paragraph markings.

The Solution

Once we knew what we wanted, we needed to find a product (or products) that would satisfy our requirements. Our support environment initially consisted of a VAX 785, a MicroVAX II, a QMS Lasergrafix 800 and a Digital LN03 printer without upgrades. (During our second year with TEX, we upgraded to a VAX 8800, a QMS Lasergrafix 2400, and added the LN03 memory cartridge to support TEX.) Our search concentrated on products that would work in our existing environment. With the wish list in hand, we began searching for a vendor product.

Most of the document processing systems today will support the basic constructs such as tables of contents, lists of figures and tables, etc. The choice of products narrows, however, when you add graphics inclusion, and narrows even further when you add mathematics support. We had some initial bias towards a WYSIWYG system, since it seems easier to learn. The disadvantages of WYSIWYG systems, however, became apparent when we began to consider implementing such a system for our large documents. The cost of a WYSIWYG system for 75 engineers was prohibitive, since each engineer would require a graphics terminal. We did not want the engineer to be able to decide upon the format of his/her section. We wanted one point of control for the format. WYSIWYG systems are also very slow, and very few could support documents of over a thousand pages in length.

Once we narrowed the field to batch document processing systems, TEX became a logical (and inexpensive) choice that would satisfy all of our needs. We liked the device independent feature, and the fact that we could enter TEX source using almost any text editor. Since we already had five or six editors, and each engineer had his/her favorite, we did not want or need any more editors. We really did not care about the incredible precision or beauty of TEX's typesetting capabilities. We chose TEX primarily for its mathematics support and for its powerful macro capability.

We spent the first month learning the basics of TEX, and attempting to get the format that we wanted. Unfortunately, our only resource was the *TEXbook* (Knuth 1984). The *TEXbook* is an excellent reference and tutorial handbook, but sometimes a human source of information would have saved a lot of time.

In our first attempt we created a file containing all macros necessary to format the Requirements Specification. As we began working on the macro packages for other documents, we realized that there was a great deal of commonality among the files. At this point we began to partition the macros into stand-alone packages which could be used where necessary. The following sections describe these packages in further detail.

1. Macros for the Document Format

The document format is defined in a file called `LAYOUT_MACROS`. These macros define a four-line header and a two-line footer, with the header and footer on opposite sides for odd and even numbered pages. All text is formatted with ragged right, and `tolerance` and `badness` are both set to 5000, since line breaks with ragged right are not very critical. Magnification is set at 1200 to make the text more readable.

The author can now enter text, using any text editor, in any format, as long as the proper TEX macros are used. TEX will typeset the text according to the format that we have defined. If we decide to change the format, we just redefine the macros. The engineer no longer has to decide what the document should look like. He/she only needs to be concerned with the content.

2. Macros for Section, Figure, and Table Numbering

The section, figure, and table macro definitions are grouped into a file called `LEVEL_MACROS`. The `\levelone{title}{label-name}`, `\leveltwo{...}{...}`, etc., macros automatically number sections. The section number and title are inserted in place of the `\level...` macro, and the section number, title and page are written to a table of contents file which can be `\input` at the end of the document. The optional `{label-name}` parameter allows the author to assign a symbolic name to the section number, and then reference the section number using the `\ref{label-name}` macro. When the file is TEXed, the correct section number is inserted for the label name.

The `\table{title}{filename}` and `\figure{title}{filename}` macros work like the `\level...{...}{...}` macros. When TEX encounters one of these macros, it will fill up the rest of the current page with text, and then place the table or figure on the following page with the appropriate number and title heading. The table or figure number, title and page number are written to a list of tables or list of figures file which can be `\input` later in the document. The second parameter `{filename}` is mandatory, and contains the file name of the figure or table. We create boxed and ruled tables using some special macros that are discussed later in this paper. We continue to use AutoCAD for figures, and use TEX's `\special{}` macro to command our QMS driver program to copy the graphics file into the final printer-ready file. The author can reference the table or figure number in the text by using the `\ref{filename}` macro.

Both the `\table` and `\figure` macros use `\pageinsert` to control placement of the input file. Variations of these macros use `\midinsert` to place a small figure or table at the current location in the text, if it will fit. The insertion macros have been modified to prevent figures and tables from getting out of sequence.

3. Macros for Formatting Tables

We created **TABLE_MACROS** to make it easier to build tables using TeX. There are three types of tables allowed: boxed centered tables, centered tables, and regular tables. The author specifies this using the `\begintablebox`, `\begintablecenter`, or `\begintable` macros, and ending the table with a corresponding `\endtablebox`, `\endtablecenter`, or `\endtable` macro. For example, to get this:

Year	World Population
8000 B.C.	5,000,000
50 A.D.	200,000,000
1650 A.D.	500,000,000
1850 A.D.	1,200,000,000
1945 A.D.	2,300,000,000
1980 A.D.	4,400,000,000
This table reprinted without permission from the *TeXbook*, page 246.	

the author types this:

```
\begintablebox
\columnright&\columnright&\cr
\changecenter{Year}&World Population&\cr
\tableline
8000 B.C.&    5,000,000&\cr
  50 A.D.&  200,000,000&\cr
1650 A.D.&  500,000,000&\cr
1850 A.D.&1,200,000,000&\cr
1945 A.D.&2,300,000,000&\cr
1980 A.D.&4,400,000,000&\cr
\tableline
\boxspan{2}{4.7cm}{This table reprinted without permission
from the {\it \TeX book}, page 246.}&\cr
\endtablebox
```

The first line after the `\begintable...` macro is the template line. There are four possible templates: `\columnleft`, `\columnright`, `\columncenter`, and `\columnbox{dimension}`. The `\columnbox{dimension}` specifies that all entries in that column are to be placed in a box of width `{dimension}`.

The TeXist can over-ride the template for a particular entry in a column by using the `\changeleft{text}`, `\changeright{text}`, `\changecenter{text}`, and `\changebox{dimension}{text}` macros. The `\leftspan{#}{text}`, `\rightspan{#}{text}`, `\centerspan{#}{text}`, and `\boxspan{#}{dimension}{text}` macros allow the user to easily span text across # number of columns.

4. Macros for Lists

The LIST_MACROS enable us to automatically number and indent lists. For example, to create a list the author types:

```
\list
\item This is the first item.
\subitem This is a subitem.
\subitem This is another subitem. I can type
this in any way I want and \TeX\ will reformat it.
\item This is the second item.
\endlist
```

TEX will reformat this to look like:

1. This is the first item.
   a. This is a subitem.
   b. This is another subitem. I can type this in any way I want and TEX will reformat it.
2. This is the second item.

With these macros, the author no longer has to worry about renumbering lists when an item is eliminated or added, and the numbering scheme for lists no longer depends upon the author's preference.

5. Macros for Commonality

Our documents are created in a skeleton format. The highest level skeleton file contains the TEX formatting commands for a particular document and the command to include the text files. The actual text for each section is partitioned into separate files. If necessary, the text files may also be treated as skeleton files; they may only contain commands to include lower level text files. This ability to combine small text files into one document solved the problem of commonality for us. An engineer could write one common section, and the section could then be `\input` into as many different places as necessary.

We have also defined macros to make almost common files truly common among the documents. We have defined `\ifdocument` macros to test for different documents. The skeleton file identifies the document type by setting a switch, such as `\srstrue`. The switch `\srstrue` identifies this as a Software Requirements Specification. Later in the document, the author could type:

```
\ifsrs Some text that should only appear in the
       Software Requirement Specification.
\fi
```

If the `\srstrue` switch is set, the text between the `\ifsrs...\fi` will be included in the document; if the `\srstrue` macro is not included in the skeleton file, the statement will not appear.

6. Macros for the Test Verification Matrix

We now generate the Test Verification Matrix in the Requirements Specification by embedding `\vmatrix{description}{code}` macros in the requirements section. The `{description}` parameter indicates the requirement description; the `{code}` parameter indicates the method of testing the requirement. The `\vmatrix` macro expands to write a `\vmxentry{section number}{description}{code}` macro to a TVM file, using the current section number in the document. Later in the document, this file is read and the `\vmxentry` macro formats the entries to create the multi-page TVM.

Since the TVM description immediately follows the actual requirement in the text, the problems of missing requirements, incorrect paragraph numbers, or changes to the requirements that do not show up in the TVM are eliminated. As the engineer is updating the requirements, he/she can update the TVM entry at the same time.

The Requirement Specification TVM file is then copied to be used in the Test Plan document. The `\vmxentry` macro is redefined to accept two additional parameters which indicate the specific test where the requirement will be tested. Since the same file is used to generate both the Requirements Specification TVM and the Test Plan TVM, they are guaranteed to be consistent.

7. Macros for Revisions

Unfortunately, even with TEX our documents are not perfect the first time they are submitted for review by the customer. They usually undergo at least one revision cycle, and changes must be made to the text. These changes must be identified with a change bar in the margin, and page numbering must be adjusted so that unaffected sections do not change.

The `\revision{filename}{# pages}` macro will read in the file specified by `{filename}`, and will format it so that it takes up `{# pages}`. For example, if the TEXist entered `\revision{sectionone}{3}`, TEX would input the file `sectionone`. If the file took up more than three pages, say for example five pages, the pages would be numbered 1, 2, 3, 3a, 3b. If the file only took up one page, the pages would be numbered 1, 2, 3, but pages 2 and 3 would say "This page intentionally left blank."

The `\changebar{# lines}` macro puts a change bar in both left and right margins for `{# lines}`. This macro is very primitive; it does not split across a page, and it does not account for non-standard spacing between lines.

8. Macros for Classification Markings

To eliminate the hand-stamping of page classification markings for classified documents, we now use `\S`, `\C`, and `\U` macros to mark secret, confidential, and unclassified paragraphs. When the document is TEXed, the `\S` and `\C` macros

will write the page number and classification type to a classification file. The output routine reads in the classification file from the previous run to determine the classification markings for the current run. This method of determining the page classification requires that the document be TEXed twice, but it is the most flexible method that we have found. The `\S`, `\C`, and `\U` macros can be used almost anywhere, for example, inside equations, inside tables, etc.

9. Macros for Verbatim Text

Occasionally we need to include files that we do not want TEX to format. The file `VERBATIM_MACROS` contains macros which will turn off all of TEX's formatting capabilities. When the TEXist needs one or more lines of verbatim text, they can use the `\begintt...\endtt` and `\beginsmalltt...\endtt` macros. If they want an entire file to be copied in verbatim, they can use the `\listing{filename}` and `\smalllisting{filename}` macros. All of these macros use the `\tt` font; the `\small` versions use the `\cmtt8` font.

10. Macros for the Requirements and Design Specification Systems

We developed Data Dictionary systems to support the generation of the Requirements and Design Specifications. These systems are online, menu-driven systems that store information about our software. They were written to generate TEX-compatible output files, which can be directly `\input` into the appropriate section in the Requirements or Design Specification. The output files contain a variety of multi-page tables, TEX "pictures," and dictionaries.

11. Macros for Test Plans, Test Procedures, and Test Reports

The Test Plan document defines the formal tests which will be performed on the software. Every test is broken into test events. Each test event consists of a description of the event, and the expected real time and post-test results. The Test Procedure document defines the exact steps that the test operator must follow to run the test, and the expected results for the real time and post-test portions of the test. The Test Procedure is used during the formal test to record the test results, and to verify that the actual results match the expected results. The Test Report document summarizes the results of the formal test, and explains any discrepancies.

The Test Plan macros number test events using the `\initevent{title}`, `\navevent{title}`, and `\testevent{title}` macros (the `init`, `nav`, and `test` refer to different modes of our software). The numbering, headers, and indentation for sub-paragraphs within the test events are controlled by `\scenario`, `\realtime`, and `\posttest` macros. These macros ensure that every section has consistent indentation and sub-paragraph titles.

The Test Procedure macros use the same `\initevent`, `\navevent`, and `\testevent` macros to number the events, and also use additional macros to format the operator actions and expected results. For example, the following text:

```
\initevent{Initialization}
\beginoperator
\optime   12:00:00
\opdesc   Turn on the receiver.
\opaction Turn the mode switch on the CDU from OFF to INIT.
\expected Receiver will go through BIT, and the
          ``TEST COMPLETE'' message will be displayed.
\actual
\endoperator
```

will produce:

3.4.1.1–I1: *Initialization*

1. Action Time: 12:00:00

   Description: Turn on the receiver.

   Operator Action: Turn the mode switch on the CDU from OFF to INIT.

   Expected Values: Receiver will go through BIT, and the "TEST COMPLETE" message will be displayed.

   Actuals or Pass/Fail: ________________________

The Test Report uses the same macros as the Test Procedure, with the addition of some special macros to handle discrepancy reporting.

The Mistakes

After working with TEX for almost two years, we can now look back and see that there were some problems that we could have avoided.

The first document that we generated using TEX was a 1500-page Design Specification. We ran a program against the 1000 modules in our software to extract and format text information, and then had over 2000 `\input` commands to include these small text pieces into the document. As more experienced people could guess, we ran into the infamous "`TeX capacity exceeded, sorry`" message about 45 minutes into the run. We were running TEX on our VAX 785 at a speed of about 10 pages per minute, and kept crashing around page 500. We were rather frustrated at 4:00 am after the third crash, but eventually we TEXed the document in three pieces to get it to work. We have now taken several steps to avoid this problem. We no longer assume that TEX has infinite storage capability, so we merge the smaller text files into larger ones that do not put such a strain on TEX's memory capacity. We have also upgraded to a VAX 8800, which runs TEX at about 100 pages per minute (on a dedicated machine), so if we have a problem it no longer takes an hour of TEX time to find it. One of our local TEXnicians has also modified the Kellerman and Smith VAX version of TEX (which we now refer to as SuperTEX) to double the memory capacity.

One of the hardest lessons to learn was to trust TEX's ability to decide upon the best format for equations. We often fought to format an equation the way we thought it should be done, rather than using TEX's way. With a little experience, we realized that TEX knew how to typeset equations better than we did, and that if we were having a lot of trouble getting something to come out right, we were probably doing it the wrong way.

We had similar experiences with text formatting. We continued to use old formats for our documents, simply because a precedent had been set. Some of these formats were very cumbersome to enter in a normal document processing system, and became nightmares when trying to make TEX handle them. For example, the original format for Test Procedures defined all operator procedures in a six-column table format. We used the `\halign` construct to format these tables, with special provisions for page breaks. Our TEXists were tearing out their hair over missing `\cr` or `&` symbols, and the final result was hard to read and even harder to work with. We realized that a different format would solve a lot of problems, so we changed the tabular format to a titled paragraph format (which TEX loved), and the results were much easier to enter, looked better, and were loved by all (especially the bald TEXists).

Summary

We now use TEX to generate all of our government standard documentation. The results are impressive; the documents look professional and are in accordance with government standards. We have developed other macro packages for internal use, including ones for overhead projector slides, letters, and status reports. After some initial negativity (usually directed at TEX's complexity), even some of the die-hard WordStar and Runoff fans are converted. Our TEXnicians now regularly get requests for macros to do things that would be impossible with any other document processing system. There are still some headaches with TEX (such as the two days I spent trying to get change bars in the margins), but in general TEX is extremely reliable and flexible.

TEX has helped us to reduce the time necessary to generate complete and consistent documentation. By using TEX and our other software documentation tools, we can spend time on the content of the documents rather than on consistency checking and formatting concerns. It is unfortunate to note, however, that we can still produce "bad" documents. We can ensure that they are consistent, and that they are "pretty," but we still don't have a way to ensure that they are well written. Perhaps a future enhancement for TEX?

Implementing TEX in a Production Environment: A Case Study

Erik Jul

OCLC Online Computer Library Center, Inc.
6565 Frantz Road
Dublin, Ohio 43017-0702

ABSTRACT

This paper presents a case study based on experiences implementing TEX in a document production environment. The discussion covers aspects of document production that may be affected by the introduction of alternative document formatting methods. Recommendations are offered which may help current or prospective users to integrate TEX successfully

Introducing alternative document formatting methods into an existing production environment requires careful analysis, planning, and implementation in order to be successful.

Alternative typesetting and page makeup technologies affect all aspects of document preparation, not just the final printed pages. New production methods raise questions about software, hardware, space allocation and equipment location, document analysis, staff training and work assignments, work flow, time estimating and production scheduling, integration with existing production methods, and cost.

My comments derive from experience implementing TEX in the Documentation Department at OCLC Online Computer Library Center, Inc. OCLC operates a database of 18 million bibliographic records accessed by more than 7,000 libraries worldwide that connect to the OCLC Online System to create or modify records online and produce offline products such as catalog cards or bibliographic records on magnetic tape. The Documentation Department supports the publication needs of the organization.

I will describe the production environment into which TEX was introduced, factors that contributed to selecting TEX, and implementation of the system for document production. Recommendations for current or future TEX sites are offered.

Existing Production Environment

TEX was introduced into a medium-sized Documentation Department that produces a wide range of documents such as user manuals, technical bulletins, re-

search reports, fliers, brochures, promotional materials, newsletters, and books. Documents for internal use such as office forms, the *OCLC Standards Manual*, the *Employee Handbook*, and internal systems documentation are also produced.

The department has a staff of 26 including word processing technicians, writers and editors, graphic artists, typesetting technicians, distribution specialists, and managers.

The Data General CEO office automation system provides dedicated, centralized word processing capabilities accessed throughout OCLC by terminals in or near employee work areas. Documents are printed on various devices at resolutions ranging from draft-quality dot matrix to 300 dpi letter quality laser output. For typesetting, files are transmitted via CEO to the Compugraphic MCS 8400 phototypesetter. Typically, type is produced as galley and manually pasted up to make pages; headers and footers are added separately. Page previewing allows full page makeup but few documents are typeset in pages.

Graphic images are created manually or by using various microcomputer-based graphics programs for inclusion with the galley when documents are pasted up. In-house duplicating and two-color printing services are available.

Ventura Publisher by Xerox Corporation provides additional desktop publishing capabilities for use by writers and graphic artists. The software runs on Wyse 286 microcomputers with an attached QMS PS800 PostScript printer.

Selecting TEX

The decision to use TEX as one document production alternative in the Documentation Department was influenced by the following technical considerations:

1. Existing TEX software
2. Staff experience using TEX in other OCLC units
3. Existing site license for ArborText Preview software
4. Existing network of Sun Workstations and file servers
5. Ethernet local area network
6. Available system administration services
7. Large library of digitized fonts

Others at OCLC had used TEX before its implementation in the Documentation Department, and OCLC owns a site license to ArborText's Preview software, an implementation of TEX for Sun computers.

In the OCLC Office of Research, TEX was used in the Graph-Text project. The Graph-Text system provides online display and local printing of typeset quality pages, including graphics, figures, and tables, produced from publisher's

typesetting tapes or files created by optical character recognition (OCR) scanning. TEX was used because of its typesetting and formatting capabilities and output on various low-resolution devices.

In addition, several OCLC staff created an extensive library of digitized fonts using the METAFONT software. Fonts were designed for use in the Graph-Text project and other potential OCLC products and services. Current fonts include two different serif fonts, sans serif, decorative, and Slavic fonts, as well as a special character set developed for the Library of Congress.

In addition to staff expertise and large font libraries, OCLC has an extensive computing environment for internal development. When the Documentation Department was investigating TEX, existing TEX facilities were mounted on Sun network file servers. The Sun-3/50 Workstation was selected for use as a TEX workstation because of its desktop design; large, high-resolution monitor; multiple window-management capabilities; mouse pointing device; 68020 processor running at 15 MHz; and built-in Ethernet connections.

The Sun Workstation provides high-speed processing to effectively create, store, and manage TEX files. The Sun windowing system enables simultaneous display of multiple files such as a document file, font file, macro file, preview file, and Sun shelltool window. The Sun-3 Workstation also provides an easily operated point-and-click text editor. These tools provide efficient and effective data manipulation and greatly facilitate the use of TEX.

By using an Ethernet connection to an existing local area network, the Documentation Department benefited from Sun network file servers and system administration provided by the OCLC development environment. Network access to multiple peripheral devices, such as Pyramid support processors, Sun file servers, or shared printing devices reduced the effective per unit cost of the workstations. Start-up costs were limited to a diskless workstation and laser printer.

A network configuration provides assured power supply, regular tape backup and file restoration procedures, monitoring by operations personnel, and the added safety of protection against power surges, fire, and other hazards attendant to stand-alone systems in an office environment.

Many general factors made TEX an attractive option for the Documentation Department: staff expertise and experience with TEX over many years; large libraries of digitized fonts; existing software and site license; in-house TEX, Sun, and UNIX system support; and extensive hardware and Ethernet connections. Additionally, the Documentation Department saw the possibility of the following:

1. Reducing the typesetting load and costs
2. Upgrading certain existing documents from word processed copy to typeset laser output
3. Reducing labor-intensive paste up by graphics staff

4. Allowing writers to expand skills to include page composition
5. Using computer processing capabilities for document preparation beyond word processing
6. Alternative to the peak and valley scheduling in typesetting and graphics

TEX offered the possibility of reducing the typesetting load and associated costs. Savings could be realized by shifting selected documents from typesetting to TEX for production. Documents that are frequently revised or short-lived such as transparencies, product price lists, and equipment set-up or reference cards could be produced adequately using TEX laser output. Documents printed on a laser printer would reduce the use of expensive phototypesetting film. Page makeup capabilities could affect cost savings by reducing manual paste up by graphics staff. Other cost savings could be realized by using preview software to reduce the paper output of the laser printer and extend the effective life of the device.

Documents produced in word processed copy such as technical bulletins and research reports could be "upgraded" to typeset quality. Documents produced on stock preprinted with header information could be replaced using laser output for header and body copy.

TEX also provided staff an opportunity to expand skills to include page composition. The precise typographic controls of TEX are similar to those offered by the Compugraphic MCS 8400 and can meet the most stringent requirements.

In sum, TEX offered the Documentation Department alternative document preparation capabilities that could positively impact the work flow of the department, provide a middle ground between typeset and word processed copy, and reduce the cost of labor and materials used to produce documents.

When other technical and departmental factors were added, TEX appeared to be a low-risk, low-cost, high-powered, and versatile alternative to the document preparation in the Documentation Department. Therefore, two Sun-3 Workstations and an Imagen 8/300 laser printer were purchased and installed.

Training

Before the equipment was installed, one writer/editor from the Documentation Department received beginning TEX training during a three-day seminar off site. The instructor was highly skilled in TEX with experience producing complex documents including mathematical textbooks. Training in elements specifically related to a production environment, however, was lacking. Controlling basic functions such as hyphenation, widows and orphans, hanging indention, two-column output, and running headers and footers could have been emphasized, along with the development of macro files containing generalized specifications for documents of the same class.

Training was scheduled to precede equipment installation by several weeks. Installation of the equipment, however, was delayed by several months, so the initial value of the training was diminished. Informal training continued in house with other, more experienced OCLC staff. Lack of access to equipment and limited time hampered these efforts.

Of concern to managers and other staff was the complexity of TEX. Initially it was viewed as too difficult for many to learn, and this was considered a drawback. Training of additional departmental staff was initiated only after document production had begun.

Production

TEX seemed to offer much to the Documentation Department but we were not yet ready to take full advantage of the system or realize its benefits. When the equipment was installed, substantial work remained before document production could begin. File transfer capabilities among existing systems (Data General, Pyramid, Sun, VAX, and microcomputers) and data storage (diskette, tape) were investigated, and appropriate software was obtained and configured.

When all was in place, production was still several months away. While many potential uses of TEX had been identified, as had documents that *could* be produced, plans were not as complete in identifying documents that *would* be produced. Moreover, when documents were identified for TEX production, current production schedules allowed little time to practice TEX skills or begin the process of moving document production to the TEX environment.

New production methods must be integrated with the existing production cycle. A careful analysis of existing production schedules and deadlines will help determine when a document or document class can be shifted to TEX. If possible, choose simple documents without pressing time constraints. Production schedules for TEX documents should allow for a learning curve. This approach will provide experience with a real document, enhance TEX skills, reveal weaknesses in the document preparation methods, provide valuable information for future scheduling and time estimating, and help ensure that existing production schedules for other documents are not delayed.

1. Document Specifications and Macro Files

Before any documents were produced, document specifications were determined and sample pages were produced and circulated for review and approval. Macro files for each document type were created according to specifications to simplify the task of encoding TEX commands. These macro files contain all font and page specifications. For example, the simple command \a{} controls all typesetting specifications for A-level heads including space above and below, automatic numeration, rule lines, font commands, and penalties to prevent headings from appearing at page bottom. To the extent possible, all foreseeable typographical

specifications are defined and documented in a macro file. This has the obvious advantage of ensuring consistency within and between documents of the same type, facilitating the adjustment of document specifications by locating the commands in a single file, and minimizing the amount of TEX coding that needs to be entered in the document file.

Since installation, the OCLC *Working Papers* and *Research Report* series have been transferred to TEX and several such reports have been produced; two directories have been produced, a phone directory for internal use and a Network directory for external distribution. Macro files govern the format of each document type.

2. Work Flow

Implementing TEX had several unanticipated effects on production work flow. Text files for documents are created using microcomputer word processing software, the CEO system, or a Sun Workstation. Because word processing formatting and text attribute commands are not needed, writers are instructed to remove these commands before saving the file. Files are loaded into the Sun system from diskette or tape. Documents are generally submitted to the department without embedded TEX commands.

For previously word processed documents such as OCLC research reports, TEX adds the additional step of embedding codes in the file. For all documents produced to date, codes have been entered by a writer/editor. This method of text entry and revision does not effectively make use of the skill, speed, and precision of word processing staff, who could enter commands that are marked on a hardcopy of a document. This procedure, while intuitively apparent, has not yet been implemented.

A word processing technician and typesetting technician have since received fundamental in-house training in Sun editing techniques and the basic concepts of TEX. Plans call for shifting data entry and text editing to them. In time, developing page specifications and macros could also shift to word processing staff or other writers.

While the time to produce documents has not been unreasonable, the distribution of effort has not maximized staff capabilities. The largest project to date, a 194-page research report containing 150 tables, has taken 143 hours or slightly less than one hour per page. An estimated 40 hours remain to produce the final document. However, a disproportionate amount of time was spent coding, formatting, previewing, and fine-tuning the document. Lack of experience with TEX accounts for much of this, but a division of labor among other trained staff and more clearly defined document preparation procedures may have provided more time to edit, communicate with authors, and revise the document—the more traditional tasks of a writer/editor.

Success in upgrading word processed documents is recognized, but few type-

set documents have been shifted to TeX. Although certain typeset documents such as price lists, forms, and transparencies seem suitable for production using TeX and laser output, none has been produced.

There is far greater willingness to "upgrade" documents than to "downgrade" typeset documents to laser output, even if the need to typeset documents is suspect. Often such documents are part of a series, and matching other typeset documents is important. Often the difficulty is finding the appropriate time in the document life cycle to shift production methods. There may also be some hesitation in moving files of large or numerous documents from one established production system to another because it is unfamiliar and not widely used by departmental staff, and may be viewed as less accessible, stable or permanent.

Recommendations

Experience has shown that careful attention to the following will ensure greater success when implementing TeX in a production environment:

1. Analyze document classes to determine what documents are candidates for production using TeX. This will provide the first real indication whether such documents exist in your production environment.
2. Select hardware/software to optimize file management and user interface. In our application, the Sun Workstation provides an effective environment to process TeX files efficiently.
3. Analyze work flow to determine staff to receive training. Training may need to include not only TeX commands and procedures, but also learning how to use a text editor, UNIX or another operating system, and procedures specific to your production environment.
4. Obtain TeX training beyond the beginning level. Training should emphasize macro design, development of document specifications, and the skills necessary to handle the typesetting requirements of your documents.
5. Provide in-house training for selected staff. Determine who will perform specified tasks in the document preparation life cycle and provide appropriate instruction. Develop in-house procedures manuals.
6. Allow time to develop and document style sheets and corresponding macros. These should be in place before production begins. Produce sample pages for critical assessment and fine-tuning.
7. Design TeX macros to achieve greater simplicity in processing and achieve uniformity among documents of the same class. Macro commands should be meaningful, facilitate text markup, and minimize key strokes needed for coding.
8. Analyze production schedule to optimize introduction of new production methods and allow for start-up time. Determine when a document or doc-

ument class can be shifted to TEX for production. Initially, choose simple documents that do not have critical deadlines.

9. If staff resources allow, assign tasks to appropriate personnel. Define who will do data entry, text editing, TEX coding and processing, and page composition, previewing, and fine-tuning.

10. Determine document standards for various stages of production. Production using TEX can often be reserved for final copy.

Conclusion

Our experience producing documents using TEX reveals both the benefits and pitfalls of implementing TeX as an alternative document production tool. TEX provides all the typographic controls necessary to produce attractively typeset documents even at the relatively low resolution of 300 dots-per-inch. Powerful and comprehensive macros simplify the coding process. This allows word processing technicians to enter the codes without requiring extensive TEX training. Macro commands also help ensure uniformity among documents of the same type. By developing files of macro commands defined by document type or function, making global changes to a document format is easy. This approach is most effective when documents of various classes are defined such as, in our case, research reports, working papers, and directories.

Because simultaneous access to multiple text or macros files is often desirable, the file management tools provided by the Sun Workstation provide an efficient work environment. The large, high-resolution monitor and previewing capabilities seem essential.

Integration of the technical capabilities provided by both TEX and the Sun system with other existing document preparation systems, current production schedules, and the skills of departmental staff continues with time. As we gain experience with TEX we are better able to define the tasks involved, determine who needs to be trained to perform those tasks, and define the type and the level of training necessary.

Implementing TEX in a production environment requires more than just installing the appropriate hardware and software. Integration of the technology into the existing environment is essential, and, because many aspects of document preparation can be affected, careful planning is required. Attention to some of the factors I have described may help current or prospective users integrate TEX into a production environment.

How and Why a Trade Typesetter Chose TEX

PETER TONKIN AND ALEX WARMAN

P. Tonkin
Trade Graphics Pty. Ltd.
445 Graham Street
Port Melbourne 3207
Australia

A. Warman
TEXworks Pty. Ltd.
157 Danks Street
Albert Park 3206
Australia

ABSTRACT

During early 1987, it appeared that changes were starting to gain momentum amongst local Melbourne typesetting businesses. In response, Trade Graphics began an investigation of desktop publishing and electronic publishing. This revealed that many typesetting businesses were either in financial difficulties because they ignored the changes which were taking place or some had become high-resolution reproduction bureaux for clients with their desktop publishing systems. Trade Graphics made a clear decision that, despite the introduction of new desktop and electronic publishing systems, the traditional typesetters' craft and expertise would continue as the core of a sound business.

This led to the development of a strategy which was aimed at getting a major increase in capability to improve service to clients, internal effectiveness and profitability. In addition, an attempt was made to identify potential areas of business which would be most fruitful for typesetting work in the future. This would allow the use of TEX as a front-end for the existing equipment and to delay the choice of a replacement phototypesetter.

This paper will outline how the strategy was developed and the aspects of TEX, TEX-related products such as the range of print drivers, and the TEX community, which have become part of Trade Graphics on-going business plans.

Changes in Typesetting Businesses During the Early 1980's

Trade Graphics Pty. Ltd has been a trade typesetting business since 1979. Over the years it has had a variety of market niches which it has serviced, ranging through advertising material, company annual reports, small booklets such as company pricing documents, educational text books, fiction books, and scientific academic journals.

In the early 1980s as microcomputers began to become visible in the commercial world, the possibility of using some of this technology seemed to open the door to *direct capture of clients' data electronically.* It was already clear that if this could be achieved, there were significant cost savings in avoiding re-keying and checking. Further, there was considerable scope for improvement in turnaround times; all benefits which could be passed back to the clients and win more business or increase the capacity at Trade Graphics.

1. Modems for Direct File Transfer

In 1982 the idea of direct capture of clients' key strokes on data files led to trials in the use of modems for transmission of client data files from their computers to Trade Graphics. If successful, the facility was to lead to a situation where dial-up modems would be available at Trade Graphics 24 hours a day so that data files could be transmitted at any time. By 1984 this approach was discontinued primarily because of the difficulties with clients' staff who often did not have the necessary knowledge and skills to set-up and operate data communications equipment for such transfers.

2. Diskettes for Direct File Transfer

When it became clear that using modems for direct transfer of client data files was not going to be a success, attention turned to trying to get data files on diskettes. An initial perceived disadvantage was the need to arrange pick-up. This turned out to be less of a problem as diskettes were robust enough to be thrust under doors at all hours without problems arising.

To pursue this approach a Baber disk reader machine was purchased. At that time in 1984, the most popular diskette size was 8 inch with 5 1/4 inch rapidly coming into vogue. More recently the 3 1/2 inch diskette has joined the game. The Baber is produced locally in Melbourne which meant that the *experts* could be consulted if some diskette format proved troublesome.

Today the Baber reads several hundred diskette formats including different size diskettes, different computer brands, different operating systems, and different file storage formats used by some software (particularly word processors with their own directories). It is being used more and more and has paid its way many times over and will continue doing so. The current model of the Baber

comes with the three diskette drive sizes as standard and retails for between $Aus 12,000 and $Aus 15,000.

While the Baber also has facilities for code translation and systematic data manipulation/transformation, these have been little used because of documentation limitations for a typesetting business short on detailed knowledge of the bits and bytes jargon of computer hackers.

3. Lowering Workstation Costs

Another motivation for the focus on direct capture of clients' data files was the *cost per workstation* for the re-keying of material from clients.

Front-end equipment consisted of 6 Compugraphic Editwriters, a single workstation and phototypesetter combined. In 1981 a Compugraphic MCS front-end system with 2 workstations and an 8400 phototypesetter were purchased. In 1984 a second MCS front-end with 4 workstations and another 8400 phototypesetter were purchased. The special nature of the user interface, ie., the file system, the text editor, back-up utilities and the connection to the 8400 gave no choice for a growing business other than more equipment from a single supplier. This is sometimes called the *lockeminski* effect, which is not an exclusively Russian problem!

The cost of an extra workstation in 1985 was about $Aus 13,000 and by 1988 this has dropped to about $7,000 per workstation on the MCS front-end.

In 1984, in an attempt to lower the cost per workstation, Trade Graphics turned to the use of Commodore 64s for simple data entry with the use of an RS232 interface to transfer data into the Compugraphic front-end systems. Using the Commodores with simple text editors gave the choice of entering some typesetting commands with the text, or not.

Consequently this offered lower costs by having keyboard operators who were simply typists while the skilled typesetters only added commands at a later stage. Hence the skilled craftsmen were able to concentrate solely on the manipulations necessary to achieve the desired layout and high quality final effect.

Another broad staffing problem was involved too; namely that training of typesetters had not been moving quickly enough to keep up with the computerised typesetting systems. There seemed to be a growing disparity between the old highly skilled manual craftsmen and the computerised systems which had some limitations compared to the manually produced high quality output. The recently trained typesetters did not know enough of the old knowledge to *bend* the new systems where necessary to minimise shortcomings. This provided another aspect of the approach to lowering data entry costs in focusing trained typesetters specifically on the end product.

The staffing problems and the limitations of the computerised typesetting

particularly affected the look and finished quality of books compared with some older book styles.

4. Insularity in the Typesetting Industry

During this period of taking the plunge into trying to use new technology, Tonkin found it particularly galling that most typesetting business people around Melbourne were very reluctant to exchange views and experiences. There seemed to have been a suspicion that talking to other typesetters might reveal secrets or run the risk of clients being pinched.

For a typesetting business like Trade Graphics this made forays into the world of microcomputers, modems, and the proliferation of diskette formats a fairly lonely exercise in frontier-bashing.

The companies supplying specialised front-end systems and phototypesetters only served to add to this situation. There was little effort put into forming *user groups* which are common in the computer world and other commercial and industrial areas. Such groups, if formed, might have pushed for standards in certain areas such as connectability of equipment of different brands and some sort of front-end uniformity.

5. Purchasing Computer Equipment Directly from Computer Manufacturers

In the early 1980s it is interesting to observe that very little general purpose computer equipment could be purchased direct from the computer manufacturers; it could only be bought from the phototypesetter suppliers who would resell computing equipment tightly integrated into their own systems.

This is rapidly changing now and has brought computer manufacturers who are used to competing in a vigourous, even cut-throat, market into direct contact with printing and typesetting businesses. The desktop phenomena is briefly discussed below, but this direct contact has brought considerable price reductions, general purpose software—particularly for operating systems—and improved ease of connecting a variety of equipment for use of lower quality printers, disk storage, and data transmission.

Desktop Publishing Makes an Appearance

1. Effects and Consequences

In the early 1980s word processing made major inroads into the commercial world of the office. Time-sharing terminals and microcomputers rapidly replaced typewriters as the tools for producing written material.

In 1984/1985 the trio comprising the Apple Macintosh computer, the Laserwriter driven by PostScript and using Adobe fonts, and Pagemaker from Aldus made a dramatic impact on the word processing market. It gave the gift of combining pictures with text, and eye-catching demonstrations could seduce you with the impression that you had a superb graphic artist hidden in the little box on your desk!

To the world of the commercial office, only recently weaned off its trusty typewriters, this seemed like another revolution in the making. Indeed the 1987 annual PC Show in Melbourne had desktop publishing everywhere. The impression given by many was that the need to send documents out to the local typesetter was a thing of the past; you could easily do it yourself. Obviously this rapidly became a great worry to the technologically aware typesetters and a great attraction to the office world.

2. Desktop Publishing Takes a Bite

By the end of 1986 Tonkin already could observe changes underway amongst the local typesetting businesses. The desktop publishing activity was brought into being by a large volume computer manufacturer, not by anyone connected with the printing/typesetting world. They offered low-cost systems to the clients of the typesetters, not to the typesetters themselves (this came a little later). There were two major developments:

1. A number of small typesetting companies went under as they found much of their work such as leaflets, brochures, reports, and forms suddenly disappeared.
2. Soon after a high-resolution postscript phototypesetter became available, some businesses started offering a *reproduction* service by purchasing Macintosh computers to read Macintosh diskettes containing final copy, which had been produced by the client. These businesses were often formed by people from a general business background or a computer background rather than traditional typesetting.

It would seem that the insularity of many typesetting business people left them vulnerable to rapid changes in technology and consequent business strategy. Some who have reacted seem to have to turned away from their traditional skills and become equipment operators for reproduction services. Only time will tell if they have made a wise choice.

Desktop Publishing versus Electronic Publishing

Meanwhile back at Trade Graphics, by early 1987, Tonkin had become concerned about several facets of the medium to long-term future:

- the effects of desktop publishing as described above
- if desktop publishing was the way to go, could his Compugraphic 8400 be used as an output device
- the lack of recognition of desktop publishing by Compugraphic and the locked-in feeling with the Compugraphic front-end and photoypesetter

In April 1987, Warman was commissioned to survey computer hardware and software, desktop publishing in particular, to specifically address the concerns above. As he had used *troff* on a Unix system and had been looking into the use of TEX as a possible replacement, these were included in the study. The study[1] was completed in August and included some of the following conclusions:

1. Desktop Publishing

- Desktop publishing products such as Pagemaker and Ventura are excellent for page composition where graphics and text were to be combined.
- The desktop publishing products had some limitations of typographic accuracy in some areas, for example handling of kerning and a large range of typefaces or fonts.
- Pagemaker and Ventura could handle small documents (up to 100 pages), but were likely to be slow and impractical for larger books.
- Pagemaker and Ventura ran on Macintosh or MS-DOS machines which meant a high cost per extra screen relative to a time-sharing computer.
- High-resolution output devices were limited to Linotronics for Pagemaker.

2. Electronic Publishing

- TEX is a high precision software typesetting system which handles automatically kerning pairs in its fonts, ligatures, high-quality hyphenation and justification of complete paragraphs. Commands are clearer than traditional two-character commands and document style ensures consistent handling of page layout, table of contents generation, pagination, indices, bibliographies, etc.
- TEX is far and away the best mathematical typesetting tool.

[1] Consultant Report for Trade Graphics Pty. Ltd., August 1987.

- TeX runs on a wide variety of computers including Macintosh and MS-DOS microcomputers as well as time-sharing systems such as UNIX.

  This offered a real choice for front-end equipment and direct from the manufacturers, not via OEM (Other Equipment Manufacturer) agreements with consequent markups and *lockeminski* effects.

- TeX's device independent output format has enabled output drivers to be commercially available for 300 dpi laser printers and several phototypesetters.

  This too moved toward real choice of output devices, almost independent of the front-end equipment, and certainly independent of the suppliers of the front-end hardware and software.

Other relevant conclusions were to continue the work on capturing client data files and progressing toward managing these files for the clients, and to move to the use of 300 dpi laser printers for draft copy, even if there were limited typefaces available.

Commercial Advantages of TeX in a Typesetting Business

In late 1987 Trade Graphics made a decision in principle to purchase an MS-DOS machine to run TeX. TeX seemed by far the best tool for the book publishing and academic publications while it seemed at least as good as the desktop publishing tools for smaller documents, that is apart from the inclusion of graphics.

ArborText were very co-operative in sending a copy of DVICG for the Compugraphic 8400 on the basis that if it were not satisfactory that it could be returned. In fact it took six weeks for the local Compugraphic agent to supply the cable specifications at the phototypesetter end; but when this information was located, the tests ran correctly straight away.

In early 1988 this triggered a firm decision to use TeX at Trade Graphics as the front-end system, replacing the Compugraphic MCS system. The TeX system was to be run on a Compaq 386/20 Desktop under MS-DOS. The ArborText output driver, DVICG, would be used to output to the Compugraphic 8400. The major strategic aspects contributing to this decision were:

- The availability of TeX on a variety of computers would make a fairly painless switch from a single workstation, such as MS-DOS, to a time-sharing system if the number of required workstations increased sufficiently. Workstations could be cheap ASCII terminals on a time-sharing system.

  This is general purpose computer equipment and is directly available from the manufacturers rather than from the phototypesetter manufacturers via their OEM agreements.

- The availability of output drivers from ArborText for several phototypesetters making choice and competition a reality.
- The availability of 300 dpi laser drivers with the same user interface as the phototypesetter drivers. In addition the Bitstream fonts give a reasonable chance of matching typefaces to the Compugraphic 8400 fonts (Trade Graphics use about 250 typefaces).

  An interesting development here is the PIP (Page Image Processor) from Autologic which makes the same fonts available on devices ranging from 300 dpi and 600 dpi lasers to their APS phototypesetters.
- Excellent quality documentation compared with the normal manuals from computer suppliers or phototypesetter manufacturers.
- The TeX User Group and a lively TeX community sharing information which is efficiently circulated by methods such as TeXhax. This in complete contrast to previous local experience of insularity amongst typesetting businesses.
- TeX has encapsulated and automated much of the traditional skills of the high quality typesetter's craft. This has the potential to restore much of the high quality of the old manual skills, if TeX is used by a typesetter with the *old knowledge.*
- TeX has automated many aspects of document structure which ensure consistency throughout a document and allow rapid reprocessing of a document if changes are suddenly necessary. This is relevant to pagination, page layout and numbering, tables of contents, indices, etc.
- TeX for mathematical typesetting.

Conclusions

It is not necessary to sing the praises of TeX's typesetting capabilities to the TUG conference. However, it is of interest that some aspects of a typesetting environment using TeX make powerful commercial sense. These are the freedom to change front-end equipment and the phototypesetters with minimal disruption to the operations. In addition, such a range of choice gives a considerable negotiating leverage. Lastly an organisation such as TUG and the TeX community is vastly different to the insular world from which Trade Graphics is slowly emerging.

An Experience in Textbook Production

JAMES D. MOONEY

Dept. of Statistics and Computer Science
West Virginia University
Morgantown, WV 26506
jdm@a.cs.wvu.wvnet.edu

ABSTRACT

This paper discusses some recent experiences and lessons learned in the production of over 700 camera-ready pages for an operating systems textbook, using TEX and a Digital LN03-plus printer in a UNIX environment. During the development of the book, plans changed repeatedly from supplying machine-readable copy in various formats to supplying camera-ready copy. Eventually, almost the entire text was produced by TEX camera-ready, on a laser printer, during several weeks of intense pressure to meet publishing deadlines. Original copy was prepared in a form suitable for UNIX *nroff*, and translated automatically from *nroff* form to TEX form. Output was produced on a DEC LN03 laser printer, and the importance of fonts tailored to the device was very apparent. Problems were faced in setting tabular material, two-column copy, figures, and other special cases, in fine-tuning appearance throughout, and in selectively correcting final output. Solutions are offered for a few of these problems.

Introduction

A 700-page textbook on computer operating systems (Lane and Mooney 1988) was developed by my colleague Malcolm Lane and myself at West Virginia University over a two-year period. TEX played a major role in the development of this text, and most of the final camera-ready copy was produced using TEX on a VAX Berkeley UNIX system driving a Digital LN03-plus laser printer. The success of the final result owes much to the power of TEX, but the process of getting there was sometimes very painful. It is likely that more and more book authors will want, or be asked, to use a similar approach in the production of their manuscripts. This paper presents our experiences in the hope that the lessons we have learned may be of benefit to those who follow.

Throughout much of the project we planned to use TEX primarily for draft copy. The final production was to be done by the publisher's in-house typesetting systems. We expected to supply the manuscript on magnetic disks or

tapes, to avoid the problems of re-keying, and to develop as simple a method as possible to incorporate the necessary typesetting codes. However, discussions with the typesetting staff gradually revealed some problems with this approach. They were able to accept documents only in simple word processor formats, and the typesetter codes were low-level and obscure. Although our formatting requirements were modest, concepts such as specific types of lists and even some necessary fonts could not be represented directly by these codes. As a result, a great deal of extra translation and manual adjustment was likely to be necessary.

The publishers continued to favor use of their typesetters until deadlines began to draw near. It then became apparent that much time could be saved if we supplied camera-ready copy. Since we were already using TEX with reasonable success for draft output, producing final copy seemed like a manageable task. I was somewhat doubtful that our overworked LN03s could provide acceptable book-quality output, but the publishers examined some early drafts and gave their approval.

I had worked in the typesetting industry before coming to West Virginia University, and had since developed a formatting system design of my own (Mooney 1982). I had followed TEX since its inception, done some TEX consulting, and introduced the system to WVU. I was thus both a more willing and experienced hacker, and a more demanding critic of the output I produced, than most book authors. But I still cannot claim to be a "TEXpert," and problems arose unexpectedly from aspects of TEX which I did not—and perhaps still do not—fully understand.

The use of TEX for working copy had required us to face and resolve some problems, but these were minor compared to the problems of producing full book-quality output under increasing deadline pressures. By the time of the final run, we were alternately praising and condemning TEX and all the other software and hardware we used as we rushed to meet each evening's Federal Express deadlines. We faced and solved some problems, we used some creative methods and some naïve ones, and we learned some lessons that would lead us to do some things differently the next time.

The rest of this paper presents some of the problems we faced and most of the lessons we learned. The next section describes our TEX operating environment, and some ways in which that environment was improved to support the textbook project. The subsequent sections describe specific technical and operational problems that we encountered, and present some conclusions and recommendations for similar projects in the future.

The WVU TEX Environment

We have been using TEX at the WVU Department of Statistics and Computer Science since about 1983 on a set of VAXes running the Berkeley UNIX operating system (currently version 4.3). The output device we have used exclusively is

the Digital LN03 laser printer. We have one standard LN03 and one LN03-plus, each equipped with two memory cartridges. Since no usable UNIX LN03 output driver was available until recently, we designed our own driver program.[1] More recently we have begun using the *dvi2ln3* driver written by Flavio Rose. This driver is much faster than our own but is not yet well integrated into the UNIX environment, and it was not available for our book.

Until our final production push we were running a UNIX-TeX system several years old, with the "Almost Modern" fonts. Red tape at both ends frustrated our attempts to obtain a more current tape from the University of Washington. This problem was overcome just in time with some direct assistance from Pierre MacKay.

We had been looking forward to use of the final Computer Modern typefaces, if only to get rid of the distorted W and M of the **amtt** fonts. To our dismay, the **gf** files of the **cm** fonts supplied on the tape appeared light and distorted on the LN03. We regenerated the most important fonts using with LN03 parameters, realizing for the first time that our printer had a "write-white" engine. The exact parameters used were those published in *TUGboat* by Stan Osborne.[2] The results were more satisfactory, although the improvement over many of the **am** fonts seems to be a matter of taste. Output on the standard LN03 and LN03-plus was distinctly different, and we elected to use the LN03-plus exclusively for final copy. To achieve better resolution, all output was run magnified 20%, and reduced during publication.

The formatting system most commonly used in our department was the *nroff* system available on UNIX.[3] My co-author was not familiar with TeX. I had experience with both systems but found *nroff* easier to use (with appropriate macros) for most routine documents. The processing time required for *nroff* was also much lower than for TeX, due in part to our slow output driver. For these reasons we found it desirable to be able to work with *nroff* for familiarity and speed, yet convert to TeX's more polished output when desired.

We met this need by developing a procedure using the UNIX script-driven editor *sed* to translate a limited set of *nroff* macros to corresponding TeX macros. Some examples of the type of translations possible are shown in Figure 1. The original intent was to maintain the ability to produce either *nroff* drafts or TeX versions from a single source. In practice we soon began using TeX output for drafts as well, and rarely produced *nroff* output. However, the *sed* translation was very useful in providing extra flexibility when the desired behavior could not easily be achieved with TeX macros alone. For example, a simple line in

[1] Information on this driver is available from the author on request. Learning how to prepare fonts for the LN03 is a story in itself!

[2] See *TUGboat*, April 1987.

[3] *nroff*'s more powerful cousin *troff* was not used since we lacked a version which could drive the output devices we had available.

the *sed* script could be used to replace all spaces in certain display regions by non-expandable space characters, without disturbing spaces in other parts of the document. Optional macro arguments, not possible with TEX alone, could be achieved in some useful cases. Reasonably intelligent translation of double quote characters to the appropriate matched sets of single quotes was also obtained by this method.

We have since extended the method to assist in translating other *nroff* documents, including pre-existing ones. The same technique can also be applied to an "*nroff*-like" extended language which allows longer macro names and mixing commands and text on the same input line, overcoming *nroff*'s two most exasperating limitations.

Figure 1 – Example Conversions from *nroff* to TEX

nroff COMMAND	TEX COMMAND	ACTION
`.DB`[a]	`\begindisplay`	Begin display region
`.DE`[a]	`\enddisplay`	End display region
`.PB`[a]	`\beginprog`	Begin program display region
`.2C`[a]	`\begindoublecolumns`	Enter two-column mode
`.TG`[a]	`\looseness=-1`	Tighten previous paragraph
`.so filename`[a]	`\input filename`	Include a separate file
`.IT "xxx"`[a]	`{\ital xxx}`	Use italic font
`.FG "title" "n"`[a]	`\smallfigure {title}{n}`	Insert figure, *n* picas deep
`.FG "title"`[a]	`\pagefigure {title}`	Insert figure, full page size
`.H1 "sectitle"`[a]	`\section {sectitle}`	Begin new primary section
space[b]	`~`	Make space non-expandable
blank line[b]	`\blankline`	Accept explicit blank line
`$`	`\$`	Escape special character
`"anything"`[c]	` ``anything'' `	Insert matching quotes

[a] Recognized only at beginning of line
[b] Recognized only in display region
[c] Not recognized in program display region

We chose not to use LATEX for our project primarily due to my preference for precise control over formatting details; this is especially important in meeting design specifications established by a book publisher. I have found it easier to obtain convenience and non-procedural markup using suitably designed custom macros. The *sed* translation provided further flexibility for this process.

Some Technical Problems

1. Page Layout and Vertical Space

The biggest headaches we found in applying TEX to book production concerned efforts to place paragraphs, headings, and displays as we wanted on each page, when that differed from the placement TEX preferred. Our designers wanted no space between paragraphs, and no expansion of the specified space between other elements, even if that sometimes required extra space at the bottom of the page. We could not persuade TEX not to insert extra vertical space in some undesirable places.

We met with some success in the placement of figures, making consistent use of the floating capability of the `\topinsert` and `\midinsert` commands. However, the placement strategy used by these commands sometimes defied understanding. Figures would refuse to appear in what seemed a natural place, or would move from a good place to a very different, poor place after a minor change. In several cases, two figures appeared on the same page, in the wrong order!

We found few solutions for these problems, although we realized that a deeper study of TEX internals might have helped if we had the luxury of sufficient time. In many cases, physical cutting and pasting was necessary. Later, we determined that some of the problems occurred because in our rush, we had failed to install the newest version of TEX itself, and were actually running with an older version.

2. Two-Column Material

Our text had a requirement to set a list of terms in two-column format at the end of each chapter. Each list was much shorter than a page, so they would either fit within a single page, or start on one and end on the next.

Knuth's two-column macros[4] gave us a place to start, but these macros do not contemplate switching from one-column to two-column and back again on the same page. Some straightforward revisions were necessary to accomplish this; our versions are shown in the Appendix.

3. Tables and Figures

The book included a large number of figures. Some of these were line drawings, perhaps with some isolated text. Others were purely text except for rules; these were primarily tables and program listings. The publisher's graphic art department produced most of the drawings. We tried to produce the tables and listings, and some simple drawings composed of horizontal and vertical straight lines only.

---

[4] The *TEXbook*, p. 417.

All figures were to span the full width of a page, but the heights were variable. We used the floating figure placement of TEX as discussed above. Although it would have been convenient to run many tables and figures in place, the specific problems of formatting each figure led us to produce them all separately and paste them in.

Program listings were generated in a "program display" environment, using a simplified set of verbatim commands and the cmtt font. This produced no special problems, but we wrestled with a few special characters and wound up producing several in the wrong font. We would like to see some version of `\verbatim` as a primitive TEX command.

Tables were a nuisance. Even the simplest tables, easily formatted by hand on our terminal screen, could not be kept formatted in the TEX output without mastering the intricacies of `\halign`. The problem is complicated, of course, by proportional spacing. Since the only monospaced font is cmtt, we set some tables in this font when a different font would have been more appropriate. The TEX font library should include a monospaced font with a truly typewriter-like design such as Courier; and there should be a simple alignment mode using ASCII tab characters and tab stops set individually by position, and requiring no special commands at the beginning or end of each line.

We attempted to produce some simple line drawing figures because their spacing requirements were fairly critical and we had some difficulty communicating them to the graphic artists. A good example was a set of memory maps, in which the height of each region had to be proportional to its memory size. A sample is shown in Figure 2. These drawings were built from a collection of special-purpose macros developed on the fly, which had to be fine-tuned repeatedly to fit the available space and to make all the line endings meet. In the final text the blank regions were shaded; the shading was added during production by the graphic artists.

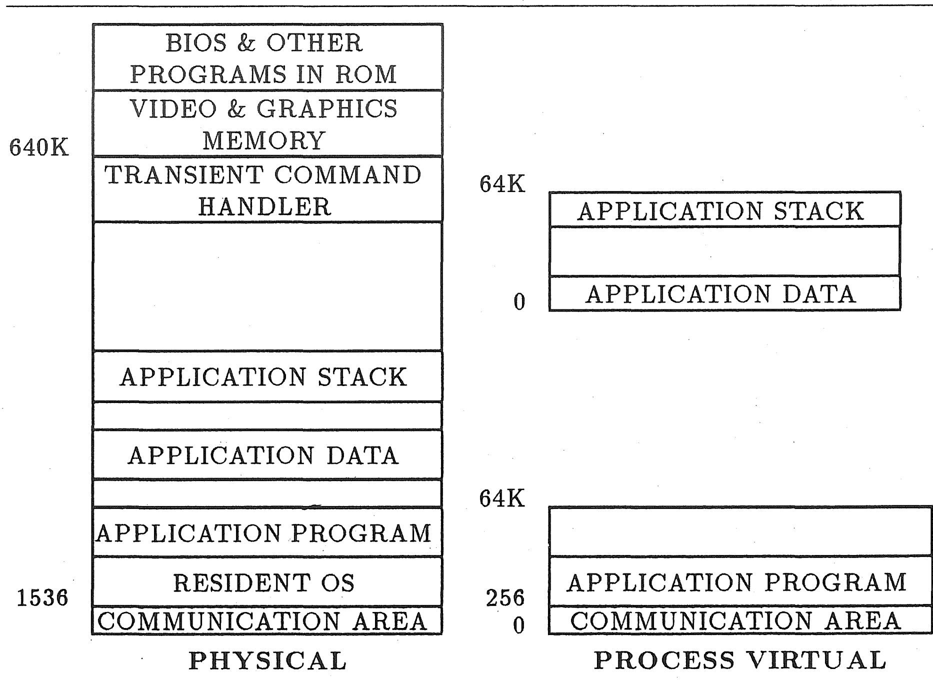
Some Operational Problems

1 Organizing a Large Document

Our complete book included many megabytes of file storage, and many parts needed to be worked on independently. The document had to be maintained as a set of separate files. The book included 23 chapters each from 10 to 25 pages long, plus various appendices and miscellaneous sections. Each chapter, appendix or section became a separate file.

It was never necessary to run the whole book at once. Each chapter was edited and printed separately as needed. In the final production all chapters were run sequentially, since starting page numbers had to be inserted in each chapter after the previous one was done.

Figure 2 – A Simple Line Drawing



All chapters were kept in a common UNIX directory together with the necessary macro files. Separate directories were used for large special sections such as the bibliography and glossary. These sections were divided alphabetically into several subfiles, and kept organized with the aid of UNIX tools such as the *sort* utility.

We used no direct mechanisms in TEX for generating bibliographies, indices, etc. Instead, UNIX tools were used to extract lines of interest from chapter files such as headings or boldfaced terms. These were then turned into tables of contents, terminology lists, and other necessary groupings by a semi-automatic process.

2 Maintaining Draft and Production Versions

Although our original plan was to produce draft copies with *nroff*, we found it more helpful to have TEX output for working copies and also for reviewers' copies. However, the review copies had to include extra space between lines. This required adjustments to a number of spacing parameters, since there was no simple command such as `\doublespace`. Two parallel sets of macros were maintained for draft and production versions. To avoid changing the `\input`

lines in every file, the desired macro file was renamed to the common name **bookmacs.tex** before each run. This was made fairly convenient by UNIX shell commands; it would have been simpler yet if the `\input` command understood UNIX environment variables.

The use of separate draft macros made it possible to use more convenient page numbers of the form "chapter–page" instead of the sequential numbers required in the final text. However, an unexpected difficulty in working with separate draft and production versions was correlating page numbers; a reviewer's comments about "page 13–9" did not always correspond to the pages we had on our desk. Some type of cross-listing of page numbers would have been helpful, but probably difficult.

3 Fine-Tuning the Output

In the final days of production, it became necessary to make frequent minor revisions of most chapters to correct typographical errors and perfect the formatting. This process was impeded by a combination of problems: we had no useable previewer, our output driver was slow, our printers were hard to keep in optimum working order, and TEX did not always cooperate in fine-tuning and reprinting individual pages without affecting others.

By the end of the project the "terminals" we were using to communicate with our VAXes were Macintosh computers simulating VT100s. Simple previewers available for alphanumeric terminals did not provide enough information. Although it seemed likely that a good previewer taking advantage of the Macintosh graphics in our context could be developed, no such program was available. As a result our only reliable proofs were hardcopy versions.

The slowness of the output driver has already been mentioned; details are beyond the scope of this paper. The TEX problems included difficulty in starting processing in the middle of a file, with correct page numbers and all processing parameters intact. Smart output drivers (smarter than ours) can selectively print a specified range of page numbers. It would have helped if TEX itself could also limit its output in this way.

The more serious problem was TEX's insistence on rearranging later pages of a chapter after minor changes to an earlier page, *even if the changed page still ended at the same place.* This behavior was difficult to understand or suppress, leading to many extra runs to correct later pages that had been satisfactory to begin with.

Some Final Comments

1 TEX in a UNIX environment

An effective software tool must fit well into the working environments in which it

will be used; it is not reasonable to ask users to adapt to a new environment for each tool, however well-designed those environments may be. Most experienced computer users prefer their usual editor, for example, to one packaged with a new language processor; and tools which use files designed for easy interfacing with other software are likely to be more successful for this reason.

Common UNIX environments are conducive to programs which can operate on simple alphanumeric terminals without graphics, and which read a single input stream and produce an output stream. In these regards TEX is well suited to UNIX, much more than to a workstation environment where some form of WYSIWYG operation is essential.

An arsenal of UNIX text processing tools is also useful with TEX. The available collection includes spelling and style checkers. Versions of some of these have been adapted to recognize the TEX command structure.

TEX does not cooperate well, though, with other UNIX conventions, especially aspects of the user interface and file system. Terminal output is verbose and not easily suppressed. Use as a filter is difficult, and the hierarchical file system is not supported. It would be desirable (and expected) to be able to create a shell command such as `runtex`, so that typing `runtex book.chap*&` would start background processes to run TEX on a set of files and spool the output directly to a printer, while the terminal continues to be used in the foreground for editing other files. Some of this can be accomplished awkwardly with shell scripts; but much of it is blocked by unwarranted assumptions within TEX. As a small but nasty example, the first period in the filenames given above would make them unrecognizable.

I suggest a modest set of revisions to the TEX user interface and that of related tools for use in a UNIX environment. These could probably be implemented with a straightforward change file. They include the following:

- Allow recognition of selected options on a TEX command line.
- Provide an option (e.g., `-q`) to suppress all terminal output except error messages, and write these on the standard error output.
- Provide options to use standard input and output as alternatives to `.tex` and `.dvi` files.
- Do not require that input filenames end in `.tex` or meet any other form restrictions.

Output drivers can work most conveniently in the UNIX environment if installed as filters for the print spooler. This requires only simple modifications; an option in the spooling command is already reserved for special processing of **dvi** files.

2 The Use of Macros

A low-level formatting language like that of TEX or *nroff* offers great power to its users, but it is too easy to get buried in the details required to do even simple things. Use of such a language is made palatable by a good macro definition facility and, hopefully, a good library of macro definitions. Like high-level programming languages, macros can make it very easy to do complex things, *if* their constructs match what you want to do.

In most working environments, good macros have to be customized. This is not too difficult for reasonably experienced TEX programmers. For routine reports and documents, macro development can be justified because the same formats will be used repeatedly. For a major project such as a book, macros developed for the special needs of the project are well worthwhile.

The spirit of top-down project development can be applied to documents as well as programs, and has proved useful in the formatting of our text. The strategy I advocate is perhaps more extreme than most: if it becomes clear while writing the document that a macro with a certain behavior would be useful, ***assume it exists*** and begin using it. The commands in the text of the document can then be kept as simple (and non-procedural) as possible, and in the great majority of cases the necessary macro can be written more easily than expected.

Conclusion

This paper has presented a review of a number of problems we encountered in the production of a large textbook. We have pointed out some lessons we learned, some possible weaknesses in TEX for such projects, and some pitfalls which we and others may avoid in future projects.

To be honest, only about 650 of the nearly 800 pages we produced using TEX were actually used in the book. Concerned about the total page count, the publishers asked us to reset the appendixes and back matter in a smaller size (about 7 point). This would have led both to procedural problems and unreadable output from the LN03. In the end, all of the appendices were re-keyed (correctly, as far as we can tell) and typeset by the publisher, saving about 25 pages.

BIBLIOGRAPHY

Lane, M.G., and Mooney, J.D. *A Practical Approach to Operating Systems.* Boston: Boyd & Fraser, 1988.

Mooney, J.D. *MFS: A Modular Text Formatting System.* Proceedings of the National Computer Conference, AFIPS Press, 1982.

APPENDIX: Two-Column Macros

```
%================================================
%  (This is file twocol.tex)
%
% doublecolumnout (output routine)
%
%     For pages ending in two column mode.
%     Double column material is in box255 as a single column;
%     possible single column material saved in \partialpage.
%
\def\doublecolumnout{%
        % set parameters for split
        \splittopskip=\topskip
        \splitmaxdepth=\maxdepth

        % compute goal height for each column
        \goalheight=\pageheight
        \advance\goalheight by-\ht\partialpage

        % split column into boxes 0 and 2
        \setbox0=\vsplit255 to\goalheight
        \setbox2=\vsplit255 to\goalheight

        % ship out partialpage plus columns
        \pageout{\pagesofar}

        % restore any leftovers to main list
        \global\vsize=2\pageheight
        \global\advance\vsize by1pc
        \global\pagegoal=\vsize
        \unvbox255
        \penalty\outputpenalty
}

% balancecolumns (output routine)
%
%   Balance columns at end of two-column mode
%
\def\balancecolumns{%
        % copy current page to box 0
        \setbox0=\vbox{\unvbox255}

        % set parameters for split
        \splittopskip=\topskip
```

```
        % compute initial goalheight for each column
        \goalheight=\ht0
        \advance\goalheight by\topskip
        \advance\goalheight by-\baselineskip
        \divide\goalheight by2

        % balance the columns into boxes 1 and 3
        {\vbadness=10000 \tolerance=10000
         \loop
            \global\setbox3=\copy0
            \global\setbox1=\vsplit3 to\goalheight
            \ifdim\ht3>\goalheight
                 \global\advance\goalheight by1pt
            \repeat
        }

        % transfer balanced columns to boxes 0 and 2
        \setbox0=\vbox to\goalheight{\unvbox1}
        \setbox2=\vbox to\goalheight
            {\dimen2=\dp3 \unvbox3 \kern-\dimen2 \vfil}

        % place all material on main list
        \global\vsize=\pageheight
        \global\pagegoal=\vsize
        \pagesofar
}

% pagesofar
%
%     Restore partial page and (balanced) current columns
%     to main list.
%
\def\pagesofar{
        % restore partial page
        \unvbox\partialpage

        % unbox and rebox the columns
        \wd0=\hsize \wd2=\hsize
        \hbox to\pagewidth{\box0\hfil\box2}
}
% begindoublecolumns
%
%   Begin double column mode.
%
```

```
\def\begindoublecolumns{\begingroup
        % store current page so far in \partialpage
        \output={\global\setbox
            \partialpage=\vbox{\unvbox255\bigskip}}
        \eject

        % set output routine for pages that complete in 2C mode
        \output={\doublecolumnout}

        % set page size to remaining column space on first page
        \hsize=\pagewidth
        \advance\hsize by-1pc
        \divide\hsize by2
        \vsize=2\pageheight
        \advance\vsize by1pc
        \advance\vsize by-2\ht\partialpage
}
% enddoublecolumns
%
%   End double column mode
%
\def\enddoublecolumns{%
        % adjust columns on final page
        \output={\balancecolumns}
        \eject

        % restore normal output routine and other parameters
        \endgroup
        \vsize=\pageheight
        \pagegoal=\vsize
}
```

Using TEX to Produce Kennel Club Yearbooks

ROBERT L. HARRIS

Micro Programs Inc.
251 Jackson Avenue
Syosset, NY 11791-4117

ABSTRACT

Many dog clubs publish annual collections of pedigrees of dogs earning AKC titles. The production of these yearbooks is labor-intensive and tedious. One club, by using a pedigree management program and TEX, has streamlined the process. The interface between the two programs was designed so the editor does not have to be a TEXpert to produce a TEX document. Preliminary work shows that halftones from scanned photographs can be included with the pedigrees in the TEX document to eliminate screening and pasting at the printers.

Introduction

Every person who breeds animals and is striving to improve the quality of the animals produced by their breeding programs needs and uses pedigrees. This is true whether we are talking about horses, cattle, cats, goats, or dogs. The pedigree gives the knowledgeable breeder valuable information about the background of his stock and some insight into the probable traits of the progeny of any breeding. While all breeders complete, share, and use pedigrees, I am going to concentrate on dog breeders.

There are three groups of people in the sport of purebred dogs who are interested in pedigrees. First is the breeder. He uses pedigrees to plan his breeding program or to select breeding or show stock. He will also give a pedigree to a buyer when he sells a puppy or adult stock. Breeders also exchange pedigrees with other breeders in the search for the perfect stud to complement their breeding program.

Next are the pedigree services. This is a cottage industry that specializes in researching the stud books to produce a pedigree for breeders and dog owners who want the pedigree of a particular individual but do not have access to the breeding records themselves. These services usually charge by the number of generations ordered. Many of these pedigrees are typed onto preprinted forms that are attractive enough for framing. Indeed, some of the services even offer parchment.

Finally, there are the national breed clubs. Each breed of purebred dogs recognized by the American Kennel Club has a parent club which is responsible for the maintenance of the breed standard and disseminating information about the breed to its fanciers and to the public seeking knowledge. The majority of the breed clubs publish some type of yearbook recognizing the accomplishments of the dogs in their breed during the preceding year. As a minimum, the accomplishments are the earning of one or more of the titles awarded by the American Kennel Club—bench championship, field championship, and obedience titles. These yearbooks or annuals usually contain a photograph of the dog, its name and vital statistics, and a three-generation pedigree.

While all three of these groups are involved in the production of pedigrees and all three could (and do) use TEX to obtain beautiful pedigrees, this paper is going to discuss the production of the yearbooks. It shows how the preparation of a TEX document can be simplified and automated. It also shows how halftones can be included in the document.

The yearbooks range in quality from simple typewritten manuscripts to fully typeset pages. Some contain only a few pages each year, while others will be several hundred pages thick. Historically, the production cycle has been something like this:

- Each month, the American Kennel Club sends to the secretary of the parent club a copy of the title acknowledgement it has sent to the owner of the dog.
- The secretary collects these slips and forwards them to the yearbook editor when that person has been appointed by the club's board of directors.
- The editor sends to each owner an invitation to participate in the yearbook. This invitation usually specifies what will be published, the cost (if any) to the owner, a form for submitting the required information, and a statement of the deadline for receipt of the form and photograph if one is to be included.
- As the forms are returned, the editor types the information in the standard format used by the club. As noted above, sometimes this becomes the camera-ready copy while other times it is sent to a typesetter.
- When all the forms are in (or the deadline has passed) and the editor has put the information into the proper format, he takes the forms and photographs to the printer who prints and binds the yearbook.
- Finally, the editor sents out the yearbooks to all who are to receive them and returns the photographs to the owners. The cycle is ready to start over.

The editor must type each and every pedigree. That, in the case of a three-generation pedigree, is fifteen names. He must get each name in the correct position on the form—both for accuracy and appearance. Hopefully he will get the spelling correct as well. There is a significant repetition of names in a large number of pedigrees over a short period of time. It is not uncommon for breeders

to flock to a particular stud, particularly if it was a big winner in the show world. And, of course, all his ancestors will be the same.

Computer programs have been available to help the breeder produce his pedigrees for five or six years. Most of these programs create and store individual pedigrees on diskette. In 1987, we introduced a new pedigree program which we named "Peder." It stores the information about a dog and its breeding in a database. When the breeder wants a pedigree, Peder searches the database and assembles the pedigree at the time of printing. This gives the user the flexibility to specify the number of generations to be printed and the ability to correct a dog's name or add titles to its name in one place and have the corrections automatically reflected in all the pedigrees in which the dog appeared.

Within two months of when we started shipping Peder, I received a call from the Puli Club of America for help with their yearbook. They sent us a copy of a few pages from their last yearbook as a sample of what they were producing. Among other things, they were having the pedigrees typeset and they wanted to continue with that quality appearance. They also wanted to maintain their present format, which was not fully typical of a pedigree.

The editor of the Puli yearbook was, in many respects, typical of the user we can expect. She wanted to use the computer to get the job done and did not have a lot of time to learn to use complex programs. She was atypical in that she was willing to be my guinea pig and put up with more than I could ask of most customers.

After surveying the electronic publishing systems available for the MS-DOS platform, I decided that TeX was the best-suited product for what I wanted to achieve. Products like Ventura and PageMaker require additional software (Digital Research GEM or Microsoft Windows) and do not lend themselves to being controlled by another program. TeX can be started—and completely controlled—from within another program. That program can create the input for TeX, cause TeX to produce the `dvi` file and finally send the resulting output to the printer. I realized that I was going to have to make TeX as transparent as possible for the novice user. The sheer volume of available commands would scare away potential users. TeX can be intimidating to the neophyte. I was also intrigued with the possibility of including the photograph as a part of a TeX document to eliminate the need to have halftone plates made and the photographs combined with the pedigree in the print shop. This step by the printer has always been a source of error. Very few yearbooks do not have at least one misplaced photograph. A printer charges $25.00 or more to screen a photograph and prepare a metal plate.

Integrating Peder and TeX

The person who would be using TeX to produce a yearbook is unlikely to be a computer wizard, let alone a TeXpert. He or she would rather be on a circuit,

showing dogs. Therefore, I had to make preparing the TeX file as easy as possible. Ideally, the editor should never have to touch the input file for TeX—at least for the production of the pedigrees. This was relatively easy. Since Peder is a menu-driven program, I added four lines to the menu (Fig. 1).

```
1.06
                    DOG-GONE COMPUTERS
                PEDER - THE PEDIGREE SCRIBE
                      May 24, 1988

                (1) Add/Edit dog profile
                (2) Print a pedigree
                (3) Initialize PEDER.TeX
                (4) Add a pedigree to PEDER.TeX
                (5) Run TeX with PEDER.TeX
                (6) Print PEDER.DVI (Epson printer)
                (7) Print table of contents
                (8) Print list of descendents
                (0) Quit this session

                What is your desire? _




           Copyright (c) 1987 by Micro Programs Inc.
```

Fig. 1. Menu for PEDER showing modifications for TeX

The first of the new entries (3) initializes the TeX file. It prompts the user for the title year. The last two digits of the year became part of the page number. It then creates a TeX file with the page numbering, vertical and horizontal sizes, vertical and horizontal offsets, and the terminating `\bye`. While not done for the first of the Puli Club yearbooks, this prolog could also set the title page. This task is selected by the user to start a new volume at the beginning of the year. The initialization information is distributed in a file `PEDINIT.TEX`. Peder creates a file `PEDER.TEX` (erasing any existing file with the same name) and copies `PEDINIT.TEX` into `PEDER.TEX`. Because I put the initialization into a separate file rather than burying it in the code for Peder, I can easily customize it for each club. Also, as the editor becomes acquainted with TeX, she can modify it to change the design of the yearbook or to add title pages, and other "fixed" information. Fig. 2 shows the contents of `PEDER.TEX` after the user has initialized it with this step, but before he has added any pedigrees.

The second line added to the menu enables the user to add a pedigree to the TeX file (4). For this, I created a style sheet (Fig. 3) that Peder could scan. I defined the commercial at-sign ("@") as Peder's substitution character. Whenever it is encountered in the style sheet, the program uses the letter and number immediately following to define what information from the database is to be filled into the style sheet. The letter "D" was defined to mean one of the

```
\hoffset=-.25in
\hsize=4in
\voffset=-.375in
\vsize=7.25in
\parindent=.5in
\footline=\hss \tenrm @y--\folio
\bye
```

Fig. 2. PEDER.TEX initialization

fields from the dog's *vita*—and is numbered in the same order as it appears in the profile used to add dog names to the database (e.g., D1 = dog's registered name). The letter "N" was defined to mean the name of one of the dogs in the pedigree and was numbered from the top to bottom of the pedigree. The letter "R" was defined to mean the American Kennel Club registration number of the dog. The sequence of numbers was the same as the one used for the dogs' names.

```
\topinsert \vskip 3.562in \endinsert \vss
\centerline{\bf @d1}
\vskip 6pt
\centerline {@d4\hss @d2\hss @d6}
\centerline {Breeder: @d13}
\centerline {Owner: @d14}
\vskip 9pt
{\settabs 3 \columns
    \+&&@n1\cr
    \+&@n2\cr
    \+&&@n3\cr
    \+@n4\cr
    \+\indent (@r4)&&@n5\cr
    \+&@n6\cr
    \+&&@n7\cr
    \vskip .25in
    \+&&@n9\cr
    \+&@n10\cr
    \+&&@n11\cr
    \+@n12\cr
    \+\indent (@r12)&&@n13\cr
    \+&@n14\cr
    \+&&@n15\cr}
\vfill\eject
```

Fig. 3. Style sheet for adding pedigrees to PEDER.TEX

Like the initialization file, the style sheet is distributed as a file on the Peder diskette. Again, that gives us—or the user—the flexibility to change the design of the pedigree without modifying the code for Peder. The style sheet shown in Fig. 3 includes registration numbers only for the sire and dam. It could just as easily include them for all the dogs in the pedigree and the style sheet for the Puli Club does just that. The two pedigrees in this paper illustrate two different style sheets.

The user enters the registration number of the dog whose pedigree is to be added to the TEX file. Peder then extracts the names of the dogs for the pedigree from its database, scans the style sheet, and writes the pedigree in TEX format to `PEDER.TEX`. Each time it encounters an attention character, it fills in the appropriate areas with the information from its database. As pedigrees are added to the file, the `\bye` is moved to the end of the file. As Peder is transferring the text to the TEX file, it watches for any characters like the "&" used by TEX for its purposes and adds the necessary backslash. To keep either Peder or TEX from choking on missing information in the database, Peder inserts the phrase "Not given" if either the call name or date of birth is missing. This is particularly important for the birthdate since it is entered as all digits in the database and expands into the spelled-out month for the yearbook.

Pedigrees can be added to the TEX file one at a time, or in batches, depending upon the way the editor wants to assemble the book. Each new pedigree is appended to the end of the existing file.

When the yearbook is completely assembled into a TEX file, the user can run TEX from within Peder by selecting option (5) (the third of the new entries) from the menu items. The dialog with TEX is shown in a window on the screen. Finally, the user can send his `dvi` file to his output device by selecting option (6). While the menu illustrated in Fig. 1 specifically mentions the Epson driver, I have generalized the selection in later versions of the program. The user tells Peder which driver to use through the DOS set command.

The style sheet used for the initial attempt with the Puli Club did not take full advantage of the power of TEX and of the possibilities of this technique. For example, we did not generate the table of contents automatically. Since we were debugging Peder and Dee Rummel (the yearbook editor) lived in California, I wanted to keep the first try as simple as possible. I produced the table of contents for her after we completed the rest of the volume.

It transpired that many Pulik have more than one title. Sometimes the titles are earned over a period of years and other times they are earned in the same year. When the club publishes its yearbook, it indicates the current title by underlining it. If the dog earns more than one title in a year, the current titles are flagged with an asterisk. Because these markings only exist for the current year, they were not entered into the database. It was not difficult to teach the editor how to add the necessary information to the TEX file before printing it.

The production cycle is now reduced to a few steps. First the editor adds profiles of the dogs to Peder's database as they are published in the stud book. This can be done monthly as the volumes arrive from the American Kennel Club. In April 1988, eighty-one Saint Bernards were admitted to the stud book. The editor may also update the titles earned by the dogs when the confirmation slips arrive from the AKC (ten Saint Bernards in April 1988). The editor initializes the TEX file for a new yearbook from Peder. As the owners return their forms confirming that they want their dogs included in the yearbook, the editor can add the pedigrees to the TEX file. On the other hand, she can wait until all the forms are in and do it as a batch. Dee Rummel has elected to do it this way. In one evening, she can add all the pedigrees to the TEX file, print them, and have them ready to go to the printer. Proofreading the pedigrees takes another one or two evenings.

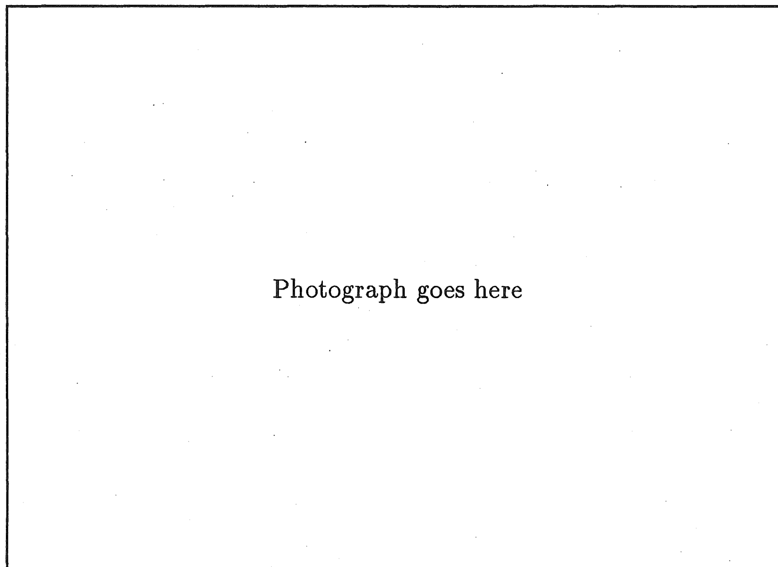
The results of this first volume have been very satisfying. It has reduced the cost of production significantly and the production time to a few days of a part-time volunteer. The initial volume still required the printer to make halftone plates of the photographs of the dogs and combine them with the camera-ready pages of pedigrees. The second volume is due to be printed shortly. One month after completing the 1982 yearbook, Dee Rummel had the copy for the 1983 yearbook ready for the printer (and then moved from California to Wisconsin, injecting a delay into the schedule).

One fringe benefit of adopting TEX for the Puli Club has been the feasibility of correctly spelling the names of many of the dogs. The Puli was admitted to the American Kennel Club stud book in 1936. After World War II, many soldiers brought Pulik home with them. Therefore, it is not uncommon for the name of a dog or its breeder to contain accent marks reflecting their Hungarian origin. Peder does not care how the user enters names, so it is possible to include TEX sequences to add the proper accent marks.

Dee typeset the seventy-one pedigrees in the 1982 yearbook without learning any more TEX than adding accent marks to Hungarian names, underlining the titles, and adding an asterisk to other titles. She is currently learning enough about TEX to typeset the president's message and preface. Since this is straight copy, it is easy to do.

The stylesheet used by the Puli Club is slightly more complex than the sample shown in Fig. 2 since space had to be made for the registration numbers. Specifically, there is a reduced baselineskip between a line containing a registration number and a line containing a dog's name. Also notice that the registration numbers are indented in generations one and three but are flush with the dog's name in generation two. This matches the format the club has used in previous years.

Fig. 4 shows the output from TEX for a pedigree in the format used by the Puli Club.



PEBBLETREE'S HANGOS-HUBA CDX
"Han"

AKC #WD943872—Dog—Whelped Feb. 7, 1978
Breeder: Dee Rummel
Owner: Wilma Peterson & Jan Arnold

Ch Péli-Völgyi Fifi WB868152 *Sire*	Int Ch Gyáli Marci Primás MET7503	Gyáli Módos Kormos MET6832
		Eszkimó Astrea MET6830
	Ivánfaréti Pipacs MET8543	Int Ch Pusztai Fürtös Fickó MET6694
		Fonyódberki Bogáncs MET5592
Szentendreparti Ancsa WB868151 *Dam*	Cac Csabaujtelepi Marci MET7203	Serif Betyár MET4932
		Dgy Csabaujtelepi Csintalan MET4426
	Velencetavi Panni MET6827	Királyaki Uki MET5883
		Bábonyi Pamacs MET6059

Fig. 4. Pedigree from the Puli Club Yearbook

Integrating Halftones into TEX Output

The next logical step will be to add halftones of the dogs to the pedigrees so the printer gets completely camera-ready copy with no paste up required. Since getting the correct picture with its corresponding pedigree is always a problem for the printer and the cost of producing the halftone screens is not insignificant, this step will improve the value of the publication while reducing its production cost significantly by eliminating the cost of screening. The majority of the pictures used in these publications are those taken by professional photographers at dog shows to chronicle a particular win. Most photographs submitted by the owner will be 8x10 in. color prints. The primary point of interest will be the dog. Frequently the photograph will include the handler and the judge. There may even be considerable irrelevent, distracting background. The reader of these yearbooks will be studying the pictures to examine the conformation of the dog, so details like angulation, ear and tail set, depth of brisket, and markings must be discernible from the print. If the editor can easily crop the photograph to concentrate on the dog, the readers will be better served.

With the exception of an article or two in *TUGboat*, I found few literature references to printing halftones as part of a TEX document. The literature from Aldus gave us some direction and an idea of the quality I should be able to achieve, but its emphasis was on incorporating halftones into PageMaker documents. A recent issue of *Colophon* from Adobe contained an excellent, albeit brief, discussion of the issues involved in processing halftones for an encapsulated PostScript file.

I started experimenting with scanning photographs and printing the results. The first problem was to get the scanner files into a format that could be used with TEX. Most scanners (and certainly ours—the Panasonic FX-RS505) save the image as either a `.pcx` (PC Paintbrush) or TIFF (tagged information file format) file. TEX device drivers want a file in the correct format for the output device (e.g., either a HP LaserJet file or a PostScript file). I discovered quickly that while PageMaker was accepting TIFF files, there was little other software that would work with this format. There also appears to be some confusion about the TIFF standard.

The software (PanaScan) that came with the Panasonic FX-RS505 scanner did not support grey scale. Its halftone conversion used dithering (either press or spiral). It did allow the user to save all or part of the scanned image. It had limited capabilities in other areas—no image sizing or image editing. Hammerlab Corporation (New Haven, CT) produces scanner software called Scan-Do. This program will control the scanner and then allow the user to touch-up and crop the image. It provides (for photographs) both dithered and grey-level formats. The grey-level images can only be saved as TIFF files. It requires Microsoft Windows (a runtime version of Windows is included). The size of the image can be controlled at the time the original is scanned. It uses the full resolution of the scanner regardless of the image size, so a reduced image will have more dots per

inch than a full size image. The image editing is quite versatile. The user can adjust the size of the "airbrush," match the gray-shade, and use up to eleven levels of magnification while editing. One missing feature that would be very useful is the capability to rotate the image.

Since PanaScan or Scan-Do produced only `.pcx` or TIFF files, it was necessary to convert the files to PostScript or LaserJet files. The conversion program I looked at was HiJaak (Inset Systems, Inc., Danbury, CT). This program converts files from one format to another for a variety of graphics formats, including PC Paintbrush, TIFF, LaserJet, and PostScript (output only). As of release 1.0B, it does not support grey scale TIFF files, so I was restricted to using PC Paintbrush dithered files for input. The conversion process is easy enough. The user can control the size of the final image as part of the conversion process. One can even convert directly to the printer for making proof copies. However, the user cannot crop or convert only a portion of the image.

The first photograph I scanned was a color photograph of a Saint Bernard. The results were totally unacceptable. Even on the screen, the dog was lost in the background when using dithering. As a side note, when scanned as a halftone with Scan-Do, the image on the screen showed promise. As noted above, I could not convert the TIFF file, so I do not know how it looks when printed.

Next, I tried a black and white photograph of a Dalmatian. The results were passable, but certainly not as pleasing as those obtained by traditional plate making techniques. There did not seem to be any difference in the quality of the printed image if the scaling was done when scanning with Scan-Do or when converting with HiJaak.

Once I had a PostScript file of the scanned photograph, incorporating the file into the pedigree was straightforward. I modified the TEX macro (Fig. 5) distributed by ArborText with their DVILASER/PS PostScript driver. I took advantage of the fact that encapsulated PostScript files contain dimension information. The PostScript standard is 72 divisions per inch which is the same as a big point, so the parameters for the macro are given in big points.

Since I can scale the photographs at the time of conversion using HiJaak, I did not include the scale parameter in the macro. There is an advantage to scaling at the time of conversion: the bounding box will be the same for all the photographs as long as they have the same orientation. Perhaps a TEX wizard can figure out a way to read the encapsulated PostScript file, extract the bounding box, and insert the values into the macro in the TEX file. Then, the program could automatically compensate for any changes.

The prolog was modified to include the macro definition, and the style sheet (Fig. 6) was modified to use the macro to incorporate the PostScript file containing the dog's picture. The picture file is named using the dog's registration number. The production procedure has the added step of scanning and saving the photographs of the dogs. The style sheet has become even more complex,

```
% Macro to use encapsulated PostScript files with TeX.
% Obtain the lower right and upper left coordinates from the
% bounding box.
% Arguments for the macro:
%   #1: name of postscript file
%   #2: lower left x-coordinate
%   #3: lower left y-coordinate
%   #4: upper right x-coordinate
%   #5: upper right y-coordinate
\def\printpicture#1#2#3#4#5{
   \nobreak
   \hbox to \hsize{
      \hss
      \dimen0=#4bp
      \advance\dimen0 by -#2bp
      \hbox to \dimen0{
         \dimen0=#5bp
         \advance\dimen0 by -#3bp
         \vbox to \dimen0{
            \vss
            \special{
               ps::[asis,begin]
               0 SPB
               /picturepoint save def
               /showpage {} def
               Xpos Ypos translate
               #2 neg #3 neg translate
            }
            \special{
               ps: plotfile #1 asis
            }
            \special{
               ps::[asis,end]
               picturepoint restore
               0 SPE
            }
         }
         \hss
      }
   }
}
```

Fig. 5. Macro for adding photographs to pedigrees

incorporating rules as in a pre-printed form. The resulting pedigree looks like the one in Fig. 7.

Incorporating photographs into documents via electronic publishing is still in its infancy. When talking to vendors and software developers, their primary interest is in business graphics—clip art, graphs, logos, and similiar mate-

rial. As the scanners and their supporting software become better at handling photographs—particularly color photographs—it should be possible to produce camera-ready copy with halftones of quality equal to that of the text obtained by typesetting with TeX. This first attempt shows a lot of promise.

Conclusion

TeX can be used to produce kennel club yearbooks and other pedigrees of a quality to satisfy all critics. By marrying it to Peder, we have been able to make it possible for a person who never read *The TeXbook* to produce a yearbook. As scanners and scanner software improve, we are going to be able to include high quality halftones of the dog in the pedigree, eliminating the screening and paste-up at the printers. To date, I have restricted the graphics work to a PostScript driver. I want to extend this to the HP LaserJet as soon as I can.

This application illustrates that it is possible for a person who has not received any training in TeX to extract information from a database and produce a TeX document. The repetitive production of tabular material is an ideal application for this technique. The same approach can be used when an organization has to maintain control over the design and appearance of the output.

Acknowledgements

I would like to give special thanks to Dee Rummel of the Puli Club of America for all her help on this project. Working the problems out with her by telephone was very trying for her at times—especially when her Tandy 1000 refused to run TeX because of a problem in Tandy's version of MS-DOS. Also, thanks to Janet Ashbey of Sugarfrost Kennels for providing the black and white photographs of the Dalmatian.

Update

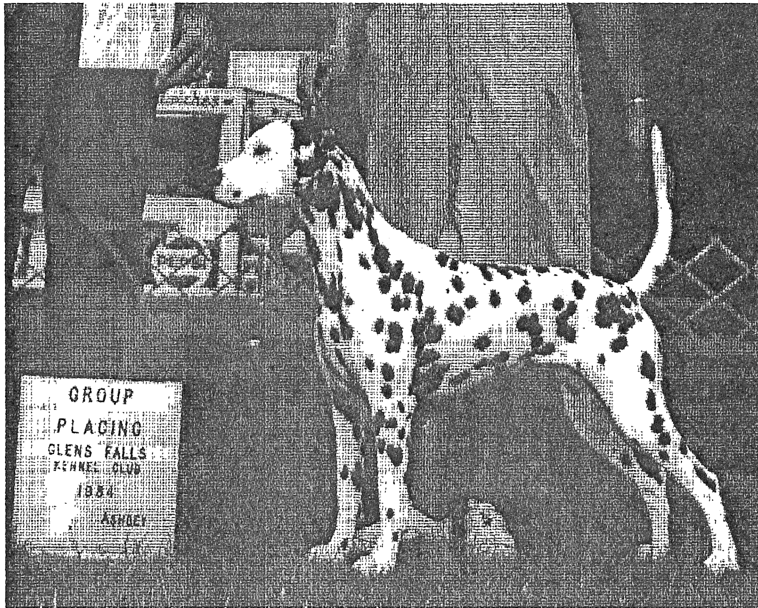
Following the presentation at the annual meeting, there was a lively discussion that focused primarily on obtaining high quality scanned images from photographs. Color scanners and image enhancement programs were mentioned. While these items do exist, they are too costly today for the average firm or person on a PC budget.

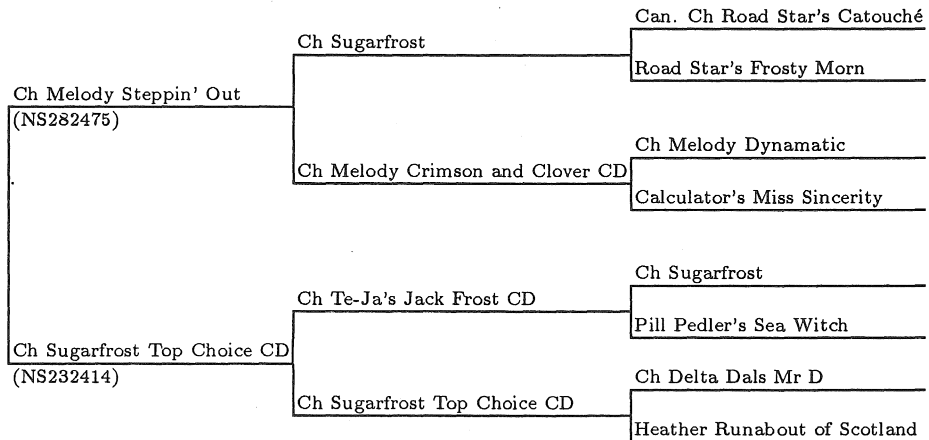
InSet Systems shipped beta copies of the latest version of HiJaak to their testers in September. It is converting more variants of TIFF files and the quality of the PostScript image is much better than those produced from PC Paintbrush files.

```
\centerline{\namefont @n8}
\vskip .25in
\printpicture{@r8.eps}{27}{536}{315}{765}
\vskip .25in
$$\vbox{\tabskip=0pt \offinterlineskip \tenrm
    \halign to \hsize{\tabskip=0pt plus1em#&#\hfil&
            #&#\hfil&#&#\hfil&#&#\tabskip=0pt\cr
        &&&&&\strut @n1&\cr
        &&&&\multispan3\hrulefill\cr
        &&&&\vrule&\cr
        &&&\strut @n2&\vrule\cr
        &&\multispan3\hrulefill\cr
        &&\vrule&&\vrule&\strut @n3&\cr
        &&\vrule&&\multispan3\hrulefill\cr
        &\strut {\tenrm @n4}&\vrule\cr
        \multispan3\hrulefill\cr
        \vrule&\strut (@r4)&\vrule&\cr
        \vrule&&\vrule&&&\strut @n5&\cr
        \vrule&&\vrule&&\multispan3\hrulefill\cr
        \vrule&&\vrule&\strut @n6&\vrule\cr
        \vrule&\vrule&\multispan3\hrulefill\cr
        \vrule&&&&\vrule&\cr
        \vrule&&&&\vrule&\strut @n7&\cr
        \vrule&&&&\multispan3\hrulefill\cr
        \vrule&\strut&\cr
        \vrule&\strut&\cr
        \vrule&&&&&\strut @n9&\cr
        \vrule&&&&\multispan3\hrulefill\cr
        \vrule&&&\strut @n10&\vrule\cr
        \vrule&&\multispan3\hrulefill\cr
        \vrule&&\vrule&&\vrule&\strut @n11&\cr
        \vrule&&\vrule&&\multispan3\hrulefill\cr
        \vrule&\strut {\tenrm @n12}&\vrule\cr
        \multispan3\hrulefill\cr
        &\strut (@r12)&\vrule&&&\strut @n13&\cr
        &&\vrule&&\multispan3\hrulefill\cr
        &&\vrule&\strut @n12&\vrule\cr
        &&\multispan3\hrulefill\cr
        &&&&\vrule&\strut @n15&\cr
        &&&&\multispan3\hrulefill\cr}}$$
\vfill    \obeylines
\noindent Call name: @d3 \hfill Breed: @d5 \hfill Reg. No: @d2
\noindent Date of birth: @d4 \hfill Color: @d7 \hfill Sex: @d6
```

Fig. 6. Style sheet for producing pedigrees with photographs

Ch Sugarfrost High Fashion





Call name: Fashion Breed: Dalmatian Reg. No: NS783683
Date of birth: March 18, 1983 Color: White and black Sex: Female

Fig. 7. Pedigree and photograph using style sheet in Fig. 6

Layout for TEX

ELIZABETH BARNHART AND DAVID NESS

TV Guide, National EDP
Radnor, PA 19088

ABSTRACT

A *Layout Language* and a *TEX Environment* have been defined to allow TEX to be used in a simple fashion to handle complex page-layout problems. The *Layout Language* is written in APL. This article describes how this system works.

The Start of the Problem

During the spring of 1987, we started to seriously consider the problem of creating the Feature Article pages seen in the glossy section of *TV Guide*. The page design of this section is quite complex. The format changes from two column to three column—sometimes in the middle of an article—and contains many runaround sections to accommodate the photographs that are included in the article.

With our own `\output` routine, we were able to create and output a sample article. For the most part it contained a lot of `\parshape` commands, interspersed with the corresponding text.

The first problem that we encountered was using the `\parshape`. By design it terminated at the conclusion of a paragraph, whenever a `\par` was envoked. In our first try, we got around this problem by creating our own paragraph macros, allowing us to create one `\parshape` per column of type.

Looking for a Solution

This solution allowed us to output some, but not all of the styles that occur in the Feature Article section. And it had additional problems. The constraint of having to use `\parshapes` and tying them to a particular section of the text promised to make revision and correction cycles a nightmare for the production personnel responsible for typesetting these pages. It is not unusual for paragraphs of text to be moved around in an article, or for pieces of the text to be removed from the middle of a paragraph, as many as half a dozen times per article in the course of its creation. This would mean that the `\parshape` commands would also have to be moved each time a Feature Article went through revision.Through each correction cycle, each column would have to be trial set to determine where the `\parshape` commands should be moved.

The next problem was that of changing the number of columns per page and the column width mid-article. We spent a lot of time trying to create output routines to do this, with little success.

In June of 1987, Barnhart presented this dilemma to the Grand Wizard in a letter that contained copies of a sample article, and the TEX code that created it. This letter explained our situation, and posed the problem of trying to separate the form of a page from the content of the page. Knuth quickly responded with a method of separating the `\parshape` commands from the text of the article, and we were on our way to a more practical, productive system. His solution also had the added side-effect of allowing us to change column widths at will by using one large `\parshape` command to create the output.

(The next section contains information from a letter by Knuth to Elizabeth Barnhart on the handling of large `\parshape`s. It is reprinted with his permission.

Knuth's Solution

Knuth suggested that our problem could be solved by creating one large `\parshape`. He explains:

> One idea is to make use of TEX's `\prevgraf` feature, by which you can continue the `\parshape` it left off in a previous paragraph. Namely, if you say

```
\newcount\linesdone
\def\par{{\endgraf\global\linesdone=\prevgraf}}
\everypar{\prevgraf=\linesdone}
```

> ... It has the advantage that TEX can do each paragraph individually, and people can use things like `\looseness` to make paragraphs a line longer (or shorter, if it's possible).

Knuth goes on to suggest that a "`\pageshape`" definition would be the next piece needed to create the pages. It consists of each column of the 2-column page being put together with 2 pieces (the `\putattop` and `\putatbot`). This allows you to create a hole in the column where photos or artwork can be dropped in. His description of how to define the "page-shaper" was as follows:

> ... With this idea in mind, you can give a `\parshape` for the entire article; you don't have to intermix shapes with paragraphs. The `\parshape` could be computed by a separate program, or by a versatile word processor, and input from a file. (I recommend putting comments in that file, e.g., 'end of left column'.) But you also need something like "`\pageshape`" to complete the layout. For this I would suggest an approach along the following lines:

```
\newread\colshape \newbox\putattop \newbox\putatbot
\openin\colshape=jobname.shape
\def\nextcol{\read\colshape to \next
    \expandafter\donextcol\next\\}
\def\donextcol#1,#2\\{\topofcol{#1}\botofcol{#2}
    \vsize=477pt\advance\vsize by -\ht\putattop
    \advance\vsize by -\ht\putatbot}
\def\topofcol#1{\global\setbox\putattop=\vbox{#1}\dp
    \putattop=0pt}
\def\botofcol#1{\global\setbox\putatbot=\vbox{#1}\dp
    \putatbot=0pt}
\def\contarrow{\vskip-9pt\moveright12pc \hbox{\arrow}}
```

The last piece of the process is to create a file that contains the description of what the `\putattop` and `\putatbot` pieces look like for each column of each page. Knuth suggests the following scheme:

> ... Now you prepare an auxiliary file, called "`jobname.shape`" if your main text file is "`jobname.txt`". This file contains lines with two entries each, separated by commas; e.g.,
>
> ```
> \skp{53},
> \skp{47},
> ,\vbox to 31\lu{insert caption text here\vfill}
> ,\contarrow\skp{22}
> ```
>
> and so on. The first entry tells what goes at the top of the column (e.g., '`\skp{47}`' means skip 47 lines); the second tells you what goes at the bottom. The entry can be blank if nothing is to be inserted. (An entry can extend over more than one line if the '{' is on the first line and the '}' comes on the last line.) Then your output routine invokes `\nextcol` before doing a column; this sets up two boxes called `\putattop` and `\putatbot`...

The two pieces created by the above coding-scheme are put together with the following definition that boxes one column of a two-column page. This `\columnbox` macro is called by the output routine to lay out a 2-column page.

> ... The definition of `\columnbox` should be changed to, e.g.,
>
> ```
> \def\columnbox{\vbox{\offinterlineskip\putattop
>     \leftline{\pagebody}\vskip-\prevdepth\prevdepth=0pt
>     \puttatbot}}
> ```
>
> Now the formatting of columns is completely detached from the copy of the article. You don't even have to use `DVIMERGE` for captions! This idea can also be extended to control where rules are placed.

Knuth also made some suggestions on handling the placement of the *continued-arrow* that is set to indicate a turnover page.

Layout Development after Knuth's Input

As we worked through Knuth's solution, we were able to use it, with some modifications, to create some fairly complex Feature Article pages. However, someone still had to type in the original—and usually very long—`\parshape` command. Any modification to the shape of the spaces in the article required editing of the `\parshape` command, a complex and error-prone task at best. Therefore, the next step was to develop a way to allow the users to talk "in English" about what a page looked like, and then to have an intermediate program that would create the `\parshape`.

The Next Step

Early experimentation with Knuth's suggestion indicated two possible problems. First, `\parshape` commands did not appear to accept *zero length* lines. Given the original intent of this command this came as no surprise. In setting a layout, however, there are gaps in the flow of text to handle captions, take-out quotes and other formatting structures. We decided that this problem could be handled by creating a macro that would specify how many lines to *take* from the `\parshape` and then how many lines to *skip*. The `\parshape` then would consist of all of the *non-zero length* lines.

The second difficulty resulted from implementation rather than from the concept of TEX. In typesetting a long article we create `\parshapes` of many hundereds of lines.Sometimes our PC-based system is not up to the task. At the moment we seem to be generating obscure errors when we have 'giant' `\parshapes`. It remains unclear to us whether this problem would persist in larger model implementations.

A Layout Language

This led to the creation of a preliminary version of a layout language and a program to generate the TEX commands that are necessary to implement the idea.

We have been using an old *TV Guide* article as a test case. It may be useful to refer to the Figures that use a simulated "ruled" copy of the published version. Here is the Layout file that describes the spaces in the article. The commands are in upper case letters with numeral arguments; comments are shown in mixed case, following a semi-colon.

```
BEGINLAYOUT; Rerun Article

COLUMNLENGTH 53
INDENT 0
COLUMNWIDTH 12.5
LEADING 9
HSIZE 27

'ARROW' SET 0.75
```

BEGINLAYOUT, COLUMNLENGTH, INDENT, COLUMNWIDTH, LEADING and HSIZE must be established at the beginning of each Layout file. Think of these lines as the 'header'. BEGINLAYOUT starts the process. We then specify the number of lines in a normal column and the 'normal' INDENT. These can be changed at any point in time, and remain in effect until changed again. The COLUMNWIDTH here is established at 12.5 picas. We assume pica measurements throughout, except in the LEADING command where leading is established in points. The HSIZE command sets the full horizontal size of the page. The SET command is used to introduce a constant that is of use to us later. In this case it is the width of the 'continuation arrow' character, for which we must leave space on some lines.

Consider the simple page-layout shown in **Figure 1**. Column one is empty, and column two only has 6 lines of text at the bottom. Most of it is a picture and some 'display type' that is not expected to be set in this pass. Now look below at the layout language to describe this page.

```
BEGINPAGE;  Page 1
  BEGINCOLUMN;     Left
  ENDCOLUMN
  BEGINCOLUMN;     Right
    SKIP 47
    PUT 1 AT 8.7
    FILL
  ENDCOLUMN
ENDPAGE
```

The physical elements (pages and columns) have been nested by indenting for readability, but this is not necessary. It is, however, our standard procedure since it produces files that are easy to read and understand. The layout begins with an empty column (no command is given between BEGINCOLUMN and ENDCOLUMN). The right column is 47 blank lines followed by a short line and then five full lines (i.e., 12.5 pica lines as specified by COLUMNWIDTH). We 'knew' how many lines to fill because we keep track of how many lines of the column have been output. Saying FILL is equivalent to saying PUT REST AT FULL where REST is the number of lines that remain in the column and FULL is the full width of the column.

Page two, as shown in **Figure 2**, is **very** complicated. There is text in both columns, in an odd shape, and the bottom of each column is left empty to cut in photographs. There is a slanted picture around which text is to be set. There

is also a picture at the bottom of the page. The description for the page-layout in **Figure 2** is shown below.

```
BEGINPAGE;  Page 2
  BEGINCOLUMN;      Left
    PUT 19 SLOPE HANG LEFT 4.9 TO 7.4
    PUT 5 AT FULL
  ENDCOLUMN
  BEGINCOLUMN;      Right
    PUT 17 SLOPE HANG RIGHT 7.9 TO 5.6
    PUT 2 SLOPE HANG RIGHT 8 TO 11
    PUT 11 AT FULL
    PUT 1 AT FULL-ARROW
  ENDCOLUMN
ENDPAGE
```

The first `SLOPE` operation specifies an area 19 lines long, where the first line is 4.9 picas long and the last is 7.4 picas. It is hung off the left margin. The lines in between will be of the appropriate interpolated length. After this slope there are five full lines and then the rest of the column is blank.

The right column is quite similar except that there are two cut lines, one for the side of the picture and one for the bottom.The `SLOPE` operation specifies starting with a line that comes in 7.9 picas from the `FULL` column width and then in 17 steps we shorten down to a line that comes in 5.6 picas from the right margin. The slope then changes direction (the bottom of the slanted picture) and expands the `COLUMNWIDTH` to 11 picas in two steps. The rest of the lines are full except for the last line of type, which accommodates the 'continued' arrow. Notice that this width is specified by taking `FULL` and subtracting the width of the `ARROW`. In this way we keep our layout independent of the specific current values of either of these quantities.

```
BEGINPAGE;  Page 3
  BEGINCOLUMN;      Left
    PUT 20 AT 8
    FILL
  ENDCOLUMN
  BEGINCOLUMN;      Right
    SKIP 20
    FILL
  ENDCOLUMN
ENDPAGE
```

Page three is simple. The layout language is shown above. Most of the page, as shown by **Figure 3**, is 2-column text. There is a square cutout that fills part of column one at the top and all of column two at the top. A picture two-thirds of the page width is on the top of this page, so we have some narrow text followed by some full text in the left column and some blank space followed by full text in the right column.

```
BEGINPAGE;  Page 4
  BEGINCOLUMN;     Left
    PUT 26 AT FULL
  ENDCOLUMN
  BEGINCOLUMN;     Right
    PUT 26 AT FULL
    INDENT 4.5
    PUT 26 AT FULL
    PUT 1 AT FULL-ARROW
    INDENT 0
  ENDCOLUMN
ENDPAGE
```

Page four is very similar to page three. The above code describes the layout for the page illustrated in **Figure 4**. In this case we want narrow text at the bottom of column two to allow for artwork.This is accomplished by setting `INDENT` to 4.5 picas and then specifying 26 lines at `FULL` (which is automatically adjusted by the amount of the indent). The small cut at the bottom of the right column for the 'arrow' to point the reader to the next page is accomplished by subtracting `ARROW` from `FULL`. Notice that `INDENT` is reset to 0 by the user, otherwise it would continue, even into the next page.

```
BEGINPAGE;  Page 5 --- Shift to 8 pica Columns
COLUMNWIDTH 8
  BEGINCOLUMN;    Left
  ENDCOLUMN
  BEGINCOLUMN;    Middle
    FILL
  ENDCOLUMN
  BEGINCOLUMN;    Right
  ENDCOLUMN
ENDPAGE

BEGINPAGE;  Page 6
  BEGINCOLUMN;    Left
    FILL
  ENDCOLUMN
  BEGINCOLUMN;    Middle
  ENDCOLUMN
  BEGINCOLUMN;    Right
    FILL
  ENDCOLUMN
ENDPAGE

ENDLAYOUT;Rerun Article
```

For page 5, we switched to another page layout. This is shown by **Figure 5**. Here we purposely set some text differently from that in the original article. Like climbing Mt. Everest, this was just to prove that *it could be done.* What we have done is shifted to three 8-pica columns from two 12.5-pica columns. On page five we only set text in the middle column and on page 6 (**Figure 6**) we set text in both the outside columns.

Notice that throughout the examples we have made use of comments. Any line can be commented by appending a semicolon and any desired text. The text will be displayed on the screen at the time the layout is being processed. We have found it useful to indicate progress in the layout activity by appending a comment to each `BEGIN...` operation, but this is only a local convention.

The Layout Language

`LEADING n`

This defines the leading of the material to be `n` points. Leading is the distance between baselines. It is assumed to be in *points* not *picas*. It is the only measure assumed to be in points, and is not adjusted by any `SCALE` command.

`'NAME' SET n`

This defines a `NAME` that we want to use as a generalized number and it sets its value to `n`. Often this is the amount of an indent or the width of a character (like the 'continued arrow'). Once `SET`, `NAME` can pretty much be used as a number (within the rules of `APL` of course).

`HSIZE m`

This defines `m` to be the width, in picas, of the whole pages of the material. Columns are collected into pages of this width.

`COLUMNLENGTH n`

This command specifies that each column contains `n` lines. It stays in effect until changed.

`INDENT p`

This command sets the `INDAMT` (see below) equal to `p` picas.All `PUT` commands are assumed to be indented automatically by this amount. `p` can be specified as either `LEFT p` or `RIGHT p` depending on whether the text is `HANG LEFT` or `HANG RIGHT`. `LEFT` is assumed if neither is mentioned. An `INDENT RIGHT p` will have no effect when text is `HANG LEFT` and *vice-versa*. Only one `INDENT` can be in effect at a time.

`COLUMNWIDTH p`

This command sets the width of a column to be `p` picas.

`BEGINCOLUMN`

This command initializes the `REST` variable to `COLUMNLENGTH`, and counts the column on behalf of the `ENDPAGE` operation.

PUT n1 n2 ...

PUT x AT n

This command works with numbers that represent line lengths. It generates as many lines of **\parshape** description as there are elements to its right at the line lengths in picas specified by **n1**, **n2**, ... For example, **PUT 8 9 10** will "put" 1 line at 8 picas, 1 at 9 picas and 1 at 10 picas. In general **AT** or **SLOPE** commands are used to generate elements, but lists of elements can also be individual numbers. For example, **PUT 10 AT 12.5** will put 10 lines at 12.5 picas.

ENDPAGE

This command indicates the end of a page. The system uses **ENDPAGE** to Figure out how many columns there are on any page.

n SLOPE t b

This command describes a sequence of **n** lines, the first of which is **t** (top) picas long and the last of which is **b** (bottom) picas long. The intermediate lines are of the appropriately interpolated lengths.

SCALE n

This command can be used to scale 'units' used for data entry into some particular measure. Its function is easy to see by example. Say we are working on graph paper where our columns are 10.5 units (on the graph paper) wide. We want to typeset these columns at 8 picas. We could give the command **SCALE 8 DIVIDE 10.5** to accomplish this objective. This command indicates to **APL** that all of the measures entered for widths should, eventually, be multiplied by $\frac{8}{10.5}$ before being handed to TEX.

SKIP n

This is equivalent to **PUT n AT ZERO**. It skips **n** lines.

ENDLAYOUT

This indicates that the layout has come to an end.

FILL

This is equivalent to **PUT REST AT FULL**. Thus it fills the rest of the column with lines at the current **COLUMNWIDTH** and **INDENT**.

The following can be used with other commands in roles where they make sense:

`ZERO`

This can be used in place of the numeric 0 where it eases reading.

`m AT n`

This behaves as though the number `n` had been given `m` times. It is usually used in a context like `PUT 10 AT 6.5`.

`REST`

This number refers to the 'rest' of the lines in any particular column. The `BEGINCOLUMN` command sets this number to `COLUMNLENGTH`, and then all of the line-generating commands (`PUT`, `SKIP`) deduct from it.

`FULL`

This number refers to the current full width of a column, in picas, as established by the last `COLUMNWIDTH` command minus the current `INDAMT`.

`INDAMT`

This number is the amount of the current indent, measured in picas.

`a DIVIDE b`

`a TIMES b`

Most of the commands can take numbers, variables and/or mathematical operations anywhere in the specification. All that is necessary is that the rules of `APL` be followed. These rules are too complex to give here, but suffice it to say that normal mathematics is allowed if everything is completely parenthesised. Otherwise `APL`'s rules will be followed and this can lead the non-APLers into trouble. The `DIVIDE` operator exists because the `APL` character for divide cannot easily be typed in an `ASCII` file. The actual `+` and `-` characters can be used directly for plus and minus. The `TIMES` operation allows access to multiplication in the same way as the `DIVIDE` does to division.

`HANG n...`

This is a 'noise' word that can be included for readability. `HANG n...` is equivalent to `n...`

`a TO b`

This is also a 'noise' word. `a TO b` is equivalent to `a,b`. If both `a` and `b` are numbers, it is also the same as `a b`.

`LEFT n...`

`RIGHT n...`

These words are used to indicate 'left'-ness and 'right'-ness. For example, `INDENT LEFT 5` indicates a left indent of 5, while `INDENT RIGHT 5` indicates a right indent of 5. In general, `LEFT` is assumed if neither is stated. Thus `INDENT 5` is the same as `INDENT LEFT 5`.

`POINTS`

This represents the quantity $\frac{1}{12}$. Thus if you want to input all of your measurements in points instead of picas, just say `SCALE POINTS` and everything will be all right.

As an example of all of this, the following command makes (good) sense: `PUT (REST-6) AT (FULL-1)` requesting that the rest but 6 of the lines in the current column be filled with text one pica narrower than the full column width.

When strings of numbers are used, as for example in `PUT` commands, then they may, or may not, be separated by commas. However, if a 'variable' like `REST` or `FULL` is used, then it must be separated from numbers by commas.

Gruesome but Insightful Stuff

This section of material should only be read by the courageous. However, if you get it, then you will really see how simple all of this stuff is underneath, and be able to do very nice things with it.

The `PUT` command understands lists of numbers. It does not care how these lists of numbers were generated. `APL` takes care of doing all this stuff for the `AT` and `SLOPE` commands. In our data representation, most numbers refer to indents or line lengths. We use positive numbers to represent indents from the left or lines hung from the left (left justified). We use negative numbers to represent indents from the right or lines hung from the right (right justified). With this idea the *implementation* of many of these facilities becomes very simple. The `LEFT` operator, for example, is simply a 'do-nothing'. It leaves positive things positive. The `RIGHT` operator simply multiplies things by −1. It makes positive things negative.

When we have some wobbly lines with a square left margin, we can just measure them and then say `PUT n1 n2 ...`. We can also say `PUT HANG LEFT n1 n2 ...` to the same effect. When we have a square right margin, things are just as simple. We measure the desired line lengths and then say `PUT HANG RIGHT n1 n2 ...`. The `RIGHT` makes the numbers negative. The `HANG` is just noise. The `PUT` recognizes the negative numbers and hangs them off the right margin, and we have accomplished our objective.

Operating the System

Prepare `filename.FRM` and `filename.TEX`. The `FRM` (for 'FoRM') file contains the layout description in the layout language described above. The `TEX` file contains the copy.

Every time the layout changes, execute the command '`LAYOUT filename`'. This will prepare a new layout format file for TEX.

Every time the text or the layout changes, you then execute a `POUR filename` to prepare a new file for printing. If only the text changes, only the `POUR` is necessary. If the layout changes, both the `LAYOUT` and the `POUR` are necessary.

A nice little added feature can be achieved by running `CHECKFRM filename` to check the content of the `.FRM` file. This, following a very nice suggestion made by Knuth in his original letter, generates a copy of the layout with horizontal rules replacing all text. This allows an easy sight verification of the description of the items in the `.FRM` file (without having to `POUR` the copy). The Figures that accompany this article are examples.

There is also a `DISPLAY` program which displays the structure of each page on the PC within `APL`. This environment has some excellent graphics facilities, and it makes it possible to display anything from single pages up through 70 or more pages of an article. The more pages that are shown, the smaller that each of them is, of course, but the shapes and the spaces are quite recognizable even when the images are small.

Internal Details—For those Interested Only

It does seem, however, that Knuth missed one nice feature for `\parshape`. It would make a lot of sense to be able to give shapes that included some lines that were 'zero length', thus indicating vertical spacing. He apparently didn't do this, and we have had to do some implementing around this problem.

The essence of the solution given here is to separate the task of composing some material into two major parts: developing the galley (properly shaped, of course) of all of the text, and then chopping this galley up into the pages that we wish to set.

Here we use the `APL` system to perform this function. The layout driver is read and processed by `APL`. This system interprets all of the commands of the Layout Language, and constructs the (sometimes very large) `\parshape` command that will cause TEX to typeset the appropriate galley.

`APL` also prepares two lists for TEX, the `fmt` list and the `pageform` list. `fmt` contains expressions of the form `(SKIP,TAKE,COLWIDTH)` representing how many lines should be skipped and then how many lines should be 'taken' from the `\parshape` galley. So that we can set a horizontal strut we also communicate the column width, else TEX wouldn't know how wide to make an otherwise

empty column. Notice that this list will allow us to get any combination of vertical spacing that we need in any column.Notice also that breaks in this list always occur at least every COLUMNLENGTH lines, thus easing TEX's data processing complexity and storage requirements.

APL also builds the pageform list. This list tells TEX how many columns are on the page that it is about to set. TEX then arranges to read the next element from the list as it actually outputs each physical page. This is needed so that the output routine knows how many boxes to put together to make up a physical output page.

The mechanics of the list processes are borrowed directly from the "*Dirty Tricks*" chapter of Knuth's *The TEXbook*.

Future Directions

The SLOPE command handles a specific class of layout problems. More complicated shapes, at the moment, must be described on a line-by-line basis using PUT statements. As soon as we know better how to characterize some of these more complex shapes with generality, we will implement further shape-oriented operations.

Further, the opportunity exists to use some combination of mouse, graphics tablet and scanner technology to capture the original shape information. All of these paths will be investigated.

We need to work out how to handle gutters, and how to describe and synchronize the placement of 'furniture' into the layouts.

Comment

APL was used to ease the development process. The prototype implementation described here was done in less than two days.

APL remains an amazingly good language for prototyping. There are also new 'pocket' implementations—available for about $100—which make the language a candidate for actual field use.

1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15
16	16
17	17
18	18
19	19
20	20
21	21
22	22
23	23
24	24
25	25
26	26
27	27
28	28
29	29
30	30
31	31
32	32
33	33
34	34
35	35
36	36
37	37
38	38
39	39
40	40
41	41
42	42
43	43
44	44
45	45
46	46
47	47
48	~~48~~
49	~~49~~
50	~~50~~
51	~~51~~
52	~~52~~
53	~~53~~

Figure 1

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53

Figure 2

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53

Figure 3

1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15
16	16
17	17
18	18
19	19
20	20
21	21
22	22
23	23
24	24
25	25
26	26
27	27
28	28
29	29
30	30
31	31
32	32
33	33
34	34
35	35
36	36
37	37
38	38
39	39
40	40
41	41
42	42
43	43
44	44
45	45
46	46
47	47
48	48
49	49
50	50
51	51
52	52
53	53

Figure 4

1	~~1~~	1
2	~~2~~	2
3	~~3~~	3
4	~~4~~	4
5	~~5~~	5
6	~~6~~	6
7	~~7~~	7
8	~~8~~	8
9	~~9~~	9
10	~~10~~	10
11	~~11~~	11
12	~~12~~	12
13	~~13~~	13
14	~~14~~	14
15	~~15~~	15
16	~~16~~	16
17	~~17~~	17
18	~~18~~	18
19	~~19~~	19
20	~~20~~	20
21	~~21~~	21
22	~~22~~	22
23	~~23~~	23
24	~~24~~	24
25	~~25~~	25
26	~~26~~	26
27	~~27~~	27
28	~~28~~	28
29	~~29~~	29
30	~~30~~	30
31	~~31~~	31
32	~~32~~	32
33	~~33~~	33
34	~~34~~	34
35	~~35~~	35
36	~~36~~	36
37	~~37~~	37
38	~~38~~	38
39	~~39~~	39
40	~~40~~	40
41	~~41~~	41
42	~~42~~	42
43	~~43~~	43
44	~~44~~	44
45	~~45~~	45
46	~~46~~	46
47	~~47~~	47
48	~~48~~	48
49	~~49~~	49
50	~~50~~	50
51	~~51~~	51
52	~~52~~	52
53	53	53

Figure 5

1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	9	9
10	10	10
11	11	11
12	12	12
13	13	13
14	14	14
15	15	15
16	16	16
17	17	17
18	18	18
19	19	19
20	20	20
21	21	21
22	22	22
23	23	23
24	24	24
25	25	25
26	26	26
27	27	27
28	28	28
29	29	29
30	30	30
31	31	31
32	32	32
33	33	33
34	34	34
35	35	35
36	36	36
37	37	37
38	38	38
39	39	39
40	40	40
41	41	41
42	42	42
43	43	43
44	44	44
45	45	45
46	46	46
47	47	47
48	48	48
49	49	49
50	50	50
51	51	51
52	52	52
53	53	53

Figure 6

Syllabi for TEX and METAFONT Courses

S. Bart Childs et al

Dept. of Computer Science
Texas A & M University
College Station, TX 77843-3112
bart@cssun.tamu.edu

ABSTRACT

The TEX Users Group is sponsoring the development of a set of syllabi for courses related to the TEX and METAFONT systems. These would include goals, prerequisites, course contents, laboratory problems and sample tests, suggestions for instructors, text(s) and references. The present article is meant to stimulate discussion in the TEX community, and its contents are thus considered to be in draft form, open to change and modification. It is hoped that this document will encourage better courses and the publishing of related TEX materials.

The descriptions and suggestions in this paper are the result of work done by two committees, one dealing with TEX issues, the other with METAFONT:

TEX	– Bart Childs, Pierre MacKay, David Ness, and Alan Wittbecker
METAFONT	– Doug Henderson, Neenie Billawala, and Pierre MacKay

The reasons for this activity are based on some problems such as:

1. students without proper background
2. widely differing backgrounds in "non-beginning" courses
3. insufficient and inconsistent classes without such structure
4. students or employers having unreasonable or unrealistic goals for classes

The following thoughts are guiding principles:

- Most TEX users are not programmers. Unless courses are explicitly for technical personnel, it should be assumed the **students are not programmers**.
- The course descriptions are based on the public offering of these courses, usually at a university site. The same course should change significantly when there is a homogeneous student body that all use **one** common system and editor for the production of a specific style of document.
- The committees represent a group of TEX professionals who have taught unwilling freshmen; worked as consultants to commercial, governmental, and educational publishing projects; taught TEX courses; ported TEX; have given

significant personal time to the TEX community; and have a broad understanding of many of the needs of the diverse elements in the TEX community.

- The committees offer the disclaimer that the design of a curriculum is never easy and have the firm understanding that all curricula are compromises. Further, the curriculum is only an initial step and should be revised in the future with changing technology.

The material presented in this article is preliminary in nature. We hope it will engender discussion throughout the TEX community, and an improved document will result from your comments and suggestions.

A brief description of these courses is included along with a schematic description of the proposed courses and their prerequisites; an outline of course contents, indicating the various levels of information to be presented at each level (Beginner/Intermediate/Advanced); and a draft of a TEX questionnaire/self-test is discussed. It should help users determine the appropriate course for them to attend.

Course Descriptions

Prior to commencing the study of TEX, one needs to be able to operate an ordinary editor. This knowledge and some desire should be all that is necessary for the **Beginning** course. These courses are nominally one week in duration. Circumstances may dictate courses being offered in various formats.

At the end of the **Beginning** course the student will understand the basic parameters which allow TEX to produce attractive documents. He or she will feel comfortable taking examples from *The TEXbook* for use, but may not yet be fully at ease modifying these examples.

At the end of the **Intermediate** course students will feel comfortable with modifying examples from *The TEXbook* to suit their purposes. They will also be able to creatively solve typesetting problems using TEX.

At the end of the **Advanced** course the student will actually understand many of the examples from *The TEXbook*. At this stage of knowledge TEX's capability as a "text-oriented programming language" can be exploited. TEX macros are a central part of this course.

1. Beginning TEX

This course provides a practical introduction for those with limited, or no, exposure to TEX, and will be composed of about equal parts lecture and hands-on sessions, including many practical exercises for each object of study.

Participants will be introduced to TEX as a series of typesetting instructions, in the context of the history of typesetting and word processing. TEX is compared with other popular formatting systems such as Microsoft WORD, Ventura Publisher, and Aldus Pagemaker.

TEX concepts to be covered include: simple paragraphs, line spacing, and fonts; special characters and accents; justification and line breaking; and the mathematical concepts of superscripts, subscripts, fractions, and equations.

Each registrant will be given a copy of *The TEXbook* and Samuel's *First Grade TEX*.

Prerequisite: Familiarity with a text editor is essential.

2. Intermediate TEX

This course is comprised of equal parts of lecture and laboratory sessions, including many practical exercises.

It builds upon the foundation laid at the beginning level. Topics to be covered include: more complicated paragraph shapes, paragraphs with labels, hanging indention; more complex interaction between glues and boxes; in math mode: Greek letters and special symbols, delimiters, displayed equations; controlling line and page breaking; simple tables.

Prerequisite: Beginning TEX or equivalent knowledge.

3. Intensive TEX

This course is a combination of the above two courses in a smaller timeframe at a higher level of intensity.

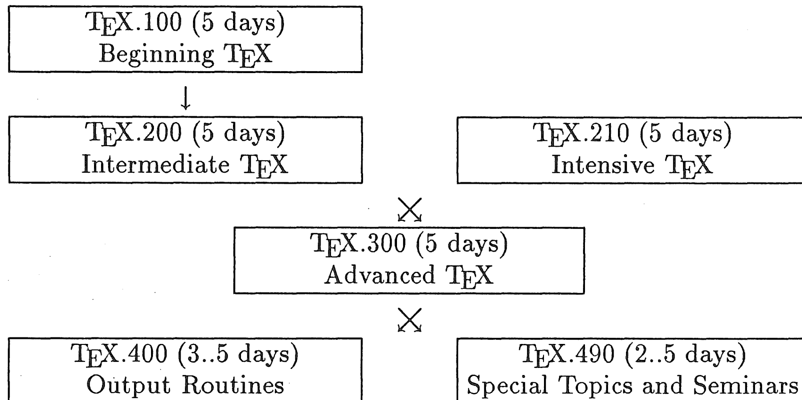
4. Advanced TEX (and Macro Writing)

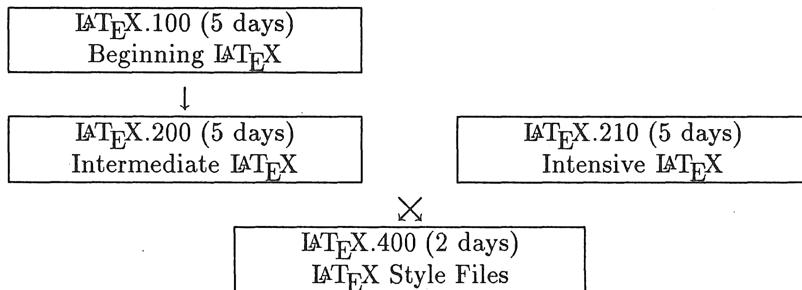
This course is designed for *all* experienced TEX users and will have emphasis on lectures with some opportunity for hands-on experimentation likely. This course is an intensive study of macro writing and designing macro packages.

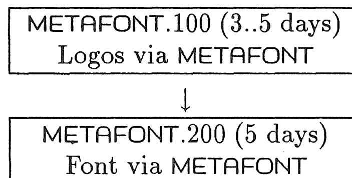
Topics will include: detailed explanation of the relationship of boxes (**vbox**, **vtop** and **hbox**) and glue; use of registers, especially box registers and counter registers; basic concepts of macros; construction of tables using **halign**, also equation arrays in math mode; loading of fonts, magnification, kerning, ligatures; controlling line and page breaking; delimited and undelimited parameters; global vs. local definitions; conditionals, loops, and counters; tools such as **let**, **futurelet**, **chardef**, **catcode**, **##s**, and **begingroup**; expansions of macros and tokens, and when expansion takes place. Students will design macros in class and analyze common constructions, with practice in interpreting already written macros so that they may be customized for special applications.

Prerequisites: Beginning and Intermediate TEX or equivalent knowledge.

5. Course Titles and Prerequisite Structure







The preceding chart shows the relations between the various courses. We would again like to repeat that this structure should be evolutionary. We expect it will change as technology, user requirements, new applications, etc. evolve and change. The *Special Topics* course is in fact set up to deal with specific problems and/or applications related to the programs.

Some possible topics for the *Special Topics* courses are:

– distributions	– porting
– drivers	– graphics inclusion and necessary changes to a driver
– sources	– book production
– sharing	– using color
– extensions	– specialized font topics
– WEB	– graphic systems and METAFONT
– C-versions	– TEX and non-English languages
– tables	– mixing left-to-right and right-to-left languages

Course Contents by Levels

In the following section, the proposed contents of TEX courses are outlined. In the handouts provided at the meeting in Montreal, courses were listed separately, by levels. We present them combined here, both to preserve space, and to indicate the increasing difficulty/complexity presented at each level. Course contents for TEX and METAFONT are included; a similar listing of contents for LATEX courses has not yet been developed.

The levels are indicated by B: for **Beginning**, I: for **Intermediate**, and A: for **Advanced**.

1. Proposed Contents of TEX Courses

- Typesetting
  B: typesetting milieu
  B: design and typesetting dimensions
  B: TEX and WYSIWYG
  I: what you should unlearn (underlines, etc.)
  I: magnification
  A: — open —

- Design
  B: margins, \leftskip, \rightskip, \narrower, typesize, \parindent
  I: penalties and how they affect design; \looseness, \tolerance
  A: database driven design, interface between TeX and other worlds
  A: \pagegoal, \prevgraf
- Programming
  B: public domain, written in PASCAL
  B: how it runs, ASCII keyboard
  B: public domain vs. proprietary versions of TeX
  I: TeX is a programming language
  A: WEB and internal structure
- TeX and Other Things
  B: comparisons: Waterloo Script vs. TeX vs. Pagemaker vs. LaTeX
  I: — open —
  A: — open —
- Markup Languages
  B: they exist
  B: what is plain.tex, vanilla.sty, ...
  B: primitives vs. macros
  I: plain.tex is info source; AMS-TeX, LaTeX: what do they do and why
  A: designing your own
- Spacing
  B: significant/insignificant spaces
  B: tilde, slash, space
  B: \hskip, \vskip, \baselineskip
  I: \vglue, \kern, \hbox, \vbox, \vspace, \vglue, \hspace, \thinspace
  A: letterspacing, sidebarring
- Boxes
  B: only in error messages
  I: moving boxes around: \raise, \lower, \moveright, etc.
  I: What \hbox and \vbox are
  I: \hboxr and \vboxr (from Stephan v. Bechtolsheim)
  A: understanding Stephan's \.boxr macros
- Glue
  B: dimensions
  B: terminology
  I: stretchability / shrinkability
  I: negative glues
  A: output routines
  A: \vsplit, \vadjust, \splittopskip, \splitmagstep
  A: \unhbox, \unvbox
  A: output routines

- Output Routines / Affecting Output
  `B:` no mention: `\hoffset`, `\voffset`, `\footnote`
  `I:` footnotes with numbers
  `A:` significant but discretionary
- Macros
  `B:` macro as shorthand
  `I:` macro with parameters, delimiters
  `A:` `\unskip` commands
  `A:` combos of macros: `\outer`, `\xdef`, `\gdef`
  `A:` all macros, particularly structure and exceptions
- Penalties
  `B:` notice that they exist
  `B:` `\hyphenpenalty`
  `I:` penalties for formatting: `\goodbreak`, etc
  `A:` everything — what they really do
- Rules
  `B:` `\hrules` and `\vrules` *au natural*
  `I:` rules, `\struts`
  `I:` rules for boxes
  `A:` discretionary
- I/O Management and Files
  `B:` comments, documentation etc.
  `I:` use of the `%`-sign to avoid inadvertent spaces
  `A:` writes index, table of contents
- Modes
  `B:` primarily mention math mode
  `I:` horizontal mode vs. vertical mode: what and how
  `I:` math modes
  `I:` restricted modes
  `A:` `\ifvmode`, etc.
- Debugging
  `B:` simple debugging, putting in artificial ends, etc.
  `I:` purposeful errors: `\showthe`, `\showbox`, `\show`
  `A:` tracing and `\showbox`
  `A:` visible boxes
  `A:` `\tracingall`
- Errors
  `B:` flesh wounds, fatal errors, misunderstandings
  `B:` when not to worry about content
  `I:` when errors can be understood
  `A:` real genuine obscurities

- Tabs
  B: `\settabs`, `\tabalign`, `\cr`
  I: — open —
  A: — open —
- Inserts
  B: simple `\topinsert`
  I: `\midinsert`, `\pageinsert`, interaction of several
  A: — open —
- Chars
  B: no mention
  I: `\def\xx{\char...}`
  A: redefine chars
- Graphics and TEX
  B: — none —
  I: space for graphics: `\boxit`
  A: PiCTEX; other things available
  A: Manual (the font), LATEX circle and line fonts; rounded boxes
- PotPourri—Anomalies, etc.
  B: — none —
  I: — discretionary —
  A: TEX and SGML
  A: .DVI and PostScript
  A: graphics/public available
- Aligns
  B: can you copy `\halign` from the book and use it
  I: can you copy an alignment and modify it: `\hidewidth`, `\omits`
  I: `\strut`, `\vrule`
  A: `\valign` and `\eqaligns`
  A: can you create an alignment; alignments and rules
- Tokens
  B: no mention
  I: no mention
  A: explain tokens
- Fonts
  B: what is a font?
  B: what does TEX need to know about fonts?
  B: what fonts are available
  I: what are sources of fonts?
  I: scaling fonts
  I: font measure vs. font 'ink' `.tfm`, `.pk`, and `.pxl` files
  I: limitations on proprietary implementations
  A: understand what METAFONT is
  A: font measure vs. font 'ink' `.tfm`, `.pk`, and `.pxl` files in more detail

- Font Families
  B: no mention
  I: mention
  A: understand and create
- Commands
  B: — open —
  I: — open —
  A: — open —
- Paragraphs
  B: **\parindent**, paragraphs, **\parskip**, blanklines
  I: positive and negative values for parameters: **\narrower**, **\hangindent**, strange shapes
  A: **\parshape**, **\prevgraf**
- Lines
  B: **\centerline**, **\leftline**, **\rightline**, **\line**
  I: line/paragraph interactions and meaning
  A: — open —
- Math
  B: display math as paragraph suspender; in-line math
  I: subscript, superscript (incl. use as footnote numbers)
  I: **\eqalign** and other mathematics typesetting
  A: broken equations
  A: special math spacing — special math fonts
- Control Structures
  B: {} grouping simple existing \ifs
  I: **\begingroup**, **\endgroup**, \if modifying existing \ifs
  I: create **\newcount**, **\newdimen**
  A: understanding new... commands
  A: **\bgroup**, **\egroup**, **\repeat** creation of \ifs
  A: **\futurelet**, **\expandafter**, **\afterassignment**, **\noexpand**
- Syntax
  B: {} surround or follow
  I: spaces that behave unexpectedly
  A: why **\obeylines** works like it does; **\obeyspaces**, **\verbatim**

2. Proposed Contents of METAFONT Courses

- Design size vs. magnification
- Command line processing
  base files: What are they? Why? How (names, etc)? Which? (cm vs. plain)

- Mode choices and default
  - proof vs. production: especially unexpected proof, on screen or on paper
  - `.gf` and `.tfm` files
  - print engines
  - white vs. black
  - necessary adjustments: blacker, fillin, o_correction, `gftoxxx` interpreters
- Mode_defs: how and why
  - experiments with blacker, etc., using `smode` and a font subset of selected characters
- Visual effects
  - optical illusions
  - vertical/horizontal
  - curves
  - diagonals
  - optical corrections
  - letter spacing
- METAFONT as a design language
- Experiments on preset examples
  - coordinates (x,y,z)
  - directions and controls
  - cycle (relation to filling)
  - connections and tensions
  - simple pens (draw your own)
- Review of command line options
  - and summary of needed utilities, including integration with TeX
- METAFONT book examples
  - equations
- Macros (controlling interaction)
  - examples
- Pens and paths
- Calligraphic designs
- Discretion and resolution
  - examples and discussions of limitations
- Proofing utilities and fonts
  - proofing and testing (`fonttest.tex`)
- Transformations and loops
  - borders
- Edges and filling

- Organization of Computer Modern
    - parameter files
    - driver files
    - program files: manipulation of parameters and driver variables
- Individual character programs
    - composites
- Ligatures and kerning
    - `.tfm` font parameters

TEX Questionnaire/Self-Test

In combination with the course descriptions and outlines provided above, the questionnaire/self-test is aimed at potential students of TEX courses, to determine their present level of knowledge, and then to better select an appropriate course. It is hoped that this will help avoid some of the frustration when students register in a course which is either too easy or too difficult, or which presents material either not needed, or not yet understandable, given their current knowledge.

A first draft of the questionnaire has been tested on several classes. It is being extensively rewritten to ensure that it can also be taken without the presence of the instructor. Once completed, this test will be available to the membership at large. It is expected that most students will use the test to ensure correct course choices; it can then be used again, early and late in a course, to gauge the degree of success achieved.

Bibliography

Samuel, Arthur L. *First Grade TEX: A Beginner's TEX Manual*. Stanford: Stanford Dept. of Computer Science. Report No. STAN-CS-83-985. 1983.

TeX Tips for Getting Started

BERKELEY PARKS

Department of Mathematics GN-50
University of Washington
Seattle, Washington 98195
parks@math.washington.edu

ABSTRACT

An introduction to TeX for the novice can be an overwhelming experience; a user-friendly approach is discussed. It is based on plugging the user's own text into a manageable model.

This approach, not generally addressed in the standard TeX sources, provides an overview with tips on how to gain an upper hand on the mass of TeX information available, a manageable finished product model into which text may be plugged, and a comparison of different ways provided by TeX to handle seemingly similar situations.

Stress is placed on considering a page as a potential table, with detailed discussion on a variety of TeX ways to create tables. For the mathematically-inclined, there is a similar comparison of alignment choices provided by math mode.

Using The TeXbook

Our requirements are best met by plain TeX (D. Knuth); other users with different needs might choose LaTeX (L. Lamport), or AMS-TeX (M.S. Spivak). Whichever system you decide upon, reading the manual from Page One is essential. As overwhelming as this may seem at first, eventually you WILL have to do it. There are an increasing number of other resources becoming available that also deserve attention. On-line experience, in combination with whichever pointers below offer you some comfort, should make the experience easier. When you look up an item referenced in the index, read the neighboring paragraph information. Soon you'll find you've read the complete chapter! Be sure to go through the exercises; they contain useful fine points.

Increasing Your TeX Vocabulary

- One good way to learn TeX is to compare line-by-line the printouts of a formatted dvi file and its TeX coding.

- Start a samples notebook. Print a copy of the TEX file as well as the TEX output file. Label the sample with the key command it demonstrates and file in a section that is meaningful. For example, the Math Department looseleaf binder is subdivided into sections for manuscript formats, miscellaneous formats, hboxes, tables, fancy math. You may have to subdivide sections as you accumulate samples.

- Work with your *TEXbook* open on your desk. At first you'll need it as a crutch; heavy TEX users have been known to wear out a book a year!

- Attach paper clips to mark the most frequently used pages in your *TEXbook* for quick access (page numbers refer to sixth printing , 1986):

pg. 52	commonly used accents
pg. 147	slightly larger delimiters
pg. 162	non-italic letters in formulas
pp. 434–438	Greek letters, operators, relations, signs

- Refer regularly to the *TEXbook* index to get comfortable with the variety of commands available. As a beginner you may want to photocopy the index (pp. 457–481, approx. 25 pages) and speed read it each week until you feel familiar with the items included.

- Make lists on 3x5 cards of the commands you miss most often or use regularly but rarely. One category would be the dimension commands—sometimes it's confusing to remember the exact structure required for parameters like `\pageno`, `\vglue`, etc.

- Keep a ready reference list of macros, definitions, and abbreviations that you have created to meet the needs of your specific subject area. Of interest to mathematics-related fields is a MathSci list of symbols and commands available from *Mathematical Reviews* for those instances when you only know the symbol by sight.

Typing Practices for Which to be Thankful

- Preventing errors is something TEXperts do. If your word processor/editor has an on-line spelling checker (as PC-Write 2.7 does), add the TEX control sequences (as well as your own `\defs`). This will safeguard against another common source of errors—typos!

- The most frequent error messages (and sometimes the most devilish to track down) are the infamous `missing { (or })` and `missing $`. When braces or dollar signs are required, it's safest to input both opening and closing signs first and then backspace to enter the text that should appear between them (e.g., in math mode for a fraction, type `\over` first and then fill in the character string).

- Visual clarity in your TeX code is extremely important. Even though TeX reads the file as one extremely long line when it formats, this is difficult for the human mind to follow. You can achieve clarity by staggering sections that are bracketed and by beginning display math mode which is surrounded by **$$** on the next line. You'll be thankful both when you're following up error messages and when you're typing complex tables or math display which nest groups within groups. This indenting creates no problems for TeX because it reads more than one space as one space.

  The complicated formula below is simplified when the source code is staggered. This makes finding portions of the formula that require changes or corrections much easier. Formulas are built moving from left to right, top to bottom.

$$\int^{{x^2+{y^2}\over 2} \over {3y_{10}}} \sqrt{{a+b}\over 2} {{f(x)^{x^2+3x+10}} \over {\prod^{100}_{i=1} {{x^2_i y_i^x} \over 10}}} dx$$

```
$$
  \int^{{x^2+{y^2}\over 2}
        \over
        {3y_{10}}
       }
  {\sqrt {{a+b}\over 2}
  }
  {{f(x)^{x^2+3x+10}}
   \over
   {\prod^{100}_{i=1}
    {{x^2_i y_i^x}
     \over
     10
    }
   }
  }
  dx
$$
```

- TeX code permits you to save keystrokes in some cases. In math mode for example, `\to` is interchangeable with `\rightarrow`.

Creating Macros for Fewer Keystrokes

One specific category of macros that you create yourself is referred to as definitions (`\defs`). Those definitions which apply to many files can be stored in a separate macro file; those that pertain to a specific file are typically put at the top of the current working file. The advantage of including them within the text file is apparent—it means you don't have to keep track of auxiliary files.

Definitions appear within the text in their shortened backslash form and are expanded during the formatting process. They replace complicated strings of characters that are tiresome to type repeatedly or those in which you frequently make typos.[1]

Below is the basic format to create your own macros:

```
\def\insert your-command-name here{{insert the-actual-def. here}}
```

Some macros require only one set of opening and closing braces around the definition, but if you get an error message, try adding the second set.

- Start a file called **mymacs** for definitions you use repeatedly in many files. Type:

  ```
  \input mymacs
  ```

  as the first line in your TEX working file and the program will pull out which `\defs` are needed during the formatting process. When you need to include a `\def` stored in your **mymacs** file as part of your text, type `\your-command-name`.

  As an example, the name 'Lipschitz' may often recur. Create a macro at the top of the file that looks like the following:

  ```
  \def\Lip{Lipschitz}
  ```

  It will be typed as `\Lip` in the body of the file, but each occurrence will print out from the dvi file as 'Lipschitz' on the hardcopy.

Error Messages

Error messages that appear when you TEX your file can seem mystifying. At first you will spend as much time acting on error messages as you do typing the file in the first place. In general, about half the error messages are specific—for example, one might tell you that on line 41 you misspelled a command (**Undefined control sequence.**) or are missing { or } or $; the other half generally may

[1] See Appendix I for a short discussion on abbreviations and defining or mapping keys—two more ways to use fewer keystrokes.

be split between TEX code with which you are as yet unfamiliar, or weird messages that may be: a) too sophisticated for your level of expertise at this time; b) really something totally different from the message (as in `\eqno cannot be used in math mode` which usually means you missed a brace); or c) not real at all because TEX has finally given up the fight to compensate for something you did earlier in the file that it perceives as unreasonable. Later you will find tips within the "garble", but at first, it is less stressful to approach error messages with the attitude that the first and last lines are the important ones. Another thing to keep in mind is that most of your errors as a fledgling TEXer will be typos in the command string or a missing $ or {}.

- As you respond to error messages in the TEX file, you sometimes alter the line count—i.e., correcting line 41 may mean adding or deleting text so that the next referenced error, say on line 87, then will actually be on line 88 or 86! Therefore, it is helpful in some cases (when your errors are from the "easy" category) to correct in reverse order, from the end of the file to the beginning of the file, to preserve accurate line number identity.

Now for the Real Thing

1. Basic Manuscript Formats

Including heading, footnotes, abstract, body of text with new sections, references... (see Appendix II).

2. Perceiving Potential Tables on Every Page

It's time to re-educate your eye so you can find tables where you may not have seen them before becoming a TEX devotee! The most common example of this phenomenon is references (see Appendix II). The reference section of your paper is really a series of columns with the citation number in the far left column and the actual cite in the righthand column (which, incidentally, can wordwrap). Another example of a table-like layout is a curriculum vitae. The best table style to use in both these cases is the `\halign` mentioned below.

Tables are built as horizontal lines from left to right, and then stacking the horizontal lines until all the table data has been typed. The basic elements needed to create a table are the particular "table" command, left and right curly braces to begin and end the table, the pound (`#`) sign which is replaced by the actual text, the ampersand (`&`) sign to indicate column tabs of a specific column, and `\cr` to indicate the end of each horizontal line in the table.

Of the four styles of tables listed below, `\halign` is the most versatile. The `\eqalign` and `\matrix` or `\pmatrix` commands are most frequently used for math displays but are useful to consider for the occasional text table.

- `\settabs` centers the "table" as a whole across page into equal width columns *or* left justifies table as a whole with spacing you specify between columns—limited potential. Note: In the latter case, width of columns is predetermined by choosing the widest entry from each column to represent that column in the preamble line.

- `\halign` left justifies the table as a whole with spacing you specify between columns—unlimited potential. Note: TEX scans all entries for a given column before determining column width.

The preamble (which you should include in your `mymacs` file) is as follows:[2]

```
\halign to \hsize{\hfil#\quad&\vtop
                  {\parindent=0pt\hsize=15truecm
                   \hangindent.0em\strut#\strut
                  }\cr
```

insert your text, indicating for each table line

beginning of new column with `&` and end-of-table line with `\cr`

```
}
```

Example for 2-column table:

Column One	Column 2: This macro creates a two-column table in which the last column will be 10 truecm wide and wordwrapped, aligning the beginning characters in the righthand column.

Add `&#` to the preamble for each additional column you wish to create before the `\vtop` column. If you include more columns, you will have to decrease the width of the last column from 10truecm to reflect this change.

```
\halign to          \hsize{\hfil#\quad&\hfil#\quad
               &\vtop{\parindent=0pt\hsize=8truecm
                      \hangindent.0em\strut#\strut
                     }\cr
```

insert your text, indicating for each table line

beginning of each new column with `&` and end-of-table line with `\cr`

```
}
```

[2] The sample preamble shown above is for a standard 6.5truein page width; 15truecm has been changed to 10truecm in the example below to accommodate the format of this page.

Example for 3-column table:

Column One	Column 2	Column 3: The macro has been modified to create a three-column table in which the last column will be 8truecm wide and wordwrapped, aligning the beginning characters of the last righthand column.

The last column functions like the other columns if there is no text to wordwrap.

- `\eqalign` centers the table as a whole with alignment of character in column following ampersand.
- `\matrix` or `\pmatrix` centers the table as a whole with TEX-determined equal spacing between columns. Note: `\pmatrix` creates left and right parentheses around the table.

To center a table **vertically** on the page, use the following command:

```
\vbox to vsize{\vfil\vbox\vfil}
```

* * *

The examples below demonstrate what different table formats have in common and how they differ from each other. Little or no spacing manipulation was used to illustrate the basic style of each. You need to adjust column spacing and justification using commands like `\quad` and `\hfil`.

- **Tables Using `\settabs` Command**[3]

To divide page equally into columns which are left justified, use the following format:

`\settabs # \columns`, where # is replaced by number of columns

starting every line with `\+` and ending each line with `\cr`. For example,

1	2	3
Richard	Jane	Spot
Man	Woman	Dog

TEX source code for `\settab` using number of columns:

```
\settabs 3 \columns
  \+ 1       & 2     & 3    \cr
  \+ Richard & Jane  & Spot \cr
  \+ Man     & Woman & Dog  \cr
```

[3] See *The TEXbook*, p. 231 [6th printing, March 1986].

To center a `\settabs` table horizontally on the page—add an extra column on the left and right sides, leaving them empty when entering data.

For columns with varied maximum width, create a sample line (a preamble) with the longest item from each column to determine the appropriate width. The print will overlap on neighbouring columns if you do not choose the longest string. Because this line is a measuring device, it will not be printed out as part of your text.

1	2	3
Richard	Jane	Spot
Man	Woman	Dog

TEX source code for `\settabs` creating sample line:

```
\settabs \+ Richard & Woman & Spot \cr
         \+ 1       & 2     & 3    \cr
         \+ Richard & Jane  & Spot \cr
         \+ Man     & Woman & Dog  \cr
```

• **Tables Using \halign Command**: Note wraparound option for *last* column.

1	2	3
Richard	Jane	Spot
Man	Woman	Dog

TEX source code for `\halign` with wordwrap:

```
\halign to\hsize{#&#&\vtop{\parindent=0pt \hsize=15truecm
                          \hangindent.0em\strut#\strut}\cr
     1       & 2     & 3    \cr
     Richard & Jane  & Spot \cr
     Man     & Woman & Dog  \cr
                          }
               }
```

• **Tables Using \eqalign Command**: Use `\hboxes` or `{\rm}` for roman letters.

$$\eqalign{\hbox{id no.}:&\ 1,2,3\cr \hbox{name}:&\ \hbox{Richard, Jane, Spot}\cr \hbox{type}:&\ \hbox{Man, Woman, Dog}\cr}$$

TEX source code for \eqalign:

```
$$\eqalign{\hbox{id no.}:\quad & 1, 2, 3                 \cr
           \hbox{name}:\quad & \hbox{Richard, Jane, Spot}\cr
           \hbox{type}:\quad & \hbox{Man, Woman, Dog}    \cr
          }
$$
```

• **Tables Using \matrix/\pmatrix Commands**: Use \hbox or {\rm} for roman letters.

$$\matrix{1&2&3\cr \hbox{Richard}&\hbox{Jane}&\hbox{Spot}\cr \hbox{Man}&\hbox{Woman}&\hbox{Dog}\cr}$$

TEX source code for \matrix:

```
$$\matrix{1&2&3\cr
          \hbox{Richard}&\hbox{Jane}&\hbox{Spot}\cr
          \hbox{Man}&\hbox{Woman}&\hbox{Dog}    \cr
         }
$$
```

$$\pmatrix{1&2&3\cr \hbox{Richard}&\hbox{Jane}&\hbox{Spot}\cr \hbox{Man}&\hbox{Woman}&\hbox{Dog}\cr}$$

TEX source code for \pmatrix:

```
$$\pmatrix{1&2&3\cr
           \hbox{Richard}&\hbox{Jane}&\hbox{Spot}\cr
           \hbox{Man}&\hbox{Woman}&\hbox{Dog}    \cr
          }
$$
```

3. Alignment in Math Display Mode

There are two standard options for aligning a column in math display mode:

1) \eqalign (which uses \eqno or \leqno to center the equation number along the justified margin)

2) \eqalignno or \leqalignno (which uses & to indicate

   that the equation number will be at the justified margin on that line).

In both cases, only one column of characters can be aligned (therefore only two columns are possible); an `&` is used to indicate which character will start the aligned column.

The differences between them include:

	`\eqalignno`		`\eqalign`
a.	more than one line with eq. no. at margin	a.	only one eq. no. centered on margin
b.	`&` eq. no. `\cr` at end of line	b.	`\(1)eqno` at end of `\eqalign`
c.	cannot break onto next page	c.	can break onto next page

The standard pattern for two columns with alignment is:

```
$$\eqalign{characters & characters  \cr
           characters & characters  \cr
           characters & characters  \cr
          }
$$
```

Adding `\eqno` or `\leqno` after the closing right brace of the `\eqalign` centers the equation number along the right- or left-justified margin, respectively.

```
$$\eqalign{characters & characters  \cr
           characters & characters  \cr
           characters & characters  \cr
           }
  \eqno(#)
$$
```

When the equation number must follow a specific line within the equation, `\eqalignno` or `\leqalignno` is used for placement at the right or left margins, respectively. This command is used when several lines are numbered individually but have a column in common aligned.

```
$$\eqalignno{characters & characters & (#)  \cr
             characters & characters & (#)  \cr
             characters & characters & (#)  \cr
            }
$$
```

- **Example 3.31:** Sometimes what looks like standard alignment is in fact aligned at more than one column. Within the same set of paired `$$`s you can use more than one `\eqalign` limited only by the maximum `\hsize` dimension of your page.

$$\begin{aligned} v_i &= y_i/((1+y_i)(1+t^2_{i-1})), & 0\le i \le q\,, \\ 1+t^2_i &= \sum^i_{j=0}(1+y_j), & 0\ge i\ge q\,, \\ u_i &= \sum^i_{j=0}v_j(=t^2j/1+t^2_i)), & 0\le i\le q\,, \\ R_{\Delta_i}^{(i)}(1-u_{i-1},v_i) &= \exp(-\Delta_i(1-u_{i-1})/2)G_i(\Delta_i v_i/2), & 1\le i\le q\,, \\ G_i(z) &= {}_1F_1((p_i+n_i)/2,p_i/2;z), & 1\le i\le q\,, \end{aligned}$$

TEX source code for Example 3.31:

```
\def\astrut{\vrule height 9.0truept depth 5.0truept
                   width 0truept}%
\def\bstrut{\vrule height 19.0truept depth 12.5truept
                   width 0truept}%
\def\cstrut{\vrule height 17.0truept depth 13.5truept
                   width 0truept}%
\def\dstrut{\vrule height 11.0truept depth 4.0truept
                   width 0truept}%
\def\estrut{\vrule height 10.0truept depth 3.0truept
                   width 0truept}%

$$\eqalign{v_i & =y_i/ ((1+y_i)(1+t^2_{i-1})),      \cr
       1+t^2_i & =\sum ^i_{j=0}(1+y_j),             \cr
           u_i & =\sum ^i_{j=0}v_j(=t^2j/1+t^2_i)), \cr
           R_{\Delta _i}^{(i)}(1-u_{i-1},v_i)
               & =\exp (-\Delta _i (1-u_{i-1} ) /2 )
                  G_i(\Delta _iv_i/2),               \cr
        G_i(z) & ={}_1F_1((p_i+n_i)/2,p_i/2;z),     \cr
        }
   \eqalign{\quad & 0\le i \le q\,,\astrut          \cr
                  & 0\ge i\ge q\,,\bstrut           \cr
                  & 0\le i\le q\,,\cstrut           \cr
                  & 1\le i\le q\,,\dstrut           \cr
                  & 1\le i\le q\,,\estrut           \cr
           }
$$
```

- **Example 3.32:** The example below appears to have two columns with individually numbered lines. This is NOT a `\leqalignno`. In fact, there are three columns aligned (first parenthesis, equal sign, and $\le$ or $\ge$ signs), so this

is actually an `\halign` using math `\displaystyle` within the four columns of the table. Notice the spacing differences between lines in this and the example above.

(3.32a)	v_i	$= y_i/((1+y_i)(1+t^2_{i-1})),$	$0 \le i \le q\,,$
(3.32b)	$1+t^2_i$	$=\displaystyle\sum^i_{j=0}(1+y_j),$	$0 \ge i \ge q\,,$
(3.32c)	u_i	$=\displaystyle\sum^i_{j=0}v_j(=t^2j/1+t^2_i)),$	$0 \le i \le q\,,$
(3.32d)	$R_{\Delta_i}^{(i)}(1-u_{i-1},v_i)$	$=\exp(-\Delta_i(1-u_{i-1})/2)G_i(\Delta_iv_i/2),$	$1 \le i \le q\,,$
(3.32e)	$G_i(z)$	$={}_1F_1((p_i+n_i)/2,p_i/2;z),$	$1 \le i \le q\,,$

TeX source code for Example 3.32:

```
\halign to \hsize{\quad#\hfil\quad&\hfil #
                  &#\hfil&\quad #\hfil             \cr
(3.32a) & $\displaystyle{v_i}$ & $=\displaystyle{y_i/
           ((1+y_i)(1+t^2_{i-1})),}$
                                 & $0\le i \le q\,,$ \cr
\noalign{\bigskip}
(3.32b) & $\displaystyle{1+t^2_i}$ & $=\displaystyle{\sum
           ^i_{j=0}(1+y_j),}$
                                 & $0\ge i\ge q\,,$  \cr
\noalign{\bigskip}
(3.32c) & $\displaystyle{u_i}$
        & $=\displaystyle{\sum ^i_{j=0}v_j
           (=t^2j/1+t^2_i)),}$
                                 & $0\le i\le q\,,$  \cr
\noalign{\bigskip}
(3.32d) & $\displaystyle{R_{\Delta _i}^{(i)}
           (1-u_{i-1},v_i)}$
        & $=\displaystyle{\exp
           (-\Delta _i(1-u_{i-1})/2)
           G_i(\Delta _iv_i/2),}$
                                 & $1\le i\le q\,,$  \cr
\noalign{\bigskip}
(3.32e) & $\displaystyle{G_i(z)}$ & $=\displaystyle{
           {}_1F_1((p_i+n_i)/2,p_i/2;z),}$
                                 & $1\le i\le q\,,$  \cr
                 }
```

Examples 3.33–3.35 show some other useful forms adapted to solve common alignment problems:

- **Example 3.33:** In math display mode a formula may require an end-of-proof symbol justified on the right margin. One way to accomplish this is to use `\displaystyle` within the `\line` command.

(3.33) $$gn(\delta) \times N_\delta \to N \qquad \blacksquare$$

```
\line{(3.33)\hfil$\displaystyle{%
                              gn(\delta)\time N_\delta\to N
                              }$
          \hfil$\endprf$
     }
```

- **Example 3.34:** Math display mode creates a fixed space above and below the pairs of double dollar signs and between lines within them. The `\line` and `\displaystyle` commands provide one way to manipulate these spaces while preserving the style.

(3.34) $$gn(\delta) \times N_\delta \to N$$

```
\leftline{(3.34)\hfil$\displaystyle{%
                              gn(\delta)\times N_ \to N
                             }$
        }
```

- **Example 3.35:** There are several ways to move a formula to the left margin; this example uses the `\item` and `\displaystyle` commands.

(3.35) $gn(\delta) \times N_\delta \to N$

```
\item{(3.35)} $\displaystyle{gn(\delta)\times N_\delta \to N}$
```

- **Example 3.36:** Sometimes misused patterns provide useful models. In the example below, the ampersands were inadvertently omitted which right justified the equations, aligning the last character of each line.

$$\phi : \begin{array}{r} gn(\delta) \times N_\delta \to N \\ (X,n) \to \delta^{-1} \exp(X)(\delta n) \exp(-X) \end{array}$$

```
$$
  \phi:
      \eqalign{gn(\delta)\times N_\delta \to N    \cr
              (X,n)\to \delta^{-1} {\rm exp}(X)
              (\delta n) {\rm exp} (-X)          \cr
             }
$$
```

Bibliography

Knuth, D. *The TEXbook*. Reading, Mass.: Addison-Wesley, 1986.

Lamport, Leslie. LATEX. *A Document Preparation System*. Reading, Mass.: Addison-Wesley, 1986.

Spivak, Michael. *The Joy of TEX*. Providence, RI: American Mathematical Society.

Additional Helpful Publications

Math. Sci. Review. Appendix D: TEX Codes and Symbols.

Plonsey, Daniel J. *A Summary of Common TEX Control Sequences*. Dept. of Astronomy, University of California/Berkeley.

Samuel, Arthur L. *First Grade TEX: A Beginner's TEX Manual*. Dept. of Computer Science, Stanford University. Report No. STAN-CS-83-985.

APPENDIX I

Editors: Defining Keys and Creating Abbreviations

There are several ways to shorten complicated character strings. Macros (**\defs**), which function as a part of TeX were discussed earlier. Some editors also provide you with ways to use fewer keystrokes, defining keys and creating abbreviations. The advantage is immediate expansion within the text as if you had typed the full string.

- For the DOS environment, there is key definition software available for programming combinations of keys to spell out frequently used character strings. Within PC-WRITE (Ver. 2.7) for example, the `ed.def` file allows you to redefine keys with impressively long strings.

  For example, many of our displays include fractions. The TeX command to create a fraction in math mode requires many curly braces (the most frequent cause of grief when debugging TeX errors). The editor allows us to define `Cntl-Alt-F` as {{}\over{}} so all we have to do is fill in the blanks. The problem of unmatched curly braces is virtually eliminated.

- In the UNIX environment, *vi* editor abbreviations can be stored in a file separate from your current working file or entered on the colon command line which makes them accessible for that editing session only. On the UNIX machine open a file (for example, `setkey`) and add a line ⟨`ab saltf \over`⟩. Whenever you type ⟨`saltf`⟩ followed by a space in your TeX file, the abbreviation will expand on the screen before your eyes! You can create sophisticated definitions with the cursor returning to the first } leaving you in insert mode. To call up your abbreviations file for the current editing session, at the : prompt in *vi*, type ⟨`:source abbreviation-filename`⟩. Shown below is part of our `setkey` file as an example:

```
ab  saltf  ^V {{ } \over { }} ^V^[2F{a
ab  salti  {\sl }^V^[hi
ab  sm1   ^V%date:  ^V^V%author(s):  ^V^V%title:%
          ^V^V^V \input mymacs.ver0688^V
ab  sm2   ^H^V%\input authormacs.ver0688^V^V
          \magnification=\magstep1^V\baselineskip=%
          18truept^V^V[9-A
ab  saltm  sm1 sm2
```

- If you are in a network environment, combining key definitions and abbreviation files can prove fruitful. It is possible to preserve continuity for specific key combinations, having them spill out the same source code regardless of whether a file is being edited in a DOS environment or via Kermit on a UNIX machine.

  Consult your local guru!

APPENDIX II

Model Manuscript
"Go Ahead, Plug in Your Own Text"

John Doe* †

Department of Mathematics
University of Washington
Seattle, Washington

Abstract. In my paper [1], the proof of the main theorem is incorrect, as F. K. Chen pointed out to me, but a minor change indicated below renders it correct. Actually, the entire proof can be recast in a simpler, more transparent form, which allows one at once to deduce the stronger form of the theorem available in the XYZ case, due to Jane Doe [3]. After correcting the original proof, I will outline the streamlined proof and its consequences.

* Partially supported by NSF grant DMS-880000.
† Partially supported by NSF grant DMS-889999 and a TUG Research Fellowship.

§0. Introduction.

In recent years there has been a major interest in the theorem of Jane Doe:

Theorem. *Let* $f_r(x) = rx(1 - x)$, $0 \leq r \leq 4$ *be a one-parameter family of mapping of the unit interval. There is a positive measure set of those* r *that* f_r *has an absolutely continuous invariant measure (abbreviation: a.c.i.m.).*

The author of this paper uses his earlier ideas from [1] to give another interpretation to Jane Doe's recent findings.

References Using \item Command

[1.] John Doe, His Paper, *Quart. J. Math. Oxford* (2) **36** (1987), 435–450.

[2.] John Doe and John Rainwater, *Some Books Collaborated Upon Together*. Providence, RI: TUG Publishing, 1988.

[3.] Jane Doe, Reaching the Point of Maturational Readiness and Typing TeX Files, *Bull. London Math. Soc.* **17** (1987), 549–553.

References Using \halign Command with Wordwrap in Righthand Column

[D1] John Doe, His Paper, *Quart. J. Math. Oxford* (2) **36** (1987), 435–450.

[JR] John Doe and John Rainwater, *Some Books Collaborated Upon Together*. Providence, RI: TUG Publishing, 1988.

[D2] Jane Doe, Reaching the Point of Maturational Readiness and Learning to TeX Files, *Bull. London Math. Soc.* **17** (1987), 549–553.

```
\magnification=\magstep1
\baselineskip=18truept
\hsize=4.75truein
\vsize=8.25truein
\vglue0.5truept
\pageno=0
\def\natnums{{ {\rm 1} \kern -.13em {\rm N} }}
\centerline{\bf Model Manuscript}
\centerline{\bf ``Go Ahead, Plug in Your Own Text''}
\bigskip
\centerline{John Doe\footnote{*}{Partially
supported by NSF grant DMS 8800000.}\
\footnote{$\dagger$}{Partially supported
by NSF grant DMS-889999 and a TUG Research
Fellowship.}}
\medskip
\centerline{\sl Department of Mathematics}
\centerline{\sl University of Washington}
\centerline{\sl Seattle, Washington}
\bigskip
\midinsert
\narrower\narrower
\noindent{\bf Abstract.}\ \ In my paper [1], the proof
of the main theorem is incorrect, as F.~K.~Chen pointed
out to me, but a minor change indicated below renders
it correct.  Actually, the entire proof can be recast
in a simpler, more transparent form, which allows one
at once to deduce the stronger form of the theorem
available in the XYZ case, due to Jane Doe [3].  After
correcting the original proof, I will outline the
streamlined proof and its consequences.
\endinsert
\vfil\eject
\beginsection \S0.  Introduction.
In recent years there has been a major interest in the
theorem of Jane Doe:
\bigskip
\proclaim Theorem.  Let $f_r (x)=rx(1-x),\ 0\leq r\leq
4$ be a one-parameter family of mapping of the unit
interval.  There is a positive measure set of those
$r$ that $f_r$ has an absolutely continuous invariant
measure (abbreviation:  a.c.i.m.).
```

```
\bigskip
\noindent The author of this paper uses his earlier
ideas from [1] to give another interpretation to Jane
Doe's recent findings.
\bigskip
\noindent {\bf References Using $\backslash$item
Command}
\medskip
\item{[1.]}  John Doe, His Paper, {\sl Quart.~J.~Math.
Oxford} (2) {\bf 36} (1987), 435--450.
\item{[2.]}  John Doe and John Rainwater, {\sl Some
Books Collaborated Upon Together}. Providence, RI:
TUG Publishing, 1988.
\item{[3.]}  Jane Doe, Reaching the Point of Maturational
Readiness and Learning to \TeX\ Files, {\sl Bull.~London
Math.~Soc.~\bf 17} (1987),
549--553.
\bigskip
\noindent{References Using \halign Command with Wordwrap
in Righthand Column}
\medskip
\halign to\hsize{\hfil#\quad
          &\vtop{\parindent=0pt\hsize=10truecm
           \hangindent.0em\strut#\strut}\cr
[D1]&John Doe, His Paper, {\sl Quart.~J.~Math.~Oxford}
(2) {\bf 36} (1987), 435--450.\cr
\noalign{\smallskip}
[JR]&John Doe and John Rainwater, {\sl Some Books
Collaborated Upon Together}.  Providence, RI:  TUG
Publishing, 1988.\cr
\noalign{\smallskip}
[D2]&Jane Doe, Reaching the Point of Maturational
Readiness and Typing \TeX\ Files, {\sl Bull.~London Math.~%
Soc.} {\bf 17} (1987), 549--553.\cr
}
\bye
```

The Art of Teaching TEX for Production

ALAN WITTBECKER

TEX Users Group
P.O. Box 9506
Providence, RI 02940
aew@math.ams.com

ABSTRACT

Few people who need to use TEX for the purpose of typesetting aesthetically pleasing documents are interested in the more mysterious aspects of the program. Fortunately, TEX can be presented as a powerful formatter for typesetting documents—that is, also as a special purpose programming language.

A production approach in teaching emphasizes those automated features of TEX that are its strength, such as interword and interline spacing, hyphenation, alignments, and mathematical formulas. This approach permits immediate use of TEX for production without the confusion of unnecessary technicalities.

Information is presented in a manageable form through a ladder of complexity from primitive instructions to standard markups to custom macro commands. After an introduction to basic instructions for paragraphs, type fonts, and paging, users are introduced to a short model of a technical paper, complete with more complex instructions for headings, tables, glue, boxes, and simple macro commands. The model is simple enough to serve as a template, permitting text to be substituted for model text and the document to be printed.

The title of this paper has several interesting etymological coincidences. *Art*, from the Latin *ars*, parallel to the Greek $\tau\epsilon\chi\nu o$, means the 'application of a skill'. TEX by design has the same root, referring to the art of typesetting. (The word *technology*, in fact, refers to an industrial art or skill.) *Teach* means 'to guide', from the Old English; the process of teaching is education, from the Latin *educare* ('to lead from'). *Production*, from the Latin *producare* ('to lead forward'),[1] is the goal of using TEX, e.g., (from the Latin *exempli gratia* meaning 'for example'), for typesetting text (the word *text*, by the way, is from the Latin *texere* meaning 'to weave'—the Greeks used the metaphor of weaving to describe a meaningful dialogue, which we should start, now).

[1] Thus, they have different perspectives.

TEX is not a sudden discontinuity in the history of typography; it is the most recent development in a long tradition. TEX is one of the most sophisticated ways of moving characters around on a page, but it owes much to the system developed by Gutenberg (circa 1440–1450). And, moveable type was invented at least three times before Gutenberg: at the Palace of Phaistos in Crete before 1500 B.C. (clay); by Pi Sheng in China around A.D. 1034 (clay); and in Korea by 1397 (bronze). Of course, Gutenberg publicized his invention very effectively.

Many conventions that TEX uses also have honorable beginnings. For example, the Egyptians were using two columns by 1500 B.C., Irish monks were using baselines (and other temporary guidelines for the conformity of letter heights) by A.D. 350, minuscule letters (later referred to as 'lower-case' by typesetters who kept them in drawers below the capitals or majuscules) were given official status by Charlemagne in his decree of A.D. 796, le Juene established the point system in 1737 for measuring metal type—TEX uses points for default measures, as well as much of the terminology of metal typography (cf. folio, leading, quad).

The type designed for TEX by Donald Knuth, Computer Modern, was inspired by Monotype Modern No. 8A. It is a vertically regular face with good contrast between horizontals and stressed upright strokes; it is considered lighter and more open than Times. Times New Roman itself is an elegant style designed in 1931. The narrow shape of its letters and tight letter-spacing work well in the narrow columns of newspapers. Styles of type are grouped by periods. Computer Modern, like Times, is a Modern style of type, characterized by the vertical axis of characters and differences in stress (the thickness of strokes). Other Modern styles are Bodoni and Didot (the first truly Modern style, designed in 1784).

Old Style types were influenced by the visual properties of writing with pens, which were held by hand at an angle (a cant), which gave letters a diagonal axis; the flat edge of the pen made thick and thin strokes, particularly on curves; letters also had projecting strokes, called serifs. Caslon, Baskerville, Century Old Style, and Garamond are Old Style types. (Young typesetters used to be told to go with Caslon when in doubt about an appropriate style.) After Transitional styles and Modern, several new styles were developed. In 1815, Vincent Figgins designed a slab-serif style called Egyptian; there are numerous Egyptian styles today. In 1816, William Caslon IV lopped off the serifs on a font (Figgins named it sans serif). Sans serif types, like Optima, Futura, and Helvetica, are increasingly popular today. Lucida, a recent style designed for electronic typesetting, has serif and sans serif faces.

TEX in the Publication Cycle

As tools become more sophisticated, authors are expected to be typists, designers, typesetters, then publishers. But, this does not have to be so. The tools allow formatting and proofing, but TEX is descriptive (although not completely so, it can be made so by macros and styles) and its rules of design are either default values or determined by macro packages. TEX as a tool has many advan-

tages: The author, secretary, editor, typesetter, and publisher all speak the same language (if it is TeX). The author does not have to understand the mechanics (as, for instance, she might with image-setting programs). The style is the same for documents, but can be changed at the command level (or macro level). It allows for precision, as well as expansion.

General Production Needs

What are the needs in a publishing environment? Publishers need to have the program presented as a typesetting program. This entails using typographic elements in the presentation: publisher's terms, such as *quad left* and *orphans*, are the language of production. Publishing is a descriptive language and not a programming language with procedural mysteries. Fortunately, TeX already uses some of that terminology. TeX needs to be presented as a higher level description language for typographic composition. Typographers tend to be insular, like programmers, and have a traditional view of typesetting. Hardware and software are less than half the game in a production environment.

TeX is a typesetting system intended for documents laden with mathematics, but it is also a full-function, general purpose composition program especially good for long projects in standard formats.

TeX is a complex program. Bad thinking and bad planning are magnified by it (and by computing in general). Although production staff could learn TeX through *The TeXbook*, that method is not fast or efficient. Publishers need training with TeX, and the training should be fast and directly applicable. Training is a large, and unavoidable, investment. TeX training must be

1. Relevant
2. Simple
3. Flexible
4. Efficient
5. Consistent
6. High quality

TeX Course for Production

This approach considers the people using TeX in their particular production context (goals, as well as the other systems in place) and is characterized by a top-down perspective, by function, and not so much by the features of TeX. It starts with what users need to produce pages. Thus, it also addresses by omission what to leave for last: How TeX really works, what else can be done with TeX.

1. Comparison with a Worst Course

What is missing from some TeX courses? The intent of the course. An overview. A definition of terms. A logical progression through degrees of difficulty. Real applications. Exercises and working sessions. Explanations. Follow-up. Probably no one course has ever neglected all these things, but some courses have been inadequate.

2. Provisions of a Better Course

A better course addresses the needs of the production environment.

1. Relevance: What is needed? How suitable is it? How suitable is TeX? What commands are necessary to produce the pages? How suitable to the needs of the users are the commands? How often are the pages produced? How many people are there in the cycle of production? How many of them know TeX?

2. Simplicity: How easy is TeX to understand? How easy is it to use? How easy is it for users to use the commands? Can they start using them immediately? Can they understand them?

3. Flexibility: How easy is it to make changes? How easy is it to alter the format? Can the users make changes? Can they use their instruction as a basis to learn more?

4. Efficiency: Can you work quickly? Can text be entered quickly? Can text be changed with minimal effort? (Keystrokes versus descriptive markup).

5. Consistency: Is it the same? Are macros compatible? Are macro packages compatible?

6. Quality: Can standards be maintained? Are style guides available? Is it portable to other systems?

3. Execution of a Course

TeX courses, now, are divided according to logical levels. The **Beginning** course is production oriented. TeX is placed in the typesetting environment, historically and functionally. TeX defaults are related to design principles.[2] TeX is related to word processors, page composition and image-setting programs, and typesetters.

TeX is treated as a set of typesetting instructions: Only primitive and plain commands are introduced and used to produce various publications exercises. TeX is also presented as a paragraph formatter. Users are shown how to alter the shapes of paragraphs, first, then, how to break paragraphs into lines, as well as into alignments and mathematical phrases, and finally, how to compose

---

[2] See Appendix A.

pages. The special characteristics of TEX—hyphenation, spacing, justification, kerning, and fonts—are woven into the progression. Errors are discussed at each level. Commands become more complex, as dimensions and parameters are added. Macro commands (as definitions) are introduced in appropriate places, at line breaks and hanging paragraphs, for instance. Users are encouraged to copy instructions and use them to produce real examples. Good habits are encouraged, from file management to structured input and comments in text and in definitions.

At the **Intermediate** level, TEX is presented as more than just typesetting instructions; it is presented as a programming language. Users break down pages, paragraphs, and lines into boxes and glues. They discover that boxes can be manipulated in ways that lines and paragraphs can not. TEX's spacing mechanisms are played with as glue, with stretching and shrinking abilities. The modes of TEX are revealed, which explains how many error messages were caused. Alignments and mathematics are reconsidered at a more advanced level. The anatomy of macros is discussed. The various kinds of macros are dissected and used. Conditionals are introduced.

Advanced topics begin with a discussion of how TEX works on a source file. Tokens are introduced. Macros are combined and nested. Users are introduced to ways to 'undo' boxes and glue. Advanced commands dealing with expansion and futurity are introduced. The role of specials in extending TEX is addressed. Understanding is emphasized above memorization.

Summary

TEX was long the province of programmers. The courses were dominated by people who needed to know how to install TEX and write drivers for devices. As front ends are developed for TEX to be used, programmers will rise again, but the era of pioneer hacking may be over. Now, the field is dominated by editors and writers who want to know different things about it, principally how to use it. These users are oriented towards production. TEX courses must address the needs of one group without slighting the other. Courses that are designed to present different aspects of TEX at different levels hold the most promise of meeting the needs of both groups.

Bibliography

Carter, Rob, Ben Day, and Philip Meggs. *Typographic Design: Form and Communication*. New York: Van Nostrand Reinhold. 1985.

Morison, Stanley, and Kenneth Day. *The Typographic Book, 1450–1935*. Chicago: The University of Chicago Press. 1964.

Rice, Stanley. *Book Design: Text Format Models*. New York: R.R. Bowker Co. 1978.

Skillin, Marjorie, and Robert Gay. *Words Into Type*. New York: Apleton-Century-Crofts. 1964.

The Chicago Manual of Style. 13th ed. Chicago: The University of Chicago Press. 1982.

White, Jan V. *Editing By Design.* 2nd ed. New York: R.R. Bowker Co. 1982.

Appendix A. General Principles of Design

- Don't use too many typefaces.
- Within one typeface, use size and weight to distinguish heading levels.
- Be conservative with graphic devices, such as rules, boxes, borders, bullets, and dashes.
- Avoid combinations that are hard to read, such as all caps, long sections in bold or italic, or lines that are set too long or set too close together for the type size.
- Follow traditional standards for differentiating the elements of a document (heads, body, labels, captions, figures, tables, running headers, and specials).

Choosing Between TEX and LATEX

SHAWN FARRELL

McGill University Computing Centre
805 Sherbrooke St. W.
Montreal, Quebec
H3A 2K6
ccsf@musica.mcgill.ca

ABSTRACT

The question of which TEX macro package to use for our typesetting needs has been asked many times since the introduction of LATEX. It is a question that cannot be answered without looking at the many factors that can be involved in the production of typeset material: who the user is, what their needs are, what type of environment they work in. These are just some of the aspects that must be taken into consideration. In fact, there is no clear cut choice, and no matter what one person says, another will say differently. However, by taking a closer look at what is involved, and by identifying the advantages and disadvantages of both TEX and LATEX, we *can* make that choice easier for those who wish to pose The Question.

Behind the Scenes

You might be wondering what prompted me to compose such a paper.[1] Then again, you might not really care, in which case you could just skip over to the next section (or the next paper!). Anyway, when I was busy putting together the events for the Annual Meeting, I felt that it would be a good idea to have that extra McGill "flavour" by having someone from the university present a paper. I knew that there were some people on campus doing some interesting things with TEX, so I figured I could count on one of them. At the same time, being on the program committee, I saw that we had no papers that had anything to do with LATEX, so I set out to find a "McGillite" to do a paper on LATEX—which is why you're reading this right now.

I know that this question of choosing the "right" package has been brought up many times, and has caused some heated discussions, but I think[2] that I bring a different perspective to the subject. In most of the previous cases where such a discussion has occurred, we usually see an experienced TEX user taking

[1] In fact, *I* would hesitate to call it a paper—nothing concrete, just a bunch of opinions.
[2] Perhaps *hope* is a better word!

on an experienced LaTeX user, where the proverbial immovable object meets the irresistible force and, astonishingly, nothing is resolved. The TeX user will tell all his/her people to use TeX because it's better and, of course, Mr./Mrs. LaTeX will say the same to their crowd. This is a very unfortunate situation that occurs with a wide variety of products all the time throughout the computer world.

However, I would not classify myself as a TeX *user*, since my involvement with TeX is one of support and training. That is, I personally have no reason to use TeX, since most of the stuff that I write could be done with any old word processor, but since I have to support intensive TeX users, I need to have a good working knowledge of the system, and so I do use TeX for most of the things I do. Since this is what I do for a living, I realize the importance of recognizing a wide spectrum of users, who have a wide range of needs to be met. This is something that needs to be addressed not only with TeX, but with every piece of computer software, and it is difficult to train users to see things from this point of view.

Finally, before I take a more in-depth look at the factors involved in making such a decision, I should clarify what I actually am referring to when I say choosing between TeX and LaTeX. When I say LaTeX, it is quite clear what I mean. TeX, on the other hand, can refer to many things: `plain` TeX, or sets of macros or macro packages that may be commercially or publicly available. In my view, you cannot compare `plain` TeX with LaTeX, and in fact you will probably not find too many TeXers that work in `plain`.[3] So I am actually comparing the many faces that TeX can have to the LaTeX macro package, and, as I will later state, this becomes one of the factors that must be examined.

Why the Need For a Choice?

The obvious answer is because both TeX and LaTeX exist. Obviously, they were both designed with specific uses and needs in mind. The key is to identify what the user wants to do, and then match their needs to what is available. Unhappily, the question of choice between the two systems is one that doesn't really have a definite answer. That is, I can not confidently say to you "TeX is better than LaTeX" or "LaTeX is better than TeX". There are several other situations that come to mind that pose the same problem: choosing between PC's and Macs, the NFL and the CFL,[4] or Big Macs and Whoppers. Perhaps these are silly examples, but the scenario in each case is the same. Sure, I have a personal preference, but if I choose strictly based on that, then in reality I am wearing blinders. This usually results in users of one product saying bad things about the other (i.e., TeX or LaTeX is bad, PCs or Macs are terrible) product without really knowing anything about it. The point that I am trying to make is that we should be willing to accept the fact that simply because we have the option of making a choice (i.e., both products exist) means that there is more than one

[3] No offense intended for those of you who do work with `plain`.

[4] Canadian Football League.

market to satisfy. So instead of trying to show how bad one is, we should gear our efforts towards making that choice easier for others.

1. Separate Entities?

Some say that we should look at TEX and LATEX as separate, independent systems. While this may be useful in explaining things to a TEX newcomer, it is also important for them to know that TEX and LATEX do have an important relationship. For without TEX, we wouldn't have LATEX, since the LATEX macros are written with `plain` TEX. Making a user aware of this is important so that they know that they can modify/create macros if they wish to take the time to learn `plain` TEX. It is often thought that LATEX is a separate entity loaded with everything (all the features). However, as we know, no piece of software can do everything.

Using the TEX System

I like to think of TEX as a programming language, since the concept is quite similar to that of writing programs. In actuality, from a text processing point of view, TEX is a markup language (as opposed to conventional WYSIWYG[5] word processors). That is, you create a file that contains a mixture of text and TEX formatting commands, and so, as the user introduces commands into their document, they must visualize how the text will be affected. With a computer program, you must try and visualize how data will be affected by various instructions. Standard word processors let you see changes on the screen as you make them, a concept that far more users are comfortable with. Learning how to program is quite difficult, and really is only understood by a small percentage of computer users. In fact, I would say that only a small percentage of computer users[6] actually understand how the applications that they use work. Most of them work from memory; they know that this command does this and that command does that. This is why they cannot solve problems when they arise. On the other hand, learning programming languages develops a good approach to problem solving.

When choosing between TEX and LATEX, this programming language analogy becomes important since not everyone can learn how to program. However, if we accept the statement above about memory usage, then almost anybody should be able to learn to use TEX. So the choice is reduced to either learning the language (and in depth, I might add) and creating your own macros, using a set of `plain`-based macros (usually a locally developed package created to address local needs only), or using LATEX, a standard macro package that addresses a wide variety of needs (in fact, most), and is easily accessible to everyone.

[5] What You See Is What You Get.
[6] 10–15%, in my estimation.

2. Working Environment

The environment that the user is working in (or planning to work in) can have a direct bearing on which system they should use. It is often said that TEX itself is usually the first part to be installed and working. This is probably true since setting up and running basic TEX is often easier than with some of the associated macro packages. Keep in mind also the nature of the many different operating systems that TEX can work under. Some of them handle TEX quite well, but when it comes to LATEX and reading in style files, some systems can provide systems personnel with a formidable challenge. An example of this would be MVS where you must work within the system restrictions on the Job Control Language. As for working on a PC, both TEX and LATEX are readily available, so the only restrictions are the limitations of the PC itself (memory, storage, operating system, etc...).

3. Macro Writing

We cannot discuss choosing between systems without some mention of macro writing and how it can affect a user's choice. Whether you choose to use LATEX or a TEX macro package, you are just using a bunch of macros that somebody has written. The degree to which a user understands the concepts and benefits of macro writing will probably weigh heavily on their final decision. Macros exist to make life easier for computer users, no matter what software they're using. They are most often used either to make something that is difficult seem easy, or to reduce the number of commands required by a user, usually by grouping oft-repeated commands together under one command. Thus they serve the dual purpose of removing the user from the low-level intricacies of the software, allowing more users to to use and understand the software, and they can increase productivity because of the nature of their power (executing many commands with just one). It is because of macros that we hear the term "power user".

The ability of the user to understand how macros work goes a long way to choosing which system to use. Knuth put most of his effort into the low level (or 'guts') of TEX, and not into macro writing. This may explain why it is difficult to create documents using the `plain` macros. Even the macros he created for *The TEXBook* (`manmac`) are not all that useful to most users since they were created only with the goal of being used in the book. Since creating TEX macros is sort of like writing programs, we cannot expect every user to understand how they work or how to create their own. In fact, there are a lot of users who have no desire to know how the system works, only that they can easily accomplish what they have in mind. Many people will say that it doesn't matter, since it is "easy" to write and/or modify simple macros. But the bottom line is that there are lots of people who can't/don't want to.

LATEX's macros are advantageous in that they are written for a general purpose as opposed to local macro packages, which are most often created for specific

solutions. General macro packages usually are maintained more constantly, and thus become more portable. LATEX is designed to work with a wide variety of systems, whereas there is no such guarantee for locally written macros. From a user's point of view, LATEX offers the chance to start writing right away without having to worry about creating any macros. However, this may also be the case with local macro packages So, while macro writing may not be for everybody, it can be done, and you can customize existing TEX/LATEX macros, but, for the most part, it is difficult, and you really have to know what you're doing, or some strange things can happen. However, LATEX's associated files are very well documented, which can ease the burden of macro modifying. It is also important to note that most TEX macros can be used with LATEX.

4. Availability of Public Domain Material

One of the nicest things about the TEX community is the way in which it shares solutions and macros. There are hundreds of TEX macro packages and LATEX style files that are available for the taking through various electronic mail repositories such as the one for LATEX files at Rochester,[7] or which can be acquired on floppy diskettes at events like the Annual Meeting. Based on the old "why reinvent the wheel" concept, users can use many of these files to solve problems that they have run into, or to do something which may otherwise seem difficult. Basically, the availability of such files allows a wider variety of users to make use of more features of TEX or LATEX, without the need to know how to write or change macros. It is simply a matter of letting users know where the files are, and how they can be obtained.

What About the User?

As with most software packages, one of the most important factors to take into consideration is the nature of the user. You must determine the needs of the user, and try to recommend something that will suit their needs and that they will be comfortable using. We must also keep in mind that users come in every shape and form. That is, with a varying degree of computer knowledge. Not every potential user is going to be too keen once they see how TEX works. Of course, user friendliness is in the eye of the beholder. But I think that far more users are more comfortable with a regular old WYSIWYG word processor, since this is probably the way they've always worked. Even if they haven't used a word processor before, its concepts are still probably easier to learn for a new user.

The first task would be to get the user to actually want to use TEX. Once (if?) this is done, you should then determine what they want to produce and what options they have. For example, if a user needed to produce a lot of tables, and there wasn't a good table-making macro package available, then LATEX would

[7] Since moved to Clarkson.

probably be better for them. I'm not going to elaborate right here, since I will talk about features a little later on. The important thing to recognize is that different types of users will adapt more readily to either TeX or LaTeX, and we must try and find out what they want to do, and what would be easier for them. Many users have told me that, while TeX may be a little harder to master than a word processor, the results are well worth it!

Advantages and Drawbacks

As I mentioned earlier, it would be unfair (if not impossible) to compare `plain` TeX and LaTeX, since most people would not create documents with `plain` TeX.[8] Also, there are not very many macro packages available (for little cost, that is) that contain all the features that LaTeX does. Therefore, I'll only point out the LaTeX features that I consider extremely valuable, and which would require extensive TeX knowledge (and time) to write the equivalent TeX macros. Basically, TeX offers many advantages to the experienced TeX user, while LaTeX offers more to the beginner, starting with the premise that, while it is beneficial to understand TeX in order to better understand LaTeX, it is not imperative.

5. Flexibility

TeX is very flexible, in that you are in total control, with the ability to change anything; LaTeX for the most part does what it was intended to do well, but it can be very inflexible when it comes to trying to modify the way it does some things. LaTeX users who understand the TeX language well will often be able to get around this. TeX is also more flexible in that you can write macros to function any way you want, while with LaTeX you may be restricted by the way that LaTeX itself works. A distinct advantage of LaTeX is the ability to alter the appearance of the document simply by choosing another style file. This point highlights a couple of reasons of why LaTeX was developed:

- LaTeX reduces the page formatting setup overhead that TeX requires. This may involve knowledge of many TeX commands and how TeX's output routines work, something that not every user can be taught to do.

- Using LaTeX encourages a logical approach to document preparation, which is what Lamport had in mind when he was developing LaTeX.[9] The user need only worry about the text and not the document layout. Many TeX documents are cluttered with unnecessary commands to alter formatting.

[8] Except for very small documents.

[9] See his article in *TUGboat* 9(1): 8–10, 1988.

6. Features

The features that I feel would be an important reason to select LATEX include the following:

- Tables
- Cross-referencing
- Automatic numbering
- Bibliography formatting (with BIBTEX)
- Handling of large documents

Trying to set up complex, nice-looking tables with TEX is difficult, even for the experienced TEX user. The LATEX table-formatting macros make things relatively easy, and offer a number of options. This is not to say, however, that LATEX will solve all your table creation problems. A macro package such as Michael Ferguson's table-making macros can make even the toughest of tables seem quite trivial, and, with a couple of slight modifications, they can be made to work with LATEX.

The ability to cross-reference tables, figures, equations, citations and the like becomes invaluable in a large document. This is a really handy and easy to use feature of LATEX. The only drawback is having to run two passes of the document to match the references, but, hey, you can't have everything!

It is also nice to be able to have various parts of the document numbered automatically. Be it titles, equations or figures, LATEX provides numbering for most aspects of a document. Having everything numbered automatically makes it easier to insert, delete or move things around in a document without having to worry about the numbering or references. You can also have nested (or different levels) numbering, turn the numbering off, or easily alter the value of any of LATEX's counters. In defence of TEX, it should be pointed out that you dont't need a whole lot of TEX knowledge to be able to create macros that use and/or create TEX counters for automatic numbering purposes.

If you create many documents that require extensive bibliographies, the BIBTEX system that comes with LATEX can eliminate most of the work required to make bibliographies. BIBTEX allows you to create a bibliographic database, with entries that can be accessed by any LATEX user. BIBTEX is an excellent feature and could have some impact on a user's choice of system, but it should only be taken into consideration if the documents they are creating require a lot of bibliography work.

LATEX also handles large files much better than TEX. This is in evidence when large documents are broken up into separate files which LATEX can selectively

process. Users who work with large documents most of the time will appreciate working with LaTeX rather than a TeX package.

Some other features which may be mentioned include indexing and generating tables of contents, but there are widely available macros for these that work well with both TeX and LaTeX.

7. Portability

Portability becomes an important issue to think about when you want to send documents to other people, or when you want to run or print documents at a site other than where they were created. In my view, LaTeX offers much more on the portability side than TeX macros. This is because of LaTeX's general nature. It comes with a number of associated style and other files, which are all included on any standard distribution, so any site running LaTeX should have all the files that you need. With locally developed macro packages, you must send a copy of the macros along with the document, and, those macros can potentially cause problems. They may need to be modified in order to work with another system, and then all kinds of problems can occur.

Another problem that can arise is when macro files are embedded, and invoked, from within document files. Unless you are told this originally, you will need to search the files to see if this occurs. With LaTeX files, it easy to see this just by looking at the `\documentstyle` command, which is usually the first thing in the document. Fonts can pose another problem. The `plain` TeX format file does not preload very many fonts; instead they are loaded within the document with TeX `\font` commands. With LaTeX, many more fonts are preloaded, so this can ease the problem of site differences as to availability of fonts.

It should be noted, however, that not everything is peaches and cream regarding LaTeX portability. In fact, LaTeX incompatibility is often underestimated, due mainly to two factors. One is the LaTeX version number (the now-famous 2.09), and the way updates are handled. Newer versions of LaTeX and its associated files are indicated with dates rather than numbers, which makes it difficult to tell if you are using the same version as somebody else. Although announcements are made periodically, and the upgraded files are available through electronic mail repositories, it still requires some poor site person who has to make sure that the newer files are acquired and set up, and that users know about them. There have been many changes to the LaTeX files over the years, and so there are potentially that many different versions running out there, which can cause major headaches. The other problem lies with local "gurus" who modify LaTeX style files. If you want to modify the LaTeX styles, at least give them new names.

8. Documentation and Support

The documentation available to make a potential user's life easier while learning about TeX or LaTeX will probably not influence their choice too much, but it

could be important to some people. Some will take more readily to learning, while others may need lots of hand-holding types of manuals. Unfortunately, not too many of these exist. From a broad-spectrum-of-user point of view, both *The TEXBook* and the LATEX manual have some major flaws. The LATEX manual is good as a user manual, but not so *The TEXBook*, although it is an excellent reference manual. You can find just about anything about TEX that you want to, but I wouldn't give it to a user to read to try and learn how to use TEX, for they would probably give up and find something else! On the other hand, the LATEX manual is seriously lacking in terms of examples for the various LATEX commands.

There are some manuals available within the TEX community that can make it easier for users to understand and learn about TEX and LATEX. They include Arthur Samuel's *First Grade TEX* and Michael Urban's *An Introduction to* LATEX.[10] Stephan v. Bechtolsheim's *Another Look at TEX* will provide the user with more TEX examples than they can imagine, and I have several other similar documents and manuals that I have picked up either from TEX or LATEX users I've met at Annual Meetings or through electronic mail.

As far as support goes, one aspect will of course be local support: to what degree is TEX supported, and how knowledgable or accessible is the local guru? But there are other areas of support such as TEXhax and TEXMag where users can make inquiries about almost anything about TEX and expect to get an answer(s) that can help them solve their problem. One of the nicest things about TEXhax is that LATEX users can converse directly with Leslie Lamport, the author of LATEX. The fact that he takes the time to answer users' inquiries is something that one should be grateful for. Of course, most of us know that we had better be sure we're not asking him about something that's clearly been documented!

One thing I have noticed about the nature of entries to TEXhax is that while TEX questions usually are about wanting to know how to do something, there are a lot of gripes and complaints about LATEX. I have heard several TEX users ask "If LATEX's so good, then why do a lot of people complain about it in TEXhax?" However, I think that many of these complaints can be attributed to: poorly (at times) documented commands in the LATEX manual, not using LATEX the way it was intended to be used and lazy readers who won't or don't know how to write or change macros to satisfy their requirements. These users should be thankful that Lamport even takes the time to acknowledge what they have to 'flame' about.

Conclusion

As I said earlier, there is no definite answer to the question of choosing between TEX and LATEX. Almost every situation will vary as to how advanced the user is,

[10] Both available from the TEX Users Group.

what they want to do, and what essential features they need to have. The key is to be able to identify these needs, and then determine which system will give the user more benefits. Regardless of what they choose to do, they have made a good start by choosing the TeX system, so experienced TeX users shouldn't be 'putting down' TeX by saying that one component or another of the system is bad or shouldn't be used. This only brings up TeX in a negative aspect, which is exactly the opposite of how we should be trying to promote TeX. Sure, rough edges do exist, but why not try and improve on them instead, and realize that we *can* live both TeX and LaTeX.

Bibliography

Ferguson, Michael. Table Making with INRSTeX. *TeXniques* Number 2. Providence, R.I.: TeX Users Group. 1986.

Knuth, Donald E. *The TeXbook*. Reading, Mass.: Addison Wesley. 1986.

Lamport, Leslie. *LaTeX: A Document Preparation System*. Reading, Mass.: Addison Wesley. 1986.

Samuel, Arthur L. *First Grade TeX: A Beginner's TeX Manual*. Stanford: Stanford Dept. of Computer Science. Report No. STAN-CS-83-985. 1983.

Urban, Michael. *An Introduction to LaTeX*. TRW Software Productivity Project. 1986. [Reprinted by the TeX Users Group, Providence, R.I.]

9. Unpublished Sources

Doob, Michael. A Gentle Introduction to TeX: A Manual for Self-Study. University of Manitoba. 1988.

McPartland-Conn, Marie. A TeX Primer. Henco Software. May 1987.

NRTC LaTeX Manual. Northrop Research and Technology Center. May 1987.

Urban, Michael. A Guide to TeX for the Troff User. TRW Software Productivity Project. No date.

von Bechtolsheim, Stephan. *Another Look at TeX*. Latest version: 3.5, July 1988.

Mathematics TEXtbook Publishing with Japanese TEX

KAZUHIRO KITAGAWA AND NOBUO SAITO

Department of Mathematics
Faculty of Science and Technology
Keio University
3-14-1, Hiyoshi, Kohoku
Yokohama, 233 Japan
kaz%keio.junet@relay.cs.net
ns%keio.junet@relay.cs.net

ABSTRACT

TEX was originally designed to typeset mathematics text. In this paper, we will describe a practical experiment to produce a non Latin-style mathematics textbook using the Japanese TEX system.

Introduction

In *TUGboat* 8, no. 1 (1987), an article appeared which very much interested us: "Book Publishing Using TEX", by Tony Siegman. We have just finished publishing a mathematics textbook, here at Keio University, written mainly in Japanese, with English mixed in; it is the first textbook to be produced using Japanese TEX. The book has about 300 pages, including exercises, answers, and supplementary assignments. TEX was, and is, preferred by our mathematics professors for some of the following reasons:

1. TEX has many good mathematical typesetting facilities
2. The exercises and extra assignments can be changed at any time, usually at least once a year
3. Other mathematical contents can always be updated at their convenience

In our case, text sources written in Japanese were prepared, revised and rearranged for a couple of these yearly revisions. Originally, text files were prepared with Japanese word processors and stored on 8-inch floppy disks; they were then converted to DOS files, but some characters were usually lost. At the time, TEX for PCs didn't exist, but we did have Japanese TEX running on our UNIX machines; so the DOS files were transferred once again, so that we could use TEX to produce the high quality output we wanted.

Most of the work is done on SUN and VAX machines by volunteer undergraduate students from our department, who usually don't know anything about TEX or UNIX. Fortunately, they have been using PCs and PDP-11s running under RT11, which was very helpful when I then taught them TEX and UNIX.

TEX, AMS-TEX, or LATEX

There are three major TEX systems: `plain` TEX, AMS-TEX, and LATEX. AMS-TEX, which is similar to TEX, adds its own macros after the `plain` TEX macros have been loaded, whereas LATEX is a distinct and separate system.

In our textbook publishing case, we decided to use AMS-TEX to prepare the source files. Most TEX users are familiar with both AMS-TEX and LATEX, with the former preferred for mathematics. Our decision was based on the following points:

1. AMS-TEX supports well-designed mathematical typesetting facilities and is good for writing mathematical equations
2. AMS-TEX source files are accepted as submissions for publication by the American Mathematical Society (AMS)
3. AMS-TEX macros follow the same structure as the `plain` TEX macros (there is no new syntax to learn)
4. Since AMS-TEX is added to `plain` TEX, we can use macros from both systems, without running into macro interference (this is not always the case with combining TEX and LATEX)
5. In LATEX, some of the mathematical typesetting facilities supported in TEX are eliminated

This is not to say that LATEX is an inferior product, only that it was not the most appropriate choice, given our situation at Keio University. Most of our source files of mathematical equations are first prepared and maintained by mathematics professors and students, and some of them are retrieved from files already written in AMS-TEX. Most of the TEXperts of mathematicians are willing to use AMS-TEX to write their papers. Considering "human efficiency" (the re-training for LATEX and the reworking of already existing AMS-TEX files into LATEX source, including macros for mathematical typesetting), we selected AMS-TEX for our mathematics textbook publication.

Macro Package

Once we had decided to use AMS-TEX, however, we found that its structured documentation facilities, compared with LATEX, were not adequate for large documents. There is no doubt that declarative markup is superior to procedural, in

order to maintain and write large documents easily. At best, AMS-TeX provides only an **amsppt** style for the UNIX world.

We believe a well-designed and comprehensive macro package is one of the most important components to produce books with TeX. We have therefore developed such a package for TeX. As we used it, we revised our macros, and have also incorporated suggestions from our volunteers and from a publisher, in order to make it better.

The macro package focuses on logical document structure, not on the equations. Basically, it does not conflict with the AMS-TeX macros, and works well with TeX on its own. Some of our macros are given as style files from AMS-TeX, and others are preloaded to `plain.tex` with `amstex.tex`. The following are characteristics of our macros:

1. Structured Document Markup:

   Our macros fully support structured documentation. The markup tags that specify document structures start with `\beginxxx` and end with `\endxxx` (`xxx` is a string), only slightly different from the `\begin{xxx}` and `\end{xxx}` syntax in LaTeX. Each part of the document must be enclosed by a pair of tags; for example, a section starts with `\beginsection` and ends with `\endsection`. These macros also support spacing, floating, and so on, and therefore source files have no commands to specify physical structure, such as `\vskip` `\hskip`, etc.

2. TeX Parameters Specific to Japanese Fonts:

   In Japan, a typical textbook uses 8, 9, 10, 12, and 17 point fonts, without magnification. There are some basic differences, however, between English (Latin-based) and Japanese (Kanji) fonts, which require certain adjustments to some of the standard TeX paramters (i.e., `\baselineskip` and `\parindent`).

   Most Japanese fonts have the same height and width, whereas Latin-based characters are generally higher than they are wide. As well, the black ratio of most Kanji fonts is larger than that of English. When using Kanji fonts, the narrow lineskip used for English makes the whole document very difficult to read, because of this black ratio. We have widened the baselineskip and changed related values from the standard TeX 10pt values, in order to achieve the look of regular Japanese books.

The most difficult problem is indexing and referencing in Japanese. In English, words are sorted by alphabetical order, but Japanese words are sorted according to their pronunciation. Sometime even the same Kanji character can have several pronunciations, which means the word will be sorted into several different positions in the list. Our code follows standard pronunciation, but we deal with such cases as different characters with different pronunciations. This

problem is intrinsic to Japanese. One solution is to enter such words in phonetic form, using Hiragana codes (Hiragana codes are based on sounds, not spelling). We designed index macros which take two arguments: one is the citation for the index, the other is its pronunciation, as specified by these Hiragana characters. The user must therefore specify sorted sequences explicitly with Hiragana, in order to indicate the correct pronunciation, which then gives a correctly sorted index.[1]

Indices which include French, Russian, and so on, along with Japanese are the most difficult and complex to sort. The non-ASCII characters of Latin-based characters are assigned control codes in TEX or they are built by combining a few characters.

Conclusions

Following this page is an example of a source file containing our macros, along with the TEX output. Our mathematics textbook, generated with Japanese TEX, was printed on a laser printer, and then xerox copied and bound by a commercial firm. However, it would be better to generate the output from a phototypesetter; the Dai-Nipon Publishing Company is now preparing to install Japanese TEX onto their phototypesetter. So, we expect that next year's version of our textbook will be improved both in mathematical and in print quality.

Bibliography

Siegman, Tony. Book publishing using TEX. *TUGboat* 8(1): 8–11, 1987.

Acknowledgements

We wish to thank our volunteers, Miss Miho Ike, Mr. Hidetoshi Unno, Miss Tomoko Ohkawa, Miss Yuri Kiribuchi, Miss Jun Sato, and Miss Hitomi Mitsumori for their excellent help.

Professors in the Department of Mathematics at Keio University have given us many suggestions, and have checked the book several times.

Finally, I am indebted to Mr. Hisao Miyauchi at Iwanami Publishing Company for his advice, and to Mr. Jou Okubo at ASCII Publishing Corporation who helped convert the original source files into DOS.

[1] The VoRTEX program developed at the University of California/Berkeley provides a similar *makeindex* command which supports option sorting by pronunciation; this is very similar to our solution.

Example: TEX source

```
\beginchapter{線形空間(ベクトル空間)と線形写像}

第1章で， ${\bf R},{\bf R}^2,{\bf R}^3$ の数空間,
さらには$n$次元数空間の定義を与えたが,
これらは位置を示す“点”としての集まりのものにすぎなかった.
今，我々はこれらが“数”としての性格(演算のある！)
を持った集合である事を述べたい.
丁度，高校の時に，位置ベクトルを習い，それらの和,
スカラー倍等の演算を定義した事を思い出そう.

\beginsection{線形空間としての ${\bf R}^n$}

$n$ 次元線形空間${\bf R}^n$ は，定義により
$$
{\bf R}^n=
  \{x=\pmatrix{x^1\cr \vdots\cr x^n\cr} |\,
    x^1,\cdots,x^n\in\ \kern -0.5em    {\bf R}\} \leqno{(2.1)}
$$
であった事を思い出そう.

今，${\bf R}^n$ 内に{\bf 加法，スカラー倍}と呼ばれる演算を次の様に定義する.

\begindef{1.1}
$x=\pmatrix{x^1\cr \vdots\cr x^n\cr},\,
y=\pmatrix{y^1\cr \vdots\cr y^n\cr}$ を ${\bf R}^n$ の点とする. この時,

$$ x+y=\pmatrix{ x^1+y^1 \cr \vdots \cr x^n+y^n \cr}
\kern 2em \hbox{($x,y$ の各第 $i$ 成分を加える. )} \leqno{(A)} $$
を，$x$ と $y$ の{\bf 和}といい，$x+y$と書く. \linebreak
(B)\quad $a$ を実数として,

$$ ax=\pmatrix{ ax^1 \cr \vdots \cr ax^n \cr}
\kern 2em \hbox{(各第 $i$ 成分を $a$ 倍する. )} $$
を，$x$ の $a$ {\bf 倍(スカラー倍)} といい，$ax$ と書く.
明らかに，$x+y,\,ax$ 共に ${\bf R}^n$ の元である.
\enddef

\beginexercise
 1 次元直線 ${\bf R}$ では，ただのたし算，かけ算である事を見よ.
\endexercise

    ...

\endsection
\endchapter
```

Example:TEX output

第2章　線形空間(ベクトル空間)と線形写像

第1章で，$\mathrm{R}, \mathrm{R}^2, \mathrm{R}^3$ の数空間，さらには n 次元数空間の定義を与えたが，これらは位置を示す“点”としての集まりのものにすぎなかった．今，我々はこれらが“数”としての性格(演算のある！)を持った集合である事を述べたい．丁度，高校の時に，位置ベクトルを習い，それらの和，スカラー倍等の演算を定義した事を思い出そう．

§2.1　線形空間としての R^n

n 次元線形空間 R^n は，定義により

$$\mathrm{R}^n = \{x = \begin{pmatrix} x^1 \\ \vdots \\ x^n \end{pmatrix} \mid x^1, \cdots, x^n \in \mathrm{R}\} \tag{2.1}$$

であった事を思い出そう．

今，R^n 内に加法，スカラー倍と呼ばれる演算を次の様に定義する．

定義 1.1　$x = \begin{pmatrix} x^1 \\ \vdots \\ x^n \end{pmatrix}, y = \begin{pmatrix} y^1 \\ \vdots \\ y^n \end{pmatrix}$ を R^n の点とする．この時，

$$(A) \qquad x + y = \begin{pmatrix} x^1 + y^1 \\ \vdots \\ x^n + y^n \end{pmatrix} \qquad (x, y \text{ の各第 } i \text{ 成分を加える．})$$

を，x と y の和といい，$x + y$ と書く．

(B)　a を実数として，

$$ax = \begin{pmatrix} ax^1 \\ \vdots \\ ax^n \end{pmatrix} \qquad (\text{各第 } i \text{ 成分を } a \text{ 倍する．})$$

を，x の a 倍(スカラー倍)といい，ax と書く．明らかに，$x + y$, ax 共に R^n の元である．

問題 1.1　1 次元直線 R では，ただのたし算，かけ算である事を見よ．

Approximate TEX for Semitic Languages

Jacques J. Goldberg

Department of Physics
Technion-Israel Institute of Technology
Haifa, Israel
phr00jg@technion.bitnet

ABSTRACT

An approximate solution to TEXing in Hebrew is presented. A font family good enough for office quality documents has been created. The process of bidirectional text entry is discussed. Because words in Semitic languages are usually short and hyphenation unnecessary, a few simple manipulations produce acceptable documents without any change in the TEX program proper. They are implemented as a preprocessor to TEX and a tiny set of macros. Freely available on a Bitnet server, this solution will work reasonably well on any TEX installation.

Why This Work?

An outstanding virtue of TEX that will never be emphasized enough is its universal availability and compatibility on just about any existing computer. Modern scientists rarely work alone; with the availability of academic electronic mail networks, collaboration has turned real-time, and parts of documents are frequently simultaneously written at several remote locations on various machines. Using TEX makes merging easy, and all authors instantly see the same text everywhere. We thus want TEX.

But some people say TEX is difficult to learn: they seem to have greater difficulty writing what they want in plain English following a \ than memorizing tens of control characters and escape sequences expected by their preferred, incompatible, word processor. At our place, the ultimate argument against joining the TEX world was: "Anyway, we need Hebrew capability for educational and administrative office work, and we do not want to deal with several word processors!".

Hebrew TEXing, in our scientific and technological University environment, required the following:

- A terminal and an editor to input bilingual text
- A font, perhaps a family (regular, bold, slanted, various sizes)
- A solution to the problem of mixed right-to-left and left-to-right formatting

- A machine-independent implementation of TEX and the DVI drivers for our vast range of computers, displays, and printers used for TEXing at the Technion (DEC-Rainbows, IBM and other MS-DOS based personal computers, MacIntosh systems, VMS and UNIX VAX machines, an IBM 3081-D VM system, Epson and clone printers, many DEC-LN03 and one IBM 3812 laser printers, and probably more which I don't know of).

Text Entry

The Hebrew alphabet has 22 letters, of which 5 have an extra alternate glyph when final (last in a word). The Latin alphabet has 26 letters. One way to enter Hebrew text—the one I would prefer if a standard could be introduced—could be to represent each of the 22 Hebrew characters by one Latin character, probably in upper case for easier context reading, typed from left to right, that is, just as in English. This would not be a phonetic representation but mere character coding. The computer could easily be programmed to select the final glyph when required. Such Hebrew and mixed language text could thus be typed on any terminal with any text editor. Adequate context marking for eventual mirroring is discussed below. I experimentally found out that physicists are unable to suggest such a convenient encoding scheme: linguists should do it for us!

In the old days of punched card data processing and the so-called BCD coding and its 64 possible values, a convention was introduced in Israel to represent Hebrew characters by the same 26 values reserved for English, plus one of the special characters. With 128 values allowed by the ASCII code, Hebrew cohabited with the 26 English lower case characters, plus the left quote to make 27. This code is now known as "old ASCII". Because, of course, there is "new ASCII", following (very late) the birth of EBCDIC and its 256 possible values: there, the 27 Hebrew characters found a home, expelling however 27 non-standard graphic symbols.

Several manufacturers optionally offer Hebrew glyphs in the display ROM of their terminals or microcomputers, matching one of these codes or sometimes both. With the new code one can see a *b* and a ג on the screen, but, with the old code, where *b* and ג have the same representation, either *bb* or גג will be displayed at will, selected by a special local control key. An additional local control key (or combination) toggles the mapping of the keyboard to either of the two sets of codes, to keep the Hebrew keys at their standard national keyboard location, a must for fast typing.

A powerful equipment manufacturer is contributing to this state of anarchy by preventing 8-bit character transmission on asynchronous serial lines to its mainframes, while using 8-bit coding on its mainframes, personal computers, and own mainframe dedicated terminals. Their personal and large computers use incompatible code, and the former cannot be used as terminals of their own mainframes with the full 256-value code!

This is why I selected for myself a Zentec Zephyr terminal (a DEC VT220 clone) working in "old ASCII" mode, hooked up through a manual switch to either an IBM 3081 under VM, a VAX 785 under VMS, or a self-assembled MS-DOS machine based on the AMPRO LB-186 board (running PC-DOS with no graphics display but with a full megabyte of RAM under normal DOS program control, which allows resident Personal-REXX on top of heavy TEX jobs).

With that terminal, at a keystroke, I can display Hebrew or English glyphs for lower case and the left quote, and toggle the keyboard mapping from Hebrew to English; another keystroke reverses the direction of the sweep on the cathode ray tube.

With such settings, bilingual text is typed in a natural way, typed characters are recorded by any regular editor as they come in, and the user reads readable English text while typing in English, with unreadable segments visible from the other language, and vice-versa.

It immediately comes to mind that sweep reversal could be avoided by typing Hebrew text in push (insert) mode: the current character in Hebrew context comes to the left, not to the right, of the previously typed one. One Hebrew word could thus be typed in the natural way and, with the 8-bit coding scheme, could be displayed in a readable way along with other English text. Such support exists on specially equipped PCs, and with modified *vi* on our UNIX systems. This technique however records sequences of words which are controlled by the end of each edited line, to which TEX is completely transparent: "What you see is all you get!". This could of course be managed by adequate postprocessing.

A Font Family

METAFONT was used to make the simplest possible font good enough for academic office work. The REDIS design described by L. F. Toby[3], was selected and encoded by E. Atashy, B. BenAbou, F. Melamed and S. Morim, as a Computer Science student project assignment.

The font was generated at several usual sizes, also slanted and bold. Punctuation was incorrectly borrowed, through my error, from the Roman family of Computer Modern fonts: it is currently being replaced by the punctuation designed for sans sérif fonts, to which the REDIS design is closer.

Our kerning is still far from good. This problem is related to the TEXing technique described below. It turns out to be very difficult for people used to Hebrew, to program METAFONT kerning of a glyph *following* the previous glyph, while for the reader it will *precede* it.

A better quality font is under development, that will meet textbook printing requirements.

Finally, a Parsi font (or Pharsi, as Semitic languages use the same glyph for both sounds, just as in Philistine-Palestine) was written by some of these

authors. It forms the base of a nearly completed Arabic font, as can be seen below.

TEXing

Donald Knuth and Pierre MacKay[1] have extensively described the general problem of mixing right-to-left with left-to-right texts with TEX. They have also shown the changes required in TEX and the DVI drivers to implement their solution. After the first version of this contribution had been submitted, Larry Denenberg told me that he had implemented the required changes in a particular DVI driver for a particular printer under UNIX. I was very happy to hear at this meeting that he already completed a DVI-to-DVI filter based on the same modifications, which I had suggested him to write instead of having to modify all existing drivers.

Unfortunately, we extensively use microcomputers and peripheral equipment for which the source code for the required programs was not available (and quite a bit of debugging expected if it were, in order to implement TEX-XET). In addition, as soon as we use a special version of TEX or drivers, we lose compatibility or become bound to multiple versions and multiple updating.

Fortunately, TEXing mixed language requires two very simple operations on text typed in as described earlier:

- reversing the order of the letters in each Hebrew word independently
- placing the next Hebrew word at the left of the previous one, or at the right end of a new line of text.

Unfortunately, good text composition requires hyphenation.

Fortunately, Hebrew is such that a *physicist's approximation* to hyphenation, to simply ignore it, or agree to hand-hyphenate here and there, turns out to be very reasonable:

- In his 1973 Harvey Prize reception address at the Technion, Prof. Claude Shannon showed the following text "THS S TH PRF THT NGLSH S RDNDNT" to establish written language redundancy. He allowed the audience one minute to decipher this line. Nobody understood why the minute, since all of us read and write Hebrew *without vowels* anyway!
- One hundred years ago, E. Renan[2] explained that Semites could get along with words of up to three letters, being anyway unable to express anything abstract, philosophical, or scientific! He reported a count of a few thousand Hebrew words only, easily obtained by arrangements of three letters, making up a limited vocabulary hardly more than a young child might know; this was of course used as a proof that Semites were lower grade human beings, later re-assigned by others to lower grade animals, to be treated as such. As

an example, compare the following French word (German has better ones) with its Hebrew translation, to be convinced that Hebrew seldom requires hyphenation:

anticonstitutionellement = בנגוד לחוקה.

To insert one or two Hebrew words in an English piece of text is very simple. A TEX group delimited by a pair of curly brackets specifies the font and the string to be typed, without even invoking the `\reflect` macro described by Knuth and MacKay[1] if the string is readily typed with the last Hebrew letter first, or invoking `\reflect` if typed in natural sequence (it's sometimes less work to type the word(s) beginning from the end than invoking the macro. Words are *so* short!).

TEXifying a Hebrew segment requires two steps (we assume the text was typed and collected by the editor in natural sequence, not with the push-insert technique). Each Hebrew word is first reflected, and then each reflected word is supplied to TEX as the argument of a macro which will place this word in a box *extending to its left* the current box making up the current line (empty to begin with, of course). Context markers indicate areas where words should or not be individually reversed, as well as the beginning (empty line) and completion of a Hebrew segment. In the current implementation a vertical bar was used to toggle Hebrew segment flagging, and the dollar sign to hold word reflection for left-to-right inserts inside a Hebrew segment (the reason is obvious: we need dual language TEX to write equations and math symbols in our material).

With these very simple operations, fairly good TEXing has indeed been achieved, as shown in the following ad-hoc short example, which could be part of a typical educational document:

תופעת Compton.

התורה האלקטרומגנטית הקלסית כבר ידעה לחזות, וכן אושר נסיונית, כי גלים אלקטרומגנטים מסוגלים לשאת תנע ואנרגיה. הנחת הקונטיזציה לא הפריעה לעובדה הזאת. ההנחה רק קבעה כי האנרגיה תמיד הופעיה בקונטים שווים ל $h\nu$. אבל היה ידוע הקשר בין התנע והאנרגיה:

$$p = E/c$$

אם האנרגיה מתקבלת במנות של $h\nu$ אז נראה בלתי נמנע כי גם התנע יתקבל במנות קצובות. לכן מצפים למינון התנע על פי הנוסחה:

$$p = \frac{h\nu}{c} = \frac{h}{\lambda}$$

כאשר λ הינו אורך הגל.

It should again be emphasized that this is a physicist's, approximate, rudimentary solution, whose single virtue is to leave TEX itself, the widely used macro packages, and the DVI-drivers, absolutely unaltered.

Implementation and Availability

A first, awkward implementation is running under MS-DOS, VM, VMS and UNIX at least, on our site. It is currently being modified, thanks to suggestions from users, in particular from Ron Greenberg at MIT who pointed out an easy and elegant alternate to my clumsy preprocessor-TEX interface, which makes the method fully transparent to the user and to other macro packages. This report was indeed produced with the new version.

Step one, context sensing, and Hebrew word parsing and reflecting, is implemented as a preprocessor to TEX, written in **C**, feeding TEX with a temporary intermediary file instead of the original (it could as well be performed by interpreting the original input with TEX macros, but the process would be slower, and, for me, much more difficult to write). There is one conditional line of code in the **C** program: one of the systems senses the end of the input file differently. The edited file is given the filetype `.ivr` and the intermediary one the filetype `.tex` so that the final DVI file keeps the name of the original.

Step two, building lines by extending the current box to the left, until its size exceeds the allowed line size, in Hebrew mode only of course, is achieved by a set of very simple TEX macros. To distinguish them from anything else, their names all start with `\ivr` (from the Bible's Avraham Ha'Ivri, of course).

The sequence of operations is therefore almost identical to regular TEXing. The `.ivr` file is first edited, then preprocessed, and the preprocessor `.tex` output file is then submitted to TEX. The document may then be viewed, printed, corrected for re-injection in the loop, admired, or thrashed as usual.

The **C** source for the preprocessor, the macro package, all the required METAFONT input files, and a demonstration input file to be taken as a Users Guide, are available free on the Tel-Aviv University IBM computer operated as a national inter-university computing center. To obtain a copy send an interactive message or a piece of mail to **LISTSERV AT TAUNIVM** with the string **GET IVRITEX PACKAGE** placed either in the interactive message or in the *body, not subject field,* of the mail. This service is managed by Mr. D. Sitman; users in trouble or in need of executable files (preprocessor, fonts) are kindly requested to write to me, not to him-because of compiler copyrights and of the great variety of output devices, these binary files cannot be made public.

Coming Soon

The updated version of the preprocessor and macro package will shortly replace the old one in the network server files.

Flexible, customized character translation will be added.

The existing REDIS Hebrew font is being improved, specially punctuation, numbers and other special characters.

The better BURKO Hebrew font will hopefully follow.

Missing glyphs in the Parsi and Arabic fonts will be created.

At this time, there are no Semitic language document organization macros. It is my hope that the availability of the basic tools will prompt users to write such macros, and needless to say to make them widely available.

L'Envoi

Whom should I explicitly thank for having made this work possible?

- Prof. Knuth for TeX and for METAFONT
- Prof. Knuth again for having made them universal and free
- Alan Spragens at the Stanford Linear Accelerator Centre, thanks to whom I took my first steps with TeX
- My dear four students who patiently made the fonts
- Ben Pashkoff at the Technion for installing TeX under VMS there
- Dean Guenther for his patient and efficient help in bringing up TeX and the environment on our VM machine
- Profs. K. Preiss and H. Harari who found some very modest funds to pay that part of the font design beyond the original student project assignment
- The Bitnet crew without whom that work would not have existed
- All those around me who refused to hear about TeX, to install it, and to care about extending it to Hebrew, which should have been their job, not mine, funding included: their determined opposition convinced me that I was right to do it.

I will conclude with a famous fragment by Maimonides included to show a piece of Hebrew text:

המלך המשיח עתיד לעמוד ולהחזיר מלכות דוד ליושנה לממשלה הראשונה. ובונה המקדש ומקבץ נדחי ישראל.... ובאותו הזמן לא יהיה שם לא רעב ולא מלחמה. ולא קנאה ולא תחרות. שהטובה תהיה מושפעת הרבה. וכל המעדנים מצויין כעפר. ולא יהיה עסק כל העולם אלא לדעת את השם בלבד. ולפיכך יהיו ישראל חכמים גדולים ויודעים דברים הסתומים וישיגו דעת בוראם כפי כח אדם. שנאמר כי מלאה הארץ דעת את השם כמים לים מכסים. (העתיק יעקב יוסף בן שלמה זאב זכרונו לברכה).

One final line to show that Arabic fonts are very important to us, and just about to come:

اليهود والعرب كلاهما بني ابراهام

Bibliography

[1] Knuth, D. and MacKay, P. Mixing right-to-left with left-to-right texts. *TUGboat* 8(1)14–25, 1987.

[2] Renan, E. *Histoire des langues sémitiques.* Paris: Calmann-Lévy, 1878.

[3] Toby, L.F. *The Art of Hebrew Lettering.* Tel Aviv: Schuster, 1987.

TEX is Multilingual

MICHAEL J. FERGUSON

INRS-Télécommunications
3 Place du Commerce
Verdun, Québec H3E 1H6
MIKE@tel.inrs.cdn

ABSTRACT

Although TEX is multiply unilingual, modifications to TEX were required to make it easily usable in a bilingual or multilingual environment. The requirements were both economic and conceptual. This paper discusses the modifications required to achieve an appropriately desirable behavior. Typesetting conventions tend to be linguistically and geographically idiosyncratic. Some of these differences will be discussed along with limitations in the current implementation.

Introduction

TEX appears to be much more comfortable being *multi-unilingual* than *multilingual.* Although it supports different languages, it seems to do so, gracefully, only one at a time. TEX became T$\hat{\rm E}$X in order to support, gracefully, several languages at a time. The problems of supporting document production, even in a unilingual environment occur at many levels. These include, but are not limited to:

- input conventions and methodologies
- language and special character sets
- internal textual coordination and consistency requirements
- external textual consistency conventions
- document design and "paper saving conventions"

This list concentrates only on those items that may be language dependent and purposely avoids all those issues involved with either multiple or exotic media. Examples of these range from voice annotation to the inclusion of graphic elements.

For unilingual environments, especially those that have an extensive printing history, there already are a set of defacto conventions for all of the above points. Unfortunately, they are usually (never?) codified and appear, at least to some outside observers, to have both semantic (syntactic, artistic, phonetic?) and

technological origins. The distinction is important because TEX and its mechanised support systems represent a new technology. This new technology may not easily duplicate the old technology but will bring new possibilities. There is hope that the new conventions will be more harmonious amongst various linguistic groups ... but past history suggests that this may not be so.

Multiple languages do not appear naturally in most documents. Exceptions, of course, are dictionaries, most appliance and machinery instruction manuals or product labels in Canada, or official documents of the European Parliament. The "base" language is defined here to mean the primary language of the document. In most, but not all cases, this primary language is easily identifiable. TEX was designed to happily accept the occasional non-base language word or phrase. The tacit assumption was that these inclusions would be typeset according to the base language conventions. It does not appear to be designed to handle large quantities of text in different languages. The primary economic difficulties (both monetary and temporal) involve:

- a need to have access to a language-specific set of fonts
- the storage of the fonts in all their desired forms
- the inability to automatically hyphenate (for paper saving and document design) any word that had a diacritic placed by TEX's `\accent` command

A number of these problems could be overcome by using METAFONT. Unfortunately, neither the temporal nor artistic ability are (were) in sufficiently large supply to make this feasible.

Most of the conceptual difficulties in the multiple language use of TEX are at the document design level. These include the following:

- there is only a single hyphenation (pattern) table resident at any time
- there may be too many diacritic combinations to include all possibilities in the font
- some language conventions change the spelling of fragments of split words and allow word splitting at other places
- some language conventions have semantically dependent ligatures
- initial and terminating fragments of a word are fixed to be 2 and 3 characters, respectively
- the maximum number of fonts within TEX is limited
- TEX's internal working memory may limit the number of simultaneously stored, pages resulting in coordination problems
- TEX's use of a paragraph as a fundamental unit of computation may prevent word-dependent processing

One of the difficulties of determining the real multilingual conceptual difficulties in the TEX architecture is that the "required" multi-unilingual requirements have not been codified. A codification will probably occur over many years. Hopefully the extensions and system described here, coincidently with the introduction of new technology, will unify the process.

If the document designer never allowed word splitting, then the conceptual difficulty of only one hyphenation table would disappear. It is unlikely that this will happen. If word splitting is allowed, then textual element coordination, as shown in two-column English/French example in a subsequent section, requires that a pattern table for each language be resident. Some of the other conceptual difficulties listed above are really semi-economic. The limitation on the number of fonts and number of characters in a font would disappear with bytes longer than 8 bits. Unfortunately, the character input problem remains. It is easier to specify a two-byte or three-byte sequence than one of 2^{16} or 2^{24} characters. Although T̂EX provides mechanisms to handle all of the above conceptual difficulties, it does not do this with all the automatic grace that one could desire.

TEX Strategies for Multiple Languages

TEX appears to have been designed with the intent of using a different font for each different, commonly used, language. METAFONT was conceived as the primary tool to support this view. "Unusual" problems, such as diacritics and spelling changes at splitting boundaries, were handled by the TEX primitives `\accent` and `\discretionary`. Unfortunately the use of either method precluded the further splitting of a word. Although an infinite supply of language-dependent fonts might yet appear, the cost of acquisition and storage along with potential document interchange problems do not necessarily make this solution desirable.

T̂EX Multiple Language Support

T̂EX comes in two basic varieties. The first, T̂EX, operates on a paragraph-by-paragraph basis and is subject to all the limitations that normal TEX has because of this mode. The second, T̂EX-W, is able to change languages on a word-by-word basis, and simultaneously allows for an enormous increase in TEX's internal memory because it uses 32-bit rather than 16-bit pointers. Both T̂EX and T̂EX-W use exactly the same font information files, and do all their arithmetic using 32-bit integers. Furthermore they retain all the kerning that is retained in the original TEX. Finally, T̂EX passes the "trip" test while T̂EX-W essentially passes the "trip" test. The word "essentially" is used because T̂EX-W writes out the `\language` value for each letter, along with the font information. In the "trip" test, this is always `lang=0`. This is the only difference in the "trip" output. Although T̂EX-W does not see much use in our environment, it is probably necessary for dictionaries and perhaps for the multiple language case of the European Parliament.

The extensions incorporated in TEX and TEX-W allow for the following:

- words with diacritics are allowed in the hyphenation patterns and exceptions.
- multiple independent hyphenation pattern and exception sets are allowed and accessed by a change in value of a new primitive `\language`
- the input character set is extended to about 250 by making all of the internal character codes above 127 permanently active
- different languages may use the same fonts
- words with diacritics on capital letters, such as CONSIDÉRIONS, will be correctly hyphenated
- words with a `\discretionary` may be, by changing the new primitive, `\dischyph` to be non-zero, optionally hyphenated
- the length of the initial and terminal fragments may be explicitly defined

Although these modifications do not automatically solve the context-dependent ligature problem, or the spelling modification problem, they do give the tools to do so.

As of 1988, both TEX and TEX-W have been in use at INRS-Télécommunications on VAX/VMS systems, without major problems, for about three years. With the recent (1987/1988) port to the IBM-PC/(MS-DOS) by Personal TEX, experience is rapidly being gained with other European languages. In addition TEX and TEX-W have been ported to the SUN and converted to C by Justin Bur of Université de Montréal, and IBM-CMS by Dean Guenther of the University of Washington. François Chahuneau of Berger-Levrault in Paris is coordinating distribution of the non IBM-PC/(MS-DOS) versions in Europe. TEX and TEX-W are being ported to several other systems.

An Example of English/French Side by Side

This section shows an example of two columns of text, coordinated on a paragraph-by-paragraph basis, with "long" paragraphs split at the bottom of a page. The left column is hyphenated according to English rules and the right by French rules. The text is taken from the "Awards Guide" of the *Natural Sciences and Engineering Research Council of Canada – NSERC*. Note that the text is split at the bottom of the page and continued on the next.

Eligibility and Application Procedures

56 The general eligibility conditions outlined in paragraphs 10 to 12 must be satisfied by the princi-

Admissibilité et modalité de demande

56 Les conditions générales d'admissibilité énoncées aux articles 10 à 12 s'appliquent dans le cas du chef

pal investigator and all other academic researchers. Scientists and engineers from industrial and government sectors who play an active role in the collaborative project may apply as co-investigators in order to stimulate programs of broad interest. These grants may be held currently with other Council grants.

de groupe et de tous les autres participants universitaires. Les scientifiques et ingénieurs des secteurs gouvernemental et industriel qui jouent un rôle important dans la réalisation du projet collectif sont admissibles comme membres du groupe afin d'encourager les programmes d'envergure. Les subventions à des projets collectifs spéciaux peuvent être détenues en même temps que d'autres subventions du Conseil.

57 Before submitting a format application, researchers should forward a letter of intent that includes the following information:

- the nature of the special opportunity;
- the names of the participating researchers and their areas of expertise;
- the significance of the research opportunity;
- an outline of the proposed research;
- a preliminary budget;
- the proposed time-frame of the project.

57 Avant de présenter officiellement une demande de subvention, les chercheurs devraient envoyer une lettre d'intention au CRSNG contenant les renseignements suivants:

- la nature de l'occasion spéciale;
- le nom des participants et leur champs de compétence;
- l'importance de l'occasion de recherche qui se présente;
- un aperçu de la recherche proposée;
- un budget préliminaire;
- le calendrier prévu.

The actual text input for this example is

```
\input enfrtwo_tug
\normalbaselines
\artnum=55
\autonumberingon
\dsh{Eligibility and Application Procedures}{Admissibilit\'e
et modalit\'e de demande}
\pp %
\pn
 The general eligibility  conditions outlined in paragraphs
 10 to 12 must be satisfied by the principal investigator
```

```
 and all other academic researchers. Scientists and engineers
 from industrial and government sectors who play an active
 role in the collaborative project may apply as co-investigators
 in order to stimulate programs of broad interest. These
 grants may be held currently with other Council grants.
@@
\rpn
Les conditions g\'en\'erales d'admissibilit\'e \'enonc\'ees
aux articles10 \`a 12 s'appliquent dans le cas du
 chef de groupe et de tous les autres participants universitaires.
Les scientifiques et ing\'enieurs des secteurs gouvernemental
 et industriel qui jouent un r\^ole important dans la
r\'ealisation du projet collectif sont admissibles comme membres
du groupe afin d'encourager les programmes d'envergure. Les
subventions \`a des projets collectifs sp\'eciaux
peuvent \^etre d\'etenues en m\^eme temps que
d'autres subventions du Conseil.
!!


\pp
\pn
Before submitting a format application, researchers should forward
a letter of intent that includes the following information:
\beginlist
\li - the nature of the special opportunity;
\li - the names of the participating researchers and their areas
       of expertise;
\li - the significance of the research opportunity;
\li - an outline of the proposed research;
\li - a preliminary budget;
\li - the proposed time-frame of the project.
\endlist
@@
\rpn
Avant de pr\'esenter officiellement une demande de subvention,
les chercheurs devraient envoyer une lettre d'intention
au CRSNG contenant les renseignements suivants:
\bl
\li  - la nature de l'occasion sp\'eciale;
\li  - le nom des participants et leur champs de
       comp\'etence;
\li  - l'importance de l'occasion de recherche qui se
          pr\'esente;
\li  - un aper\c cu de la recherche propos\'ee;
```

```
\li   - un budget pr\'eliminaire;
\li   - le calendrier pr\'evu.
\el
!!
```

Variations on the same production theme could involve outputting of text in different languages on pages with the same number In this case the text coordination is induced through the requirement that an entire section appear on the same numbered page in all language editions. Interestingly enough, although the requirements appear to be similar to the multi-column coordination above, the constraint that the maximum of a sum of section lengths appear on the same page induces a different macro architecture.

TEX and TEX-W Modifications to TEX

The intent with the modifications to TEX was to solve both the economic and conceptual problems and still maintain the efficiency and elegance of the original TEX. Recent extensions have been a result of the perceived needs of European languages other than French and English. The full extent of these needs is not yet known. This section describes the changes, introduced via a WEB change file, to the original TEX program.

1 Language-Dependent Patterns and Exceptions

The key conceptual modification to TEX was the allowing of language dependent hyphenation patterns and exceptions. The correct set of patterns/exceptions are chosen via the value of the new primitive `\language`. Details of the input conventions for patterns can be found in the references cited in the bibliography. When an exception is entered, it becomes an exception only for the current language.

We recently had a need to produce a document in Polish. The writer found that the French patterns did a much better job than the English patterns but that there was a tendency to split, incorrectly, "sz" and "rz". Inhibition of these patterns was added to the French patterns and a tri-lingual format file was made. The increase in pattern table size was negligible. The table compression scheme in TEX is wonderfully efficient. This, of course, is not a substitute for producing a specially tailored set of patterns and exceptions for a language but does indicate the power of the system.

2 Hyphenation of Words with Diacritics Produced by `\accent`

TEX hyphenates words by sending a string of characters that it thinks may be a complete word to the hyphenation procedure. To do this it removes all implicit kerning and ligatures and reconstitutes them afterwards while inserting the appropriate discretionary nodes. Since it cannot reconstitute anything other

than implicit kerning and ligatures in the same font. it refuses to allow any text fragment that does not satisfy these conditions to be a potentially hyphenatable word. T$\hat{\rm E}$X and T$\hat{\rm E}$X-W extend the reconstitution possibilities by explicitly identifying those characters that are accents and, recently, those `hlist` nodes that were produced by the `\accent` primitive. Accents or diacritics are identified by having an `\lccode` of 1 or 2. It is assumed that the character following the diacritic is the character that is supposed to be accented. Since the `subtype` of an `hlist` node was not used, the `\accent` processing now sets the `subtype` to the value of **A**. Non-accent `hlist` nodes have subtype value of 0. This allows identification of the work done by the `\accent` processing to be reconstituted and hence allows for hyphenation of words that include accented capital letters ... all of which require that TEX raise the accent. The restrictions with this procedure are as follow:

- both accents and accented letter must be in the same font
- only single accents are allowed
- it is assumed that the `hlist` node has only one character in it
- explicit accent placement modifications will prevent hyphenation

TEX now requires that all characters in a word to be hyphenated to be in the same font. This is necessary to allow the kern and ligature reconstitution to be successful.

3 Hyphenation of Words with Discretionaries

Both TEXs allow for the optional hyphenation of words that already include discretionary hyphens. This is invoked by setting the new primitive `\dischyph` to a non-zero value. It is in force for an entire paragraph. The fragment that starts and/or finishes with a discretionary node is hyphenated as if it were an entire word. This option is useful, perhaps, in pure English text since a hyphen "-" in a word automatically inserts a discretionary node. Note that there is a semantic difference between the hyphen in a compound word and the one that is used for word splitting at the end of line. This semantic clash appears to be one of the reasons that TEX does not allow this type of word splitting.

4 Start and Terminate Fragment Lengths

Two new integer primitives `\starthyph` and `\stophyph` have been introduced. The values of `\starthyph` and `\stophyph` are the minimum number of characters allowed in the initial and terminating word fragment respectively. These may be set to any positive value. The defaults are the same as in TEX, namely 2 and 3.

5 Input Character Codes

Both T̂EXs allow for the specification of input characters that map to internal TEX codes that are greater than 128. All characters so mapped are permanently declared active. This means that a sequence may be defined to represent that input character. If a sequence has not been defined, the same "invalid character" error message that TEX emits occurs. The single character on the DEC VT-200 series of terminals for the "é" has an octal code of 351. This character is given a definition of `\'e`. Whenever it is input, it is immediately stored internally in T̂EX as its sequence. This allows for more complex possibilities. Suppose the sequence "ck" was a single active character. It could be given the definition "`\discretionary {k}{k}{ck}`". This also allows for a method to handle context-dependent ligatures. However, neither of these methods are automatic. Automatic solutions to the context dependent ligature problem are probably processor intensive—any ligature is potentially invalid. It is not yet clear whether an automatic method for discretionary spelling and context-dependent ligatures is a universal mechanism for a wide range of languages.

Conclusions and Hopes

Although T̂EX and T̂EX-W have increased dramatically the applicability and usefulness of TEX, they do not have all the automatic grace or capabilities that could be desired. Some of the obvious defects are as follows:

- discretionary spelling is not automatic
- context dependent ligature determination is not automatic
- accent positioning cannot be modified without preventing hyphenattion
- multiple accents are not allowed
- explicit "manual" kerning prevents hyphenation

Unfortunately, it is still not clear whether the solution to these defects is either necessary or sufficient for even semi-universal multilinguistic applicability of TEX. An interesting new problem arises in Europe if Turkey is admitted into the European Community.

Bibliography

M.J. Ferguson, A Multilingual T̂EX, Pp 65–74 in *Proc. of TEX for Scientific Documentation, Strasbourg, June 1986,* Jacques Désarménien, Berlin, Springer-Verlag, 1986.

M.J. Ferguson, A Multilingual T̂EX, *Rapport technique*, 87–23, INRS-Télécommunications, May 1987.

M. Spivak, *Multilingual T̂EX Manual*, Mill Valley, Calif. Personal TEX, 1988.

Experiences with TEX in Finland

KAUKO SAARINEN

Computing Centre
University of Jyväskylä
Seminaarinkatu 15
40100 Jyväskylä
Finland

EARN/BITNET: Saarinen@FINJYU
INTERNET: Saarinen@JYLK.JYU.FI

ABSTRACT

TEX does not hyphenate words if there are accents, which presents problems in Finland where the three national languages all require diacritics. Multilingual TÊX and other methods to solve problems due to national languages will be discussed. An overview of experiences with TEX at the University of Jyväskylä is also presented.

The National Languages of Finland

There are three national languages spoken in Finland: Finnish, Swedish, and Lappish. The Swedish-speaking minority is about 6% of the total population of 5 million inhabitants. In the very north, beyond the Arctic Cirle, there is a small number of inhabitants speaking Lappish. The vast majority of the country speaks Finnish. A common feature of the national languages is frequent use of accents. Especially the characters **ä ö å Ä Ö Å** (**å Å** in Swedish only) are heavily used. In fact, when a Finn sees a national character, such as **ä**, no accent is seen. For any Finn **ä** is a single letter and its meaning and pronounciation is different from **a**. For instance, the Finnish words *saari* and *sääri* have the meanings 'island' and 'leg', respectively. It is clear that every word processing system in Finland must be able to handle national characters correctly.

1 Problems and Solutions in Using TEX in Finland

Most problems using standard TEX are due to the national characters with diacritics, which are frequently used as stated above. These characters can be easily produced by TEX using accents. However, there is a problem with hyphenation. The standard TEX does not hyphenate words containing the characters above if they are made using accents.

There is still another problem due to national characters. Using a seven-bit ASCII character set, the characters ä ö å Ä Ö Å normally replace characters { | } [\ ] on Finnish and Swedish keyboards. For this reason, TeX control characters are usually changed to / < > instead of \ { }. The change is easily done for `plain` TeX and for AMS-TeX as well, but there are severe problems with LaTeX, where \{, \|, \}, \[, \\, and \] are reserved combinations. Use of the LaTeX package has been fairly limited so far due to these difficulties.

1.1 Attempts to solve the hyphenation problem

The hyphenation problem has been solved at the University of Jyväskylä and at the Swedish University of Turku by making special fonts which contain extra characters. The method is a difficult one as device drivers must be modified also, but it works fine. However, this method is dangerous if TeX input files, and especially DVI files, are transferred from one site to another. Another possibility to solve the hyphenation problem is to add all possible discretionary hyphens \- to TeX input files using a preprocessor. This method is working well, but it requires extra effort and processing time. Still another possibility is to modify the TeX source programme. Finnish hyphenation rules are then added to TeX somewhere. The source programme of TeX must be available to do the job and this is not possible if you are using the commercial PC TeX, for example.

1.2 The Multilingual TêX solution

The Multilingual version of TeX, or TêX, has been in use on PCs since the beginning of 1988. The very first tests of TêX were surprising. As a test case, TêX and LaTeX with four language patterns—Finnish, Swedish, English and French—were compiled into a single `.fmt` file in order to see how TêX would work. Then the LaTeX local guide of some 30 pages was processed on a PC with 640 KB of memory. The result was a success!

TêX contains many useful features compared to TeX. The most important features are language-specific patterns, switching from one language to another, and the fact that words containing accents can be hyphenated.

New useful commands have also been added, such as `\dischyph`, `\stophyph` etc. for controlling hyphenation. TeX requires that there be at least 3 letters in the last syllable of a word in order to hyphenate it. Using the command `\stophyph=2` of TêX this value can be changed to 2, which is suitable for Finnish. In Finnish, the last syllable of a word often consists of 2 letters. Another useful setting is `\dischyp=1`. If you write a discretionary hyphen \- in a word processed by standard TeX, the word will not be hyphenated elsewhere. If you wish to allow hyphenation in every legal place of a word you must explicitly add all discretionary hyphens, which can be frustrating. Giving a value of 1 to the previous parameter, TêX can hyphenate words elsewhere also. So, you may give hints for hyphenation if there is a difficult part in a word. This feature is very

useful, at least in Finnish. Hints for hyphenation are most often needed at the word boundaries of compound words.

There still remains a problem common to both versions of TEX. The problem is the position of umlaut accents. A local typesetting expert said that there is no strict rule as to where the umlaut accents should be positioned. After looking at output from TEX he added that in general, there seems to be too much space between a letter and an umlaut compared to the general convention in Finland. The result looks worse when higher magnification is used. A study of old newspapers revealed that umlauts were positioned higher in the past. Of late, they have been moved to a lower position—it's a pity for TEX!

It is said in TEX manuals that kerning is disabled when using accents. Local users of TEX seem to be satisfied with the kerning of TEX, as was the typesetting expert.

In order to hyphenate any language, hyphenation patterns are needed. Existing patterns must be found somehow and if you are using an exotic language, say Finnish, they must perhaps be written locally.

It is essential that T̂EX is based on standard fonts, which is important for compatibility and for transferring DVI files. If there are the same CM fonts on PCs and on the VAX/VMS mainframe, DVI files can then be transferred from a PC to a VAX. The transferred files can then be output further on laser printers run by the mainframe. If transferring is done with Kermit, as in Jyväskylä, DVI files must be slightly modified before processing them by a device driver of a laser printer.

An example is given below to demonstrate the superiority of T̂EX in processing Finnish. At first there is an input file processed by PC TEX. Only a partial listing of the output log file is given containing all the `Overfull...` warnings. Then the same text is processed by the Multilingual version of PC TEX.

An example of differences between TEX and T̂EX

```
This is TeX, Version 2.0 (PCTeX 1.50, (c)Personal TeX, Inc 1986)
(preloaded format=plain 88.6.11) 11 JUN 1988 09:34
**&plain test1
(test1.tex (\pctex\texinput (\pctex\texinput\twelve.tex)
Overfull \hbox (8.4722pt too wide) in paragraph at lines 33--43
Overfull \hbox (6.8662pt too wide) in paragraph at lines 33--43
Overfull \hbox (7.51944pt too wide) in paragraph at lines 60--67
[1]
Overfull \hbox (5.0932pt too wide) in paragraph at lines 84--93
Overfull \hbox (25.99554pt too wide) in paragraph at lines 111--124
Overfull \hbox (7.04027pt too wide) in paragraph at lines 135--144
Overfull \hbox (24.28075pt too wide) in paragraph at lines 145--151
[2]
```

```
Overfull \hbox (17.91219pt too wide) in paragraph at lines 179--189
[3]
Output written on test1.dvi (3 pages, 10300 bytes).
This is TeX, Version 2.1 (preloaded format=mlplain 88.2.29)
11 JUN 1988 09:31
(Multi-Lingual PCTeX 2.10, (c)Personal TeX, Inc 1987.  S/N 20158)
**&mlplain test2
(C:\OMA\TEST2.TEX (C:\PCTEX\TEXINPUT\JYLKPC.TEX
-Conv. JYLK-PC complete-) (C:\PCTEX\TEXINPUT\TWELVE.TEX) [1] [2] [3]
Output written on C:\OMA\TEST2.DVI (3 pages, 10396 bytes).
```

Voilà! Regardez l'exemple s'il vous plaît! C'est si bon avec TÊX!

2 Experiences with TEX at the University of Jyväskylä

After elaborating on the problems due to umlaut accents, an overview of other TEX-related affairs at the University of Jyväskylä is presented. TEX history, user profiles, previewing of DVI files on the screen and PC TEX distribution, among other things, are discussed.

2.1 TEX history at the University of Jyväskylä

TEX has been in use for quite a long time at Jyväskylä. The main points are listed in chronological order.

- The first version TEX78 of TEX was running on UNIVAC 1100 in 1983. A wheel printer was used as an output device. There was no serious use due to the simple output device.
- TEX has been in use on VAX/VMS since the end of 1985. The output device was a single QMS LASERGRAFIX 800 laser printer.
- The first TEX course was held in February 1986 with about 20 participants.
- The first PC TEX was bought in the autumn 1986 and the Campus site license agreement was signed 1987.
- Multilingual PC TEX has been in use since the beginning of 1988.

2.2 User profiles

A typical user of the AMS-TEX package is a mathematician. Our mathematicians tried other text processing programmes before. However, the use of these programmes was soon history after getting TEX. At the Faculties of Mathematics and Statistics, secretaries also use TEX. However, there are many other users also, for example, at the Faculty of Humanities. Students naturally are eager to use TEX a lot. I suspect sometimes that they believe that the impressive output of their papers will guarantee a better response from their teachers. Users other than mathematicians have generally used `plain` TEX so far.

Different languages have been processed. At least the following ones have been processed and probably in the listed order: Finnish, English, Swedish, French, German, Spanish, and Russian. There are patterns available for each of these except for Spanish and Russian at the moment. Russian has been used by a couple of users. The cyrillic fonts and macros for using them were bought from the American Mathematical Society. The Finnish hyphenation patterns written locally are fairly concise and straightforward compared to many other languages. They are by no means perfect but seem to work fairly well for practical purposes. If someone needs the Finnish patterns, I'm ready to send them to anyone using electronic mail.

Using Multilingual PC TEX and the extended ASCII character set, the limitations concerning LATEX are removed and, as a result, the use of LATEX is as easy for Finnish as for English. It is expected that use of LATEX will grow.

2.3 Previewing DVI files on PCs

A very user friendly feature when using TEX on the PC is the possibility of viewing DVI files on the screen. There are two different viewing programmes available at the University of Jyväskylä. The commercial MAXview is used and the Campus license agreement is signed for it. MAXview is a flexible programme, because it can use any fonts installed for your printer, and does not need any extra fonts.

The other viewing programme has been made in co-operation between the Universities of Jyväskylä and Helsinki. The locally written viewing programme contains better zooming possibilites than those of MAXview but special fonts are needed. The reason for making their own viewing programme was, originally, that there was a PC with a nice graphics card, but no commercial programme available for this particular graphics option.

2.4 Hints for distribution of PC TEX

Installation of PC TEX can be quite difficult for an end user if installation must be done using the original diskettes, which have been bought commercially. There are choices of different output devices, graphics etc. It is more user friendly to make a basic installation first and then distribute it for end users.

Due to little experience with the MS-DOS system, an attempt to distribute PC TEX using MS-DOS commands such as BACKUP and RESTORE was tried. These commands are easy to do but there are problems with different levels of MS-DOS and also with different machines of the same level of MS-DOS. This method simply does not work in practise. A better way to distribute PC TEX diskettes is to make the basic installation first and then make a batch queue that copies all the needed files from diskettes onto the hard disk of PC. This method works fine and updates can be made easily.

2.5 Mixed views of interest

Jyväskylä is a nice small town with about 70 000 inhabitants and 270 kilometers north of Helsinki. The town is surrounded by lakes and forests and in the middle of the city there is a large hill. The very beautiful University campus is located at the west end of the hill. There are 6000 students at the University of Jyväskylä. TEX is being run on a VAX/VMS mainframe and on PCs. TEX will be installed on the new Sun mainframe computer also. Laser printers with a resolution of 300 dots per inch have been used as output devices so far. The well-known TEX manuals are used and there is a local TEX manual in Finnish of about 70 pages. At the moment, two enterprising students are writing manuals for PC TEX and LATEX in Finnish as summer work.

Summary

TEX is widely used at many universities throughout Finland, even though it is not so easy to learn. However, it's often surprising to see how rapidly people can learn to use TEX when they have enough motivation and that motivation is—the professional-looking output! In practise there is not a good alternative for TEX at the moment for writing professional-looking mathematics.

And last, but not least it's nice to point out that this article was prepared using TEX on a PC. The program coordinator for this year's Annual General Meeting, Dean Guenther, sent TEX macros of the proceedings to me using e-mail. After preparing the article, it was sent back using e-mail again—how small our planet really is today!

Acknowledgement

Some details of the article are based on discussions with Dr. Ari Lehtonen, who is an advanced TEX user himself, working in the Department of Mathematics at the University of Jyväskylä.

Bibliography

Knuth, Donald E. *The TEXbook.* Reading, Mass.: Addison-Wesley. 1984.

Lamport, Leslie. *LATEX—A Document Preparation System.* Reading, Mass.: Addison-Wesley. 1986.

Spivak, Michael. *PC TEX Manual.* Mill Valley, Calif.: Personal TEX. 1985.

Spivak, Michael. *The Joy of TEX.* Providence, RI.: American Mathematical Society. 1986.

Spivak, Michael. *Multilingual TEX Manual.* Mill Valley, Calif.: Personal TEX. 1987.

Using the Emacs Editor to Safely Edit TEX Sources

STEPHAN V. BECHTOLSHEIM

Integrated Computer Software, Inc.
2119 Old Oak Drive
West Lafayette, IN 47906
(317) 463 0162

ABSTRACT

In this paper we will discuss the use of a programmable editor like Emacs to safely edit TEX sources. We will argue that by using this editor it is possible to dramatically reduce the number of errors made entering TEX codes together with text in a file. It is possible this way to achieve a comfort in editing TEX sources which approaches the quality of WYSIWYG systems without inheriting their limitations.

Introduction

The title of this paper has changed somewhat compared to the one listed in the program because I decided that focussing on the editing of TEX sources with Emacs was enough of a subject to deal with. Lynn Price will discuss the issue of translating SGML into TEX etc.

This paper tries to explain the advantages of using the Emacs editor to edit TEX sources. For those TEX users who know this editor this paper will probably not offer anything new. On the other hand, for TEX users who are not programming wizards and who would like to learn something new, hopefully this paper will show that using a sophisticated editor such as Emacs does help enormously when entering and correcting TEX sources.

I would now like to discuss some general principles which are important for the later discussion of a TEX mode for the Emacs editor and the relationship between TEX and WYSIWYG systems.

1. *WYSIWYG systems have problems inherent to their design*: the main problem is the generation of consistent document layouts. These systems are getting better; in general, though, programmable systems like TEX are more powerful and versatile.
2. *WYSIWYG systems look flashy but they cannot solve the problem of writing either*: they may look as flashy as a James Bond movie, but to write in an organized and well-structured way is something they can't solve either.
3. *Other than deciding what typesetting system to use, choosing a text editor*

is the most important decision you make when generating documents: if it takes one hour to generate a document with TEX (or with any typesetting system for that matter), then observe that you will spend about 55 minutes of this hour in the editor changing the text. The remaining 5 minutes are used to execute TEX and print your document. Therefore, if only for this timing reason, it is extremely important that you use a flexible and powerful editor which supports editing TEX sources.

4. *Prevention is better than correction*: if you can prevent a heart attack in the first place you save yourself much more than if you have one and have to go through all types of surgery, etc. So if you can prevent an error in TEX you do yourself a big favor.

5. *The trivial and minor errors are the most frequently occurring and the most annoying ones*: if you could avoid only the "silly errors" like omitting closing curly braces then this would already help you a lot, more than anything else. Therefore, we will discuss methods to prevent silly errors *when the text is entered into the computer.*

6. *Some of TEX's instructions are used much more frequently than others*: for instance, curly braces occur much more frequently than `\relax`. Therefore it is worth thinking about the more frequently occurring control sequences more than about those which are used less frequently.

Considering all the reasons just listed should convince you that *using the best available editor* is simply a very smart idea.

The Emacs Editor

Let us now discuss the Emacs editor. We will specifically refer to the GNU Emacs editor. At the end of this paper you will find information of how to get this editor, etc. Let me first discuss the LISP interpretor built into this editor. There are two areas of this interpretor to be looked at:

1. Because of this LISP interpretor you can write regular LISP programs in this editor which have nothing to do with editing text. Let us give some very simple LISP programming examples here:

```
; variable: number of apples.
(defvar apples 0 "Number of apples")

(setq apples (+ apples 1))  ; Increment it.

; Define a LISP function to add apples.
(defun Increment-Number-Of-Apples (number)
   "Add NUMBER to the current number of apples."
   (setq apples (+ apples number))
)
```

```
; Call previously defined LISP function.
(Increment-Number-Of-Apples 5)
```

2. This LISP interpretor is augmented by various editing primitives. So one can actually write programs which edit text as we will see in the examples below:

```
(forward-char 5)              ; Go five characters forward.
(backward-char 5)             ; Go five characters backward.
(forward-char -5)             ; Same.
(insert-string                ; Insert the given string.
    (format "Letter mailed at %s\n" (date)))
(goto-char (point-min))       ; Go to the beginning
                              ; of the buffer.
(goto-char (point-max))       ; Go to the end of the buffer.
(find-file "xx.tex")          ; Find specified file for editing
(set-buffer "xx.tex")         ; Switch to specified buffer
(recenter)                    ; Redraw screen
(delete-other-windows)        ; Make current window the only
                              ; one on screen.
(split-window-vertically)     ; Split current window in two
                              ; windows
(other-window)                ; Select the next window
```

Observe that so far we have not discussed editing in the traditional sense. We have only explained how a user could write editing programs, programs which change text.

Some LISP Functions of an Emacs TEX Mode

We will now present some simple LISP functions as they would occur if part of a TEX mode for the GNU Emacs editor. The following discussion will serve two purposes:

1. We will discuss methods in general to improve the safety of editing TEX sources.

2. We will discuss how our ideas can be implemented using the Emacs editor.

Curly Braces

One of the easiest ways to avoid problems with forgetting to enter closing curly braces is to enter curly braces **always** in pairs. In other words, when you enter an opening curly brace you enter the closing one right away and then you position the cursor between the two curly braces. Here is a LISP function to do exactly that:

```
(defun TeX-Curly-Braces
    (insert-string "{}")
    (forward-char -1))
```

The next trick is to invoke this function automatically when an opening curly brace is entered. This is done with the following instruction:

```
(define-key TeX-mode-map "{"        'TeX-Curly-Braces)
```

From now on, entering an opening curly brace on the keyboard will call the function `TeX-Curly-Braces`. Typing an opening curly brace will no more just insert an opening curly brace into the text. The `define-key` LISP instruction makes this link for the opening curly brace only effective when TEX sources are being edited.

A similar type of construction can be applied to parentheses and square brackets. It is my personal experience that once you install this feature it is something you won't want to be without.

Superscripts and Subscripts

For superscripts and subscripts the automatic insertion of curly braces can be enforced too. Here is how this is done for superscripts:

```
(defun TeX-Superscript
    (insert-string "^{}")
    (forward-char -1))

; "^" causes "^{}" to be inserted.
(define-key TeX-mode-map "^"        'TeX-Superscript)
```

Dollar Sign

It is natural to use the same type of treatment for the dollar sign. This time though we will be a little more sophisticated: a dollar sign anywhere in the line is taken as an instruction by the user to enter an inline math mode equation—two dollar signs are entered and the cursor is positioned between the two. So far this is exactly along the lines of the treatment of an opening curly brace. Now we will deviate slightly: this doubling of the dollar sign does not happen if the dollar sign is preceded by a backslash, because this is the usual way in TEX to print a dollar sign. Furthermore, a dollar sign entered at the beginning of a line is interpreted as the start of a mathematical equation in display math mode. In this case two double dollar signs have to be entered with an empty line between the two double dollar sign groups. Here is a little LISP program implementing the above ideas:

```
(defun TeX-dollar ()
   "$ handling in TeX: $ generates $$ unless escaped (\$).
   If at the beginning of a line enter display math mode."
   (interactive)
   (if (char-equal (preceding-char) ?\\ )
       (insert-string "$")
     (if (bolp)
         (progn
           (insert "$$\n\t\n$$\n")
           (forward-line -2)
           (end-of-line))
       (insert-string "$$")
       (forward-char -1))))

; Bind the above function to $.
(define-key TeX-mode-map "$"        'TeX-dollar)
```

Here is another variant of this same LISP function you could use in LATEX where the environments `math` and `displaymath` can be used for inline and display math mode:

```
(defun TeX-dollar ()
    (interactive)
    (if (char-equal (preceding-char) ?\\ )
        (insert-string "$")
      (if (bolp)
        (progn
          (insert "\\begin{displaymath}\n\t"
          (insert "\n\\end{displaymath}\n")
          (forward-line -2)
          (end-of-line))
        (insert-string "\\begin{math}  \\end{math}")
        (forward-char -11))))
```

You see that it is very easy to reprogram the editor—just pick your favorite version of `TeX-dollar` to generate a TEX source according to your style.

Font Change Functions

Because font changes occur very frequently and one of the things we wanted to do was to help input frequently occurring TEX instructions. Let us see what we can do in the case of font changes. Here are three little functions which are bound to `ESC ^B` for boldface, `ESC ^I` for italics and `ESC ^T` for typewriter font. A font change can now be entered very quickly:

```
(defun TeX-boldface ()
    "Boldface text."
    (interactive)
    (insert-string "{\\bf }")
    (backward-char 1))

(defun TeX-italics ()
   "Italics text."
   (interactive)
   (insert-string "{\\it \\/}")
   (backward-char 3))

(defun TeX-typewriter ()
   "Teletype style."
   (interactive)
   (insert-string "{\\tt }")
   (backward-char 1))

(define-key TeX-mode-map "\e\^b"   'TeX-boldface)
(define-key TeX-mode-map "\e\^i"   'TeX-italics)
(define-key TeX-mode-map "\e\^t"   'TeX-typewriter)
```

Automating the Handling of the Italic Correction

Observe that in the above example the italic correction is always inserted by `TeX-italics` regardless of whether it is needed or not. There should be no italic correction in the case of a period or comma following the closing curly brace of a group terminating a font change to italics. So we will now provide a little LISP function which, if necessary, removes the italic correction from the text:

```
(defun TeX-period-comma ()
   "Take out italic corrections before periods and commas."
   (interactive)
   (save-excursion
     (forward-char -3)
     (if (looking-at "\\\\/}")
         (delete-char 2)))    ; deletes \/
   (insert-char last-input-char 1))    ; inserts '.' or ','

(define-key TeX-mode-map "."        'TeX-period-comma)
(define-key TeX-mode-map ","        'TeX-period-comma)
```

More Sophisticated Ideas

The previous section showed one of the simplest examples for LISP functions in

a TEX mode I could think off. Now let us discuss what else can be done. We will here elaborate on one idea and that is the idea of supporting inputting environments of LATEX into documents. Here are the functions our LISP procedure will perform:

1. `\begin{...}` and `\end{...}` are entered in pairs followed by positioning the cursor between the two constructs. One of the nice side effects is that this way the nesting of environments is always correct.
2. The spelling of the environment's name is checked. If an illegal environment name is entered it is rejected. There will be no illegal environment name when the document is being processed.
3. For certain environments like `enumerate` the LISP function might also generate a couple of `\items` properly indented. So your screen, after having told Emacs to enter an `enumerate` environment, looks as follows:

```
\begin{enumerate}
    \item
    \item
    \item
\end{enumerate}
```

4. One can easily define a LISP function `Insert-Item` which will insert a new `\item` properly indented on a line by itself.
5. Environment name completion can be used: if a partially determined environment name (for example, `e` is sufficient for the `enumerate` environment because no other environment starts with `e`) then the editor will insert the complete `enumerate`.

How to Proceed From Here

Assuming I have been able to convince you that using the GNU Emacs editor is indeed a good idea, let us now discuss how you proceed from here. This editor runs on a variety of machines, but it does not run on PCs or on Macintoshes. These machines are simply too small. There are Emacs implementations which run on PCs, but what I have seen so far they don't deserve the name Emacs: they are far less general, and of course part of the limitations are limitations imposed by MS-DOS.

Also observe that while there are other Emacs versions around, GNU Emacs from the Free Software Foundation is by far the best and most reliable version of this editor.

The GNU Emacs software can be picked up via `ftp` from `prep.ai.mit.edu`. It is also possible to order a tape (1/2") or a SUN 1/4" cartridge by writing to the following address: Free Software Foundation, 675 Mass Avenue, Cambridge, MA 02139. The cost is around $150.00. The editor is freely redistributable; that

is, it is illegal to charge any fees to redistribute the editor. Actually you are encouraged to pass the editor on to other people.

This editor comes with a TEX mode, but Nelson Beebe from the University of Utah has developed a TEX mode which is probably much better worked out than any other TEX mode for this editor. He is attending this conference and he may give his opinion about the state of the various TEX modes for GNU Emacs.

Finally a warning at the end: Emacs is a very powerful editor, but it is also a rather complex piece of software. It requires maintenance as does any large program; it requires training and support. So if you decide to show non-programmers this editor you must know that it will take time and patience on your part to train people using this editor.

Conclusion

In this paper we have tried to explain that with the help of a powerful editor such as Emacs, it is possible to preserve the flexibility of a programmable system while editing TEX sources very conveniently. It is possible to dramatically reduce the number of errors generated. It is also possible to implement functions which are normally only available in a WYSIWYG system.

Using SGML and TEX for User Documentation

LYNNE A. PRICE

Hewlett-Packard Co.
3200 Hillview Ave.
Palo Alto, CA 94304

ABSTRACT

The Standard Generalized Markup Language (SGML), defined in International Standard (ISO) 8879, is a notation for representing documents and making their inherent structure explicit. The open-ended list of SGML applications includes document interchange, formatting or typesetting, loading databases for information retrieval, stylistic or linguistic analysis, and computer-aided translation. The combination of SGML and TEX is a natural one. This paper reviews the philosophy of SGML and then describes a particular environment where SGML and TEX are used together, giving specific examples of how processing is shared between the SGML application and TEX macros.

Principles of SGML

The Standard Generalized Markup Language (SGML) evolved from macro-based word-processing and text-formatting tools. Like a TEX macro package, it encourages a writer to use **descriptive markup,** identifying structures within a document, rather than **procedural markup,** specifying processing. For example, "this is a section heading" is preferred to "center this line in boldface". As the word "generalized" implies, documents prepared with SGML can be processed in various ways. Descriptive markup allows authors to concentrate on content rather than appearance [2]. Since the syntax of an SGML document is independent of any processing performed, no recoding of the document source file is needed to submit an existing document to a new application. The same markup used to prepare a book index, for instance, might also be used by an information retrieval application to locate text relevant to selected terms.

SGML is defined in International Standard (ISO) 8879 [3], adopted in October 1986. It views a document as a hierarchy of **structural elements.** For example, a manual may be composed of front matter, some chapters, optional appendices, and an index. Similarly, a chapter may be a series of sections, while a section is composed of text and optional figures, tables, lists, and so on.

No finite set of structural elements can account for the vast flexibility permitted in written texts. SGML therefore provides features for defining types of doc-

uments and then coding particular documents that belong to the defined types. Possible document types include reference manuals, journal articles, memos, and letters. A document type is formally defined with a **document-type definition** that itemizes the structural elements permitted in documents of that type and defines the contexts in which each element can occur. Most SGML users concentrate on creating and maintaining documents that conform to existing document-type definitions and hence do not need to learn the syntax for specifying new definitions.

Document-type definitions frequently distinguish elements that are formatted in similar fashion. For example, newly introduced terms and titles of books may both be typeset in italics. However, logically they are different structures. Markup that distinguishes between them allows software to support maintenance of glossaries and bibliographies.

The document-type definition can control context-sensitive interpretation of parts of a document. For instance, an asterisk may be interpreted as a code for the multiplication symbol inside a mathematical formula but as a footnote indicator elsewhere. Context-sensitive knowledge of valid document structure also permits various abbreviations of SGML constructs, called **markup minimization.** If it is known, for example, that every chapter begins with a chapter title, the SGML processor can recognize the first words in a new chapter as the title whether or not the writer has explicitly coded them as such.

Most SGML markup consists of identifying the beginning or end of structural elements. The most common convention (which can be overridden) is to mark the beginning of an element with the element name enclosed in angle brackets and to mark the end of an element similarly, but with the element name preceded with a slash. These delimiters are illustrated in the following (deliberately verbose) example:

```
<glossary>
<title>Glossary of Animals</title>
<entry><term>Aardvark</term>
  <definition>The first animal listed in a
              dictionary.</definition></entry>
<entry><term>Cat</term>
  <definition>A domesticated <xref>feline</xref>.
  </definition></entry>
<entry><term>Dog</term>
  <definition>A domesticated <xref>canine</xref>.</definition>
</entry>

                    .
                    .
                    .

</glossary>
```

This example assumes that the document-type definition specifies rules for creating glossaries. Glossaries in this context are assumed to have titles and

to contain multiple entries. Each entry has a term followed by a definition. Definitions may contain cross-references to other terms in the glossary.

The document-type definition may also specify context-sensitive text-entry conventions. For example, glossaries may be defined so that the title and terms never extend past the end of a line and that entries are separated by blank lines. With these definitions, SGML treats the following exactly like the more complete form shown above:

```
Glossary of Animals

Aardvark
The first animal listed in a dictionary.

Cat
A domesticated <xref>feline</xref>.

Dog
A domesticated <xref>canine</xref>.
                    .
                    .
                    .
```

Combining TEX with SGML

Although SGML allows multiple applications to be performed on a single source file, document formatting is the most common application. TEX is a natural choice for the back-end of an SGML-based formatting system. Several independent reports of environments where the two are used together have been made [1, 6, 8]. The complementary nature of the two languages is such that TUG has maintained liaison with the SGML standards community in the United States since 1982 and *TUGBoat* regularly publishes liaison reports.

Even when no applications other than formatting are planned, some experienced TEX users prefer to code their documents in SGML, which can be automatically translated to TEX, rather than using TEX for the original source file. Such an approach enables the markup minimization features discussed above. In addition, SGML's knowledge of context automates some forms of error checking. Many TEX users can sympathize with someone who inadvertently omits the closing brace after an emphasized phrase and generates several pages printed in a boldface font or who neglects to close an indented list and discovers the rest of the document in narrow columns. An SGML parser, referencing the appropriate document-type definition, knows that an emphasized phrase cannot span multiple paragraphs and that an indented list cannot cross a chapter boundary. When such markup occurs, the effect can be limited to a single paragraph or chapter and appropriate error messages issued. This context checking is an inherent property of SGML rather than something that must be laboriously built into individual macros.

Various techniques for defining SGML applications have been used in converting SGML documents to TEX [5, 9]. One such system, called MARKUP, is described briefly below [7]. It should be noted that these tools typically simplify conversion of documents conforming to a particular SGML document-type definition into the form required by a particular TEX macro package; they do not automate preparation of any SGML document for an unspecified book design.

MARKUP and the HP Tag Project

User guides and reference manuals for Hewlett-Packard computers, software, and electronic instruments are produced by staff members located throughout the world in more than fifty independent writing departments. SGML supplies a means of standardizing markup conventions throughout the company, thereby allowing interchange of files without requiring replacement of all existing text processing software and the corresponding hardware. A shared markup technique also provides a vehicle for discussion among writers in different groups.

Over the last two years, the company has developed an internal SGML standard for documentation markup and implemented supporting software. HP Tag is an SGML document-type definition that describes the structure of Hewlett-Packard user documentation. The first application of HP Tag is typesetting with TEX. This package, available on two different computer systems, is used for English documents as well as manuals written in other languages. Its first production release was made available to documentation groups in February of 1988, and there are now over 250 users.

Programs have also been written to load HP Tag documents onto CD ROM for interactive retrieval and to prepare material for input to computer-assisted translation. Plans for the next several months include conversion of HP Tag documents to and from WYSIWYG systems as well as enhancement of existing applications. Areas receiving particular attention include revision control and increased support of illustrated material.

A general-purpose SGML parser and application generator called MARKUP facilitates development of HP Tag applications. As are the HP Tag applications, MARKUP is an internal tool rather than a Hewlett-Packard product. MARKUP applications are specified in a notation similar to that of traditional parser generators such as *yacc* [4]. The MARKUP programmer completes a table which indicates the processing to be performed for each instance of every element included in the document-type definition. Table entries specify actions to be taken at the beginning of the element, within it, and at its end. These actions are triggered whether the element is delimited by explicit tags or implied by the minimization conventions.

When the MARKUP application generates a TEX source file analogous to the original SGML input, the actions usually consist of the TEX markup corresponding to the SGML construct. For example, the string `{\it` might be generated at

the beginning of a book title, an introduced term, or a variable component in a computer command, while } is generated at the end of these structures. When a quotation mark occurs within normal text, the TEX open-quote convention `` is generated; when a quotation mark occurs within a quote element, the close-quote sequence '' is output. Similarly, \bye is generated at the end of the document, whether or not the writer bothered to enter an end-of-manual tag.

When necessary, actions can also be entered as C code to be executed when the corresponding structure occurs. Examples of the use of C code in the MARKUP application definition are given below.

Style of the Generated TEX Coding

The style of TEX macros used in the HP Tag project is illustrated in the following example. The SGML form of the input is shown first:

```
The ++red pencil++ is among the most versatile of
editing tools.  And versatility is essential, for the tools
and for the editor.  Jan White--in his valuable handbook
<book>Editing by Design<\book>--writes of the varied
skills needed to "organize the material in such a way that its
!!significance!!  stands out."
```

The TEX input automatically generated from this paragraph has a different style than the same material would if manually coded:

```
\beginpar The %
\pushfont\termtext
red pencil\/\popfont{} is among the most versatile of
editing tools.  And versatility is essential, for the tools
and for the editor.  Jan White\EMDASH{}in his valuable handbook
%
\pushfont\booktext
Editing by Design\/\popfont{}\EMDASH{}writes of the varied
skills needed to ``organize the material in such a way that its
%
\pushfont\emphtext
significance\/\popfont{} stands out.''\endpar
```

Although HP Tag observes the same convention TEX does of separating paragraphs by one or more blank lines, in the generated file, paragraphs are explicitly delimited by the **\beginpar** and **\endpar** control sequences. Many control sequences are preceded by % and a line break to ensure that the generated line does not exceed TEX's input buffer and can be comfortably viewed on an 80-column screen. Macros such as **\popfont** and **\EMDASH** are followed by empty braces to prevent inadvertent concatenation of a control sequence name with any following text. Font changes are made through a font stack controlled by macros **\pushfont** and **\popfont**. In this way, font changes can span TEX groups.

The HP Tag TEX macros are not parameterized as they would be if the calls were user-written instead of automatically generated. Options are explicitly coded rather than allowed to default. For example, the chapter title is normally printed on the outside margin of the page footer, but the user can specify a different footer if the chapter title is too long to fit in available space. User-invoked macros should be designed for the usual case. The chapter macro needs one parameter, the chapter title. To override the default page footer, the user can call a second macro. Since the HP Tag macros are automatically invoked, however, the chapter macro can have two parameters, the chapter title and the footer specification, even though the values are usually identical. This repetition is not tedious to the user, since he enters the chapter title only once. Furthermore, there is no risk that two copies intended to be identical will in fact differ.

Early versions of the MARKUP application used fewer macros than the current one. Consider, for example, the TEX code used to start a list. A macro is not necessarily needed for this purpose. Macros are used to give a convenient label to sequences of instructions that are needed repeatedly. In this case, the code is isolated in the start-list cell of MARKUP's definition table. MARKUP invokes it when needed and, in effect, it has already been given a logical name (start-list). However, debugging is simplified when macros are used; the TEX source file generated by MARKUP is more readable when it contains macro calls. Furthermore, the macro approach allows one developer to concentrate on writing and debugging macros while someone else implements the MARKUP translation scheme.

Errors in an HP Tag file are reported by the MARKUP application. It is intended that reports of overfull and underfull boxes be the only error messages produced when TEX processes a file generated by the HP Tag convertor, as long as the original HP Tag source file was structurally correct. Minimizing TEX errors is especially important because authors are not required to be able to use TEX or to understand most of its messages. However, detecting errors in the translation phase forces messages to be written in terms of the user's view of the document rather than referring to internal details of the formatting implementation that the user probably will not understand.

Allocating Tasks to TEX and MARKUP

The processing required to typeset an HP Tag document is divided between the HP Tag to TEX conversion application and a TEX macro package. In some cases, such as counting items in a list or chapters in a manual, it is rather arbitrary whether the work is done with TEX or C code. Formatting-related tasks, of course, are reserved for TEX. The C interface is used to maintain data on cross-references, to replace data characters that have special meaning to TEX with control sequences that cause those characters to be printed, and to translate 8-bit accented letters to corresponding sequences of accents and letters.

The ability to supplement formatting instructions with a conventional programming language has enriched the type of processing performed. C makes some processing easier to do and allows some actions that cannot be done with TeX. For example, when a new term is introduced in an HP Tag document, the HP Tag to TeX translator outputs TeX control sequences to print the term in a contrasting font. It also saves the term in a table. If the manual contains a glossary, terms from this table are compared to glossary entries. A warning is issued to the author identifying any terms entered in the table but not defined in the glossary. In the future, terms identified in the text may be used to generate a draft of the glossary from a database of standard definitions. The application may verify that glossary entries are in alphabetical order, using the collating sequence appropriate to the language in which the book is written. Similar techniques might be applied to generate and alphabetize a potential list of related documentation from the manuals mentioned throughout a text.

The current application also uses C code when a period or comma occurs after an emphasized phrase or book title. The HP Tag environment prints the punctuation in the same font as the preceding material whether the author has placed it within the element or after it. A consistent style is thus produced throughout a set of manuals regardless of whether the writers have considered this issue. This consistency is enforced by the MARKUP application which delays generating the TeX instruction to return to the original font until determining whether punctuation has occurred.

Other types of consistency are enforced through the HP Tag document-type definition. SGML makes them available immediately with no special attention by the programmer developing an application. For example, HP Tag specifies that an ellipsis can be entered as 3 periods, optionally separated by spaces ("..." or ". . ."). Whichever convention an author uses in a particular instance, all HP Tag applications will generate the same results. Similarly, a pair of adjacent hyphens, possibly preceded or followed by spaces, is always interpreted as an em-dash. (This convention deviates from the normal em-dash convention of TeX.) As illustrated in an earlier example, the HP Tag to TeX conversion program translates these sequences to the control sequence `\EMDASH` which outputs an em-dash and prohibits a preceding line break.

Conclusions

In summary, using TeX as a back-end to SGML provides authors with all the formatting power of TeX but with a simpler markup scheme. SGML's context-driven input stream enables markup minimization and checking for context-related errors. Error messages are phrased in terms of the structure of the document instead of unsuccessful attempts to process it. Applications other than formatting can be performed with no change to the source file.

Bibliography

[1] Brüggemann-Klein, A., P. Dolland, A. Heinz. How to Please Authors and Publishers: A Versatile Document Preparation System at Karlsruhe. Pp. 9–31 in *TeX for Scientific Documentation*, Proceedings of the Second European Conference, Jacques Désarménien, ed. (*Lecture Notes in Computer Science* 236). Berlin: Springer-Verlag. 1986.

[2] Coombs, J. H., A. H. Renear, and S. J. DeRose. Markup Systems and the Future of Scholarly Text Processing. *Communications of the ACM* 30:933–947. 1987.

[3] International Organization for Standardization, Information Processing—Text and Office Systems—Standard Generalized Markup Language (SGML), ISO 8879-1986(E).

[4] Johnson, S. C., and M. E. Lesk. Language Development Tools. *Bell System Technical Journal* 57:2155–2175. 1978.

[5] Le van, H., and E. Terreni. A Language to Describe Formatting Directives for SGML Documents. Pp. 98–119 in *TeX for Scientific Documentation*, Proceedings of the Second European Conference, Jacques Désarménien, ed. (*Lecture Notes in Computer Science* 236). Berlin: Springer-Verlag. 1986.

[6] Price, L. A. SGML and TeX. *TUGBoat* 8(2):221–225. 1987.

[7] Price, L. A. A Parser Generator for SGML. Pp. 118–123 in *PROTEXT IV: Proceedings of the Fourth International Conference on Text Processing Systems*, J. J. H. Miller, ed. Dublin: Boole Press. 1987.

[8] Smith, C. DAPHNE (Document Application Processing in a Heterogeneous Network Environment): An Implementation Based on the Standard Generalized Markup Language (SGML). *SGML Users Group Bulletin* 1(2):75–82. 1986.

[9] Smith, C. A General Interface Solution for SGML Formatting Applications. *SGML Users Group Bulletin* 2(2):87–89. 1987.

DVI Previewers

KEN YAP

Department of Computer Science
University of Rochester
Rochester, NY 14627
ken@cs.rochester.edu

ABSTRACT

DVI previewers provide a convenient and valuable link in the authoring process that saves time and costs. While the turnaround time of medium resolution printers is measured in minutes, that of a previewer is measured in seconds. In return for speed, several trade-offs have to be made. The resolution is sufficient to verify page layout, page breaks, and the placement of large objects, but not to easily observe finer details such as spacing corrections. There is the problem of obtaining previewing fonts at the desired resolution. Embedded graphics that use page description languages such PostScript present another problem.

Three previewers, **dvitool**, **xdvi** and **texx**, are used as case studies. These exhibit a variety of approaches to the problems mentioned as well as different styles of user interface. Some speculation on future prospects for DVI previewers is indulged in.

The Authoring Cycle

The traditional authoring cycle on a glass teletype interface to TEX is *edit/format/print*. Although TEX is better than other formatters at catching syntactic errors in a document, verifying page layout, page breaks, and the placement of entities like equations and paragraphs requires hard copy. Unless one is fortunate to have a printer adjacent to one's terminal, a trip to the printer room is required, perhaps even a wait for one's print job to reach the head of the queue. This imposes many of the characteristics of the old-fashioned batch job environment on work, even if TEX is run interactively. The turnaround time and printing costs make this approach even less palatable for tasks like debugging TEX macros.

The widespread availability of graphics displays has changed this. These machines have screens that are addressable by pixels instead of by characters. A common resolution is 80 dots per inch (dpi). With such screens it is possible to display a rough representation of the printed page. The authoring cycle then becomes *edit/format/preview*, and *print* is, one hopes, the final stage executed only once.

Principles of Operation

A DVI previewer is a program that interprets the information in the DVI file. It takes the glyph and positioning information in a DVI file and combines this with font files to obtain raster images that to be displayed on the screen. A DVI previewer has many similarities to an output driver but, of course, previewers are intended to run interactively. Fortunately, the DVI format was designed to allow random pages to be located rapidly. Commands to display a specific page are easy to implement.

Displaying DVI output on a bitmap display is conceptually simple. When the program encounters an output command, it just takes the raster of the character in the current font and transfers it to the screen at the current location. Rules have to be drawn with line drawing commands. Positioning commands simply update the current position on the screen. When an end-of-page command is encountered, the previewer displays the current page and awaits a user command. Messy little details such as maintaining the DVI stack, font bookkeeping and caching, obeying the MAXDRIFT rule to guarantee that the error does not exceed one pixel, and interactive control are the concern of the previewer author.

Returning to a previous page or repainting the page is a common operation and many previewers transfer the characters both to the screen and to an off-screen memory bitmap in unison so that redisplaying the page is fast. Some previewers even attempt to rasterize the next page while the user is looking at the current one.

Background

The three previewers in this case study come from various sources. `Dvitool` is a SunView program from the VorTEX tools developed at the University of California at Berkeley, `xdvi` and `texx` are contributed software from the MIT X Windowing system tape. All three previewers run on UNIX. `Dvitool` is distributed with a set of previewer fonts, while the other two previewers use fonts intended for downloading to laser printers. All three previewers display on bitmapped screens.

Fonts

There are two principal means of obtaining bitmap fonts for previewers. The first is to generate previewing fonts at the screen resolution with METAFONT. At low resolutions, METAFONT simply has no opportunity to work well. Rounding error ([3] Chapter 24) takes its toll. As expected, fonts with less high visual frequencies, like `cmtt` and `cmss`, survive better. Hand-tuning with a font editor is needed[1] to get a good appearance. Release 1.0 of `dvitool` provided tuned fonts. In later releases, this was abandoned because of the sheer number of fonts to be treated.

[1] Nay, probably mandatory for commercial products.

The second method is to sub-sample the printer fonts; **xdvi** and **texx** use this approach. The basic method is to reduce each $n \times n$ block of pixels to a single pixel. The algorithm is usually simple thresholding, that is, if more than a certain fraction of bits in the block are 1, the result is 1. This method has the advantage that any font available to the printer is also available to the previewer, economizing on disk storage. Sub-sampling takes no notice of character outlines and often produces shapes that have broken outlines and that are recognizable only in the context of other letters. Sub-sampling is a CPU-intensive operation because of the bit operations involved. It needs to be done only once for each distinct font however. Anomalies may appear in the final image. A user once complained that **xdvi** previewer was losing some LaTeX lines. A little investigation showed that the lines were thin enough and straddled pixel blocks in just the right way to suffer elimination. Another disadvantage is that the zoom factors are limited to those provided by integer divisors of the printer resolution.[2]

If zoom is provided using dedicated previewing fonts, fonts have to be kept at several magnifications. The disk storage devoted to previewing fonts may rival that for printer fonts, even though the individual font files are smaller.

If the sub-sampling method is used to generate fonts, the previewer can show the user the appearance of the output page by turning off sampling. In effect this provides the user with a magnifying glass. **Texx** is one previewer with this feature. The rasterization process is even slower, but this feature has proved useful for debugging minute spacing corrections in mathematical formulae.

When a previewer cannot find a font at the right size or resolution, it can substitute the same font in a lower magstep or lower resolution, on the principle that it is more useful to display something close than nothing at all. Some previewers allow user-specified substitutions. This is good, for example, for looking at that odd DVI file that arrived without TeX source and uses obsolete **am** fonts.

Non-TeX fonts such as the Adobe PostScript fonts are a problem. Recent output drivers allow native printer fonts to be intermixed with Computer Modern fonts. These fonts are usually printer resident and proprietary, which means that bitmap versions are generally not available at low resolution. Perhaps the problem will be solved when type foundries release low-resolution versions of printer fonts for previewing purposes.

Vector Displays

Vector displays are output devices that have enjoyed more popularity in the past. Unlike bitmap displays, the screen is not considered as a matrix of dots which can be addressed on a per-dot basis, but as a screen upon which lines are drawn.

[2] Fonts in steps of TeX's magsteps can provide zoom too, but there is no guarantee that a given font is available at a given magstep.

Typically, the driving program sends the display a list of commands specifying the start and end points of lines. These vector displays are to pen plotters what bitmap displays are to bitmap printers.

Previewers exist for vector displays. It is a bad idea to try to simulate a bitmap display on a vector display because the communication channel is too slow to send thousands of dots. It is more sensible to draw the best approximation to the character with strokes. TEX fonts are normally bitmap fonts so other fonts have to be substituted. The Hershey fonts, which were originally developed for plotters, are a popular choice because of their easy availability (practically free). One has only to assemble fonts that resemble Computer Modern. Even so, many mathematical symbols and foreign ligatures are missing. The characters plotted are the best Hershey approximations to Computer Modern so it is pointless to pay heed to anything other than gross positioning problems on the screen.

Changing the magnification on vector displays is easy when vector fonts like Hershey are used—only the scale factor between the stroke dimensions and the display needs to be changed. However, the relatively thick lines drawn by vector displays impose a limit on the fineness of details that can be discerned.

Previewer Limitations

Low resolution is the bane of previewers. A 10 points the character a in Computer Modern Roman is just under 0.07 inches wide. At 80 dpi this means just 6 horizontal pixels are available to represent the character. The drastic drop in quality can best be appreciated by looking at bitmaps of the letter a at 300 dpi and 80 dpi.

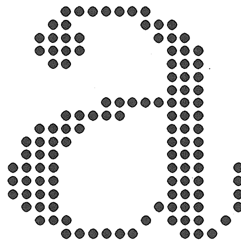


Figure 1: Bitmap of the letter a at 300 dpi

Fine details of letters like the tails are simply lost and only in the context of other letters does the human eye perceive the low resolution version as an a.

One way of artificially raising the effective resolution is to abandon the idea of displaying pages at true size. For example, the display resolution can be set to 120 dpi (120 dots per document inch, not screen inch) or greater, which helps improve display quality. Pages then display greater than life-size and a full page

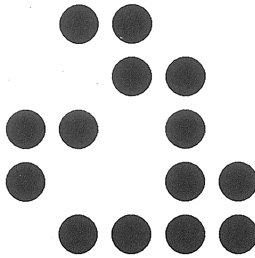


Figure 2: Bitmap of the letter a at 80 dpi

cannot fit on the screen anymore. Commands have to be provided to scroll the page. These are necessary in any case for small screens, and to display oversize pages, for example, those generated for large conference proceeding mats.

Grey Scale Fonts

The curse of low resolution may be partly alleviated with grey scale or color monitors, for those lucky enough to have one. If the size of a pixel is small enough, brightness and size can be traded off. Warnock [5] found grey scale fonts readable in small point sizes at which black and white fonts were illegible. 256 grey levels can be generated by a block of 8×8 pixels and thus would appear to improve the effective resolution by a factor of 8 over a black and white display. However it is not clear that a linear relationship between brightness and apparent resolution holds for human perception. In any case, only a factor of 4 is needed to raise the effective resolution to that of a 300 dpi printer. A previewer recently posted to Usenet, `dvipage`, uses grey scale fonts. These fonts are generated on the fly by filtering printer fonts.

Previewers in Use

How do users typically use previewers? As previously noted, fine details are invisible on the screen. Fortunately, previewers are mostly used to verify the composition and placement of large entities, such as equations, paragraphs and figures. Before the advent of graphical editors like `fig`, previewers were invaluable for debugging LaTeX pictures. The placement of floating figures is also easily observed with a previewer. The difference between ordinary and bold text often cannot be seen. However, italic text is visible as such.

Only a small set of previewer operations are essential. Most users only require scrolling, forward and backward paging, and the ability to jump to a specific page. It is desirable to provide searching by both physical and logical page (TeX's `\count0` and `\count1`). After that come the frills. Zooming is useful, as

is the means to trim blank margins to maximize use of the screen area. **Texx** has, as mentioned before, a two-level zoom, which suffices for most needs. String search is handy but is very difficult to implement correctly for all circumstances.

Previewers are slow to start up and one can amortize the overhead of reading fonts and other work if one instance of the previewer is kept active, instead of invoking a fresh copy after every change to the document. It is useful to be able to change to a different DVI file while in the previewer or, at least, to reread the current DVI file. `Dvitool` can be combined with TEX in a *format/preview* loop. Users at this site (University of Rochester) often keep a `dvitool` window active, ready to use at a moment's notice. **Texx** has a command to reread the DVI file. This is used to display the results of the latest rerun of TEX on a document.

`Dvitool` allows customizable key bindings, à la Emacs. Obviously the author is a fan of Emacs. An informal survey of users at this site showed that hardly anyone used private bindings. The development effort would have been better spent on making the tool conform to some windowing interface style (pull-down menus, scrollbars, and similar things), for which standards are emerging.

Extensions

TEX was designed primarily for typesetting text. Often, users need more than what LATEX pictures or PICTEX can provide. (DVI files generated by these methods require no non-standard fonts.) The `\special` feature of TEX is used for this pictures. In one method, `tpic` specials are used to specify geometric entities, e.g., lines and circles. Obviously, the previewer must be capable of interpreting these specials or blank pictures appear on the screen.

The other major approach is to include specials that command the output driver to interpolate printer-specific code in some page description language[3] (PDL) like PostScript. This approach is often used for including digital halftone pictures such as the output of screen dumps. Although TEX has been persuaded to print halftone pictures[4], these pictures can overflow TEX capacities and standards do not yet exist for encoding and software, so the use of printer-specific specials is common.

Such specials are a problem for previewers. Even assuming the marketplace standardizes on one PDL—an unlikely possibility, previewing such documents requires implementing a general PDL interpreter in the previewer, a daunting task.

Unlike text, graphical objects are less amenable to automatic placement and are therefore more likely to require previewing even though they are more difficult to handle than text in previewers.

---

[3] A page description language defines the set of typesetting commands understood by the printer.

Teletype Previewers

We should also note in passing another, cruder variety of previewer intended for use on glass teletypes. An example is a recent proposal in *TUGboat*[2]. Here at this site, we have had good service from `dvitty`. It is useful for glass teletypes, dial-up connections, and when one is too impatient to start up a bitmap previewer. In the output from this previewer words run together, font changes go unnoticed, and ligatures and math symbols are illegible. This previewer allows one to note the position of page breaks and floats, and that is about all.

The Future

The resolution of displays will improve. Already 120 dpi and higher resolution displays are supplied as standard with some workstations. The human vision system has a maximum resolution of about 60 cycles per visual degree. This translates to about 600 lines per inch at normal viewing distance[1], so screens do not have to improve indefinitely. Some day the resolution of screens will equal that of the printed page and there will be no need to make adjustments for the different output resolution.

DVI previewers for bitmap displays are often constructed on top of a graphical library or some windowing system, such as QuickDraw, SunView or X. It is likely that a user interface standard will emerge for previewers using the same windowing system, thus reducing the learning load for users.

TEX has placed computer typesetting on a firm mathematical basis. It is a prime candidate for the formatting sub-system in publishing systems. Previewers are an anomaly introduced by the discrepancy between the speed and quality of screens and printers. As the quality of screens approaches that of printed copy, previewers will become widely integrated into publishing systems. There will be interactive tools that allow not only viewing the output of a run but also allow editing objects for the next run. The *format/view* cycle will be shortened by WYSIWYG tools that handle smaller pieces of input. The publishing system of the future will allow an author to combine page, text, drawings and half-tone or color photographs on the page from a variety of sources. It is conceivable that publishing stations, analogous to workstations, equipped with prodigious formatting power, font, and image resources, will become standard off-the-shelf items in vendors' catalogues.

Acknowlegements

Thanks are due to the authors of the three primary examples of previewers studied in this paper. **Dvitool** (Jeff McCarrell, UCB), **xdvi** (Eric Cooper and many others), and **texx** (Dirk Grunwald, University of Illinois) all run on UNIX. Many thanks also to the authors of other previewers for sending the author useful documentation. The programs and authors (or informants) are: **DVIVIEW** (Peter Scott, JPL), **dvibit** (Nelson Beebe, University of Utah),

DVIDIS (Jerry Leichter, Yale University), DVIPerq (Paul Milazzo, Rice University), CDVI (Wayne Sullivan), DVI3279 (Dr. Georg Bayer, Rechenzentrum der Technischen Universität Braunschweig), DVI82 (Malka Cymbalista, Weizmann Institute of Science), DVItoVDU (Andrew Trevorrow), TXMAPPER (M.L. Luvisetto and E. Ugolini, CNAF).

I am grateful to M. Srinivas for his editorial comments on this paper. I lay claim to any residual errors.

Bibliography

[1] Bigelow, C. Proceedings of the Typography Interest Group, ACM CHI'85. *SIGCHI Bulletin*, 17(1):9–15, July 1985.

[2] Brown, M. An ASCII Previewer for TEX. *TUGboat*, 9(1):34–36, April 1988.

[3] Knuth, D. E. *The* METAFONT*book*. Addison-Wesley, 1986.

[4] Knuth, D. E. Fonts for Digital Halftones. *TUGboat*, 8(2):135–160, July 1987.

[5] Warnock, J. E. The Display of Characters Using Gray Level Sample Arrays. *Computer Graphics*, 14(3):302–307, July 1980.

PreTEX: Tools for Typesetting Technical Books

ROBERT L. KRUSE

Department of Mathematics and Computing Science
Saint Mary's University
Halifax, Nova Scotia
B3H 3C3
Bitnet: **kruse@stmarys**

ABSTRACT

PreTEX is a new system for simplifying the typesetting of technical books with TEX. It consists of a preprocessor for TEX (written in Pascal), a large package of macros, and auxiliary programs for producing an index and other supplements, and for performing other tasks. The preprocessor supplies much of the detailed markup required for good results with plain TEX and greatly simplifies the typesetting of computer programs. The macro package allows an author to use only logical markup in the text itself. At the same time, it provides the book designer with flexibility in the placement of elements, choice of dimensions, color use (when appropriate), and selection of type fonts. The indexing software produces page ranges automatically and edits the index to three levels of entries and subentries.

Introduction

This paper introduces a TEX preprocessor, macro package, and auxiliary programs, collectively called the PreTEX system. I developed the initial version of PreTEX for my own use in writing and typesetting several technical books (computer science textbooks) during the past few years, but now it has grown in power and generality to a point where it may prove useful to other authors and to publishers of technical books. The system is, in fact, rather large, consisting of more than 3000 lines of Pascal in the preprocessor, more than 500 macros, together with several smaller computer programs for special purposes such as constructing an index or extracting specialized material such as solutions to exercises or private documentation that is included with the text but is intended to be published either separately or only in special versions of the document.

At present, PreTEX is implemented on the IBM PC and compatibles and on the VAX/VMS system. Since the programs are written in standard Pascal (except for input and output) and the macros in plain TEX, the PreTEX system

can be quickly ported to most computers supporting both a Pascal compiler and TEX.

1 Goals

1.1 Author's perspective

The writing, typesetting, and production of a technical book are long and fraught with problems, but the process can be simplified by the separation of concerns, that is, by concentrating on one aspect of the process at a time. During the writing of a book, the author should not need to think at all about the typesetting process. This means that the markup in the manuscript should be strictly logical, specifying *what* each element is but never *how* or *where* it should appear in the final format. Certainly the author should never need to be concerned with the *minutiæ* of typesetting, details like italic corrections, various kinds of dashes or quotation marks, the insertion of backslashes before common mathematical symbols, or special commands needed to set keywords or comments from a computer program in different fonts.

The first goal of the PreTEX system is to relieve the author of as many of these concerns as possible. The preprocessor does so by automatically inserting much of the markup required by TEX to achieve its high standard of typesetting. It also translates many common symbols into the equivalent control sequences demanded by TEX. The macro package then complements this work by providing a large number of logical markup commands, chosen to describe all the elements appearing in most technical books.

At this first stage of its use, PreTEX is geared toward the technical writer, with names of commands chosen to be meaningful for an author, a straightforward syntax easier to learn than plain TEX, and with the verbal style for mathematics that mathematicians who use TEX appreciate, but with a simplified syntax.

1.2 Designer's perspective

The second stage in book production is design. Good design requires choosing a visual presentation that enhances the content of the book and commends it to its intended audience. To achieve these goals the design artist needs a great deal of flexibility and the ability to specify all the dimensions, the choice of type fonts, and the use of color in a language similar to that which book designers traditionally use. PreTEX therefore provides not only a selection of predetermined styles (from which the author and designer can make an initial choice) but files (with documentation) defining all the type fonts and dimension parameters used in the style. The design artist can therefore begin with a sample style and by changing the choice of fonts, the use of color (if any), and the values of various dimensions can alter the style as desired.

For the more complicated elements, such as chapter openers and section heads, the macros specify the design by putting together simpler building blocks. The artist more experienced with PreTEX can create new styles by putting these building blocks together in different combinations, as well as by changing the dimensions and type fonts.

With a screen previewer, the artist can experiment with many variations of the design easily and without the substantial work of producing mock-ups by hand.

1.3 Programmer's perspective

For any software system to continue to prove useful, it must be extensible to meet new requirements. The system should be as open and flexible as possible. The PreTEX macros have therefore been designed to be modular, well documented, partitioned into files for easy access, and available for the TEX macro programmer to change as needed.

In everything it does, PreTEX remains consistent with the features of plain TEX, so that almost anything written in conformity with the rules of plain TEX will be processed in exactly the same way, and so PreTEX can be extended by writing new macros as desired. Anything that can be accomplished with plain TEX can also be accomplished under PreTEX with no additional difficulty.

1.4 Further objectives

During the production process, the author and production editor need various extracts from the text, so PreTEX provides utility programs that produce element lists, art lists, and the table of contents automatically. Another utility extracts computer program listings from the text so that they can be tested. Further software takes TEX's output of index entries (inserted by the author into the original text), sorts them, determines page ranges, edits them into main entries, subentries, and sub-subentries, and typesets the index in one, two, or three columns as desired, all automatically.

Solutions to problem sets, specialized documentation, commands to extract material for transparency masters, or other such supplementary material, can be embedded in the text files but will not be typeset with the text unless desired. Instead, this material can be automatically extracted (along with appropriate elements from the book text itself) and organized as a separate document. Hence the production of a textbook and its solutions manual or other supplements can proceed at the same time, and with any revision to the textbook its supplements can be brought up to date very quickly.

At present, sample PreTEX styles have been made for both Computer Modern and PostScript fonts (and combinations thereof). For a PostScript output

device, PreTEX provides macros and POSTSCRIPT procedures for producing two-color separation and halftone screens.

2 Preprocessor Environments

Much of the work of PreTEX is done by its preprocessor, which is written in Pascal. Tasks that can be done easily by TEX are passed through the preprocessor without change and processed by TEX macros. The preprocessor handles those tasks that are more easily performed by a computer program operating directly on strings of characters. Commands to the preprocessor are either legitimate TEX control sequences (which are then also sent through for further processing by TEX) or are constructions that would be illegal in plain TEX (such as using a circumflex or underscore outside of mathematics). In this way, the preprocessor maintains complete compatibility with plain TEX.

At any time, the preprocessor treats its input in accordance with one of several *environments*, in each of which the input is processed differently, and the same symbols may have quite different meanings. A hyphen between two digits, for example, becomes a minus sign (−) in a program (or in mathematics, of course), but becomes an en-dash (–) in text. Between two letters a hyphen in text remains, but becomes an em-dash (—) if it is surrounded by blanks.

At present, the environments implemented in PreTEX are text (the default, outer environment), mathematics, computer programs in various languages (Pascal, Modula-2, C, FORTRAN, Prolog, Smalltalk, and several others), algorithms, verbatim, and caps/small caps.

2.1 Text environment

The text environment differs only in small ways from that provided by plain TEX, but in ways that simplify the input for the author who is not already conversant with TEX. Straight double quote marks (") are translated into opening or closing double quotes according to simple rules that almost always produce the correct results. If the rules fail, then the standard TEX constructions remain available. Hyphens, single quotes, and dots change meaning according to the context. Font changes in text can be made with only two or three keystrokes. PreTEX keeps track of which fonts are oblique; it inserts an italic correction when the current font changes from oblique to straight and the following character is not one of a comma, period, or slash.

All kinds of elements in the text are automatically numbered. These include sections and subsections at various levels, theorems, figures, tables, footnotes, and list entries. There are several kinds of automatic list constructions, including arabic numerals, upper- and lowercase roman numerals, and upper- and lowercase letters.

2.2 Mathematics environment

In the mathematics environment, similarly, the syntax is simplified. Most common mathematical terms can be written without an initial backslash, and certain combinations of special symbols are translated into the equivalent TEX control sequences. When the user types three dots (...), for example, PreTEX substitutes either `\cdots` or `\ldots` according as the context requires.

2.3 Program environments

It is in the environments for typesetting computer programs that the PreTEX preprocessor achieves its major results. When programs are set in a variable-width font, the spacing around various symbols should be adjusted to reflect the symbol's use as a binary operator, a unary operator, a delimiter, or whatever. PreTEX therefore recognizes enough of the syntax of each programming language to determine how to interpret each symbol. Sometimes this interpretation varies with the context.

Even more importantly, many of the symbols used for special purposes in TEX are used for quite different purposes in computer programs. Curly braces, for example, normally delimit groups in TEX and do not appear in the output. In C, however, they are important for connecting several statements together as one block, and in Pascal they mark the beginning and end of a comment. Hence they must appear in the typeset version. One approach, of course, would be to change the `\catcode` of the special symbols used in programs. But doing so would make it difficult to insert TEX commands into a program. Macros for both index entries and marginal notes, for example, take parameters normally delimited by curly braces. Hence PreTEX does not (often) change category codes but instead recognizes whether material in braces is a parameter to one of the TEX macros in its lexicon or consists of program statements to be processed further.

PreTEX, moreover, enters its text environment when it processes the parameter of a macro in its lexicon, a comment in a computer program, or a `\vbox` or `\hbox` in mathematics, so that these constructions can contain font changes, special PreTEX text features, mathematics, or even another program segment as the user wishes.

2.4 Verbatim environment

The verbatim environment comes closer to reproducing its input exactly than plain TEX can with only its `\obeylines` and `\obeyspaces` macros. In the PreTEX verbatim environment all symbols are shown exactly as they appear in the input file except for the one command that ends the environment and a TEX control sequence preceded by the special command `\obey`, which will then be obeyed.

2.5 Caps/small caps environment

A few words should be said about the caps/small caps environment, since most TeX users think of using capitals and small capitals only as a font change. Doing so limits the use of capitals and small capitals to typefaces where such a font is provided, and this may be a limited number (only ten point roman and typewriter in the standard Computer Modern family, none in the PostScript family). Instead, the user should be able to use capitals and small capitals in any typeface. PreTeX provides this ability in its caps/small caps environment by translating lowercase letters to capital letters taken from an appropriately smaller size of the same typeface. In conjunction with PostScript, PreTeX allows the use of small capitals with any typeface at any size.

3 Macros

A more detailed description of the macro package in PreTeX may be published in another forum, but let me mention here only the basic philosophy of organization of the macro package. This philosophy is to present the macros on three levels.

The first level is that of the author who is concerned only with the logical description of the elements in the manuscript. On this level the macros should be simple and easy to describe, but sufficiently rich to characterize all the features that ordinarily appear in technical books.

On the second level come the macros concerned with book design. Here a much larger set of control sequences become available, but with basic constructions made up of building blocks that can be put together in many different ways with minimal interaction or side effects on each other. The user with only a basic understanding of TeX groups, boxes, glue, and the like should be able to understand and reassemble the macros that appear on this level.

Finally comes the technical level on which the precise definitions of all the various structures are given. It is only on this level where the more sophisticated constructions in TeX (such as conditionals, recursion, token lists, `\expandafter`, `\futurelet`, or `\aftergroup`) make their appearance. Very few users should ever have to look at the macros on this level. They should, nevertheless, remain accessible and be well documented so that the experienced macro programmer can make modifications as necessary.

4 Index Processing

The final important phase of PreTeX is construction of the index for a book. Again, presentation of the details must be deferred, but a general outline follows.

As TeX proceeds, it constructs files of index entries extracted from the text. These files are then sorted alphabetically by a postprocessor that first edits the entries by translating TeX control sequences into appropriate strings of characters if they generate alphabetic output and suppressing other control sequences.

The output from this phase is designed to be easily read by a person, so the author can correct errors or inconsistencies in the complete index at this time.

Next comes a second automatic process that notes the beginning and end of each page range, collects similar entries, and constructs the page ranges. This processor deletes duplicate entries and amalgamates overlapping page ranges. It includes facilities for attaching brief notes to the page numbers as appropriate. This processor also edits the entries by finding common initial segments in the text of successive entries and thereby arranges them into main entries, subentries, and sub-subentries. Its output file is now in a form that can be easily processed by TEX, and so the index is finally typeset. The special macros for typesetting the index can produce the index in one, two, or three columns as desired. These macros use the TEX `\mark` feature to repeat the text of a main entry if the list of its subentries continues past the end of a column. This repetition of main entries (and of subentries in the case of a continuing list of sub-subentries) can appear at the top of each column (when appropriate), at the top of the first column of the page only, or at the top of the first column of only even-numbered pages, as desired.

Conclusions

Many authors, impressed with the excellent quality of typesetting that TEX produces, are nonetheless intimidated by its complexity and with the substantial effort required both to learn TEX and to write macros sufficiently robust and flexible to handle the problems of typesetting a technical book. The PreTEX system provides all the tools that such authors need to produce the highest quality results without learning the intricacies of TEX or doing any programming of TEX macros. The language and command structure of PreTEX are easy for a technical writer to learn and sufficient to describe all the elements appearing in most technical books.

For the book design artist, PreTEX provides a good range of sample designs and a great deal of flexibility in modifying and adapting these designs. These modifications are produced by changing dimensions, type fonts, and basic building blocks described in terms similar to those traditionally used by design artists and typesetters in presenting the specifications for a book design. With no change at all to the manuscript itself, a book may be typeset in a wide range of styles, with one- or two-color printing, with or without material appearing in a margin, and so on.

For the TEX macro programmer, finally, PreTEX provides a solid basis for further development, with macros that are arranged and documented in a modular form. The entire system is designed to be compatible with plain TEX, so it can be extended and modified as desired with no loss of the flexibility and power provided by the features included in plain TEX.

Availability and Acknowledgements

At present, PreTEX is being tested in the production of several textbooks and their supplements, with different designs and with both one- and two-color printing. Extensive documentation is also being prepared. It is hoped that this testing and documentation will be completed during the summer of 1989, and PreTEX should then become available for more general distribution.

My students Steven A. Matheson and J. David Brown have assisted me in many ways in the development of PreTEX. They have helped with programming in Pascal and in TEX, with testing, and with applying PreTEX to various documents. I am grateful for their contributions.

CapTEX: Industrial Strength TEX

MIKE SCHMIDT

Honeywell Bull Ltd.
1101 Prince of Wales Drive
Suite 255
Ottawa, Ontario
K2C 3W7
Schmidtm@cmr001.bitnet

ABSTRACT

Contrary to the usual TEX-based systems, CapTEX uses TEX's strengths to build a markup language that defines document structure, rather than document format. Structure-oriented documents are much more suited to the "pour into a mold" approach required by the technical publishing industry. CapTEX is the core of Honeywell Bull's CAP (Computer Aided Publication) package. This system is designed for large-scale use, and includes such things as controlled updating, spelling and style checking, and document databases. While other TEX-based systems concentrate on personalized formatting for printing on paper, CapTEX concentrates on the structural elements of documents, so that the same source document can be used in many different ways, among them demand printing, on-line browsing, and interactive help. CapTEX is an excellent system for producing "standard" documents, no matter what the "standard" (nor how many standards there may be!).

Background

Back in the late 70s, when Honeywell Bull's CP-6 operating system was being developed, the early stages of software engineering were making themselves felt as the design team opted for machine-resident documentation, which would be delivered both as site-printable manuals and help databases. Internal technical information and error messages were kept directly in the source files, to be extracted by a process only vaguely similar to Don Knuth's `WEB` system. A very tight version and change control system was also put in place, for both the code and the documentation.

At that time, the only text formatter available to the CP-6 design team was a re-hosted version of Multics Compose, which, to some degree, was the precursor of the *nroff/troff* class of formatters. It did not, however, handle such things as multiple fonts, or, in general, anything not suited to a line printer.

Starting with TEXT (the CP-6 name for this formatter), and building around it for the next 8 to 10 years, the CAP package slowly took shape. All CP-6 manuals have since been published with CAP. They are available in three forms: as published manuals, as print files for line printers, and as help databases. By the mid-eighties, the package had acquired a variety of tools: spelling checkers, style analyzers, forbidden/restricted word dictionaries, the beginnings of document databases, and a menu shell to make it all easy to use. Honeywell Bull began to consider that this package, which had been very effective internally, might become a marketable product. When inexpensive laser printers appeared, quality improved dramatically, but it was obvious that the TEXT formatter was no longer sophisticated enough to make CAP marketable.

At this point, the author, having used TEX at another site, became involved in the project, helping to select TEX as the new formatter, and to reconstruct the CAP system around it. TEX was finally chosen, even though some flavor of *nroff/troff* would have been much closer to the TEXT formatter in use at that time. However, TEX is much more flexible and powerful, and was already available on CP-6.

Introduction

CapTEX is a TEX macro package implementing the formatting function for the CAP system. It is directed at the technical publishing done by large software houses, engineering firms, government departments, manufacturing firms, etc. In these environments the technical writers don't make many formatting decisions, leaving that to a layout designer. The documents need merged text and graphics, and must be deliverable in several ways. Finally, the documents are revised often, and several writers may be involved in a single document.

The goal of CAP is to provide a high degree of automation in the technical publishing environment. We required a robust system, where expert document designers make many of the layout decisions. Technical writers should be able to concentrate mostly on the accuracy of their material.

CapTEX was designed to provide all the facilities of TEX, especially the typeset quality of the output, but within an environment with minimal markup in the document sources. Final layout is determined by style definitions, and documents can be moved from one style to another easily. This makes documents easy to update, and very consistent. It also makes them easier to use with version control systems.

Basic TEX directly handles such things as accented characters, grouping, and math mode. CapTEX is built on plain TEX, and tries to maintain as much compatibility with it as possible. However, the writers, unless they are typesetting math, don't need to be aware of plain TEX at all.

Because of Honeywell Bull's international ownership, CapTEX directly supports technical publishing in most European languages. CapTEX makes use of

a completely standard TEX, specifically so other versions of TEX can be used, assuming they are compatible with plain TEX. Michael Ferguson's Multilingual TEX, for example, can be used to handle the hyphenation when more than one language is needed. In Canada, many government documents are published in dual columns, French and English side by side. While CapTEX doesn't support paragraph aligned columns in its initial release, future releases probably will.

1. Document Structure

Probably the most unusual aspect of CapTEX, when compared to other TEX macro packages, is its writer/document designer division. The document designer has significantly more control over the layout than is usual in other TEX-based packages, such as LATEX. CapTEX's writer markup language, Document Structuring Language (DSL), is primarily used to identify the different structural elements of the document: the title, chapter, multi-level sections, lists, figures, tables, graphics. Such markup then also serves to logically break up the document for use in a larger database, and for building help trees. Style definitions associated with each type of structure determine the final document layout.

Formatting controls are still needed, of course, if only to handle any exceptions to the pre-defined formats. CapTEX therefore supplies the regular gamut of controls to manipulate fonts, spacing, page and line breaking, unformatted and verbatim modes, and various forms of alignment. However, writers rarely need to use such commands, since style definitions automatically determine the layout.

2. Style Definitions—The Designer's View

There are two levels of style information:

1. The *document style*, which defines all the constructs available for the entire document.
2. *Subordinate styles*, for each major structural element, such as chapters, lists, tables, figures, etc, all of which have their own styles.

This approach to style information permits easy sharing of elements such as table styles among different document formats. Complete document styles can then be built by simply pulling the right pieces from some library of pre-defined structural elements. Subordinate styles can be built either in isolation, or in conjunction with a document style.

From the style designer's point of view, CapTEX uses a simple syntax of the "fill in the blanks" variety, which lends itself well to forms-driven interfaces, although there is, as yet, no such interface. A wide variety of styles can be created quite easily, even without any real knowledge of TEX. Contrary to such systems as LATEX or AMS-TEX, where document styles require a solid grasp of TEX, CapTEX only requires a basic understanding in most cases.

2.1 Document styles

Document style files are really no more than a collection of lesser style information:

- Device class—this defines the device class, and automatically chooses fonts suitable to the device. This allows CapTEX to automatically support different devices, each with their own fonts. This is especially important when preparing line printer output. This is also convenient for CAP's menu manager, because a specific device driver can be selected based on the device class.
- Media class—this defines the paper stock, the orientation, basic margins, and such items as two- or one-sidedness. New media classes could easily be defined for such things as slides, etc.
- Finally, the collection of style definitions for chapters, tables, and other elements. All the formatting rules and parameters not defined by the device or media classes must be found in the subordinate styles.

2.2 Subordinate styles

The basic set of formatting rules comes from the chapter styles. Various chapter styles can be used to handle prefaces, chapters, appendices, the table of contents, index, glossary, and bibliography. Each chapter is treated as a group; changes in layout are local to the chapter. Chapter styles control such things as headings and footings, page numbering, basic layout, section headings, etc.

The chapter style mechanism is a very convenient way of switching formats going from front matter to the main sections to closing matter (appendices, bibliographies, an index, etc). Some form of "null" chapter style determines the formatting for smaller documents, where specific chaptering is not called for, such as memos or short reports.

Table styles determine all aspects of tables, from captions, table of contents entries, to the preamble TEX uses to build the table. CapTEX can handle two kinds of tables: large, multi-page tables with running headers (titles) and/or footers, and small tables, limited to a single page, which can float forward. Figure styles handle the equivalent functions, and also support large and small forms, while graphs always fit on a single page. Floats usually migrate no further than the top of the next available page, always maintaining their original sequence. In multi-column documents, tables, figures, and graphs may float within a column, or may be column-independent, occupying the entire page width.

List styles define various kinds of environments, usually itemized or enumerated lists, or other special purpose environments. Where chapter styles can only be used sequentially, one chapter replacing the preceeding one, list styles can be nested to any level.

Finally, there are also equation, footnote, index entry, bibliography and glossary styles. Index entry styles, for example, can be used to emphasize certain kinds of entries. Database software can then easily identify the different classes of entries.

All of this places the major burden on the document designer, who must identify and define all these subordinate styles. CapTEX's aim is to reduce the writer's load to the bare minimum, while sacrificing none of TEX's power and flexibility. The designer is left with nearly all the layout decisions. CapTEX tries to make the designer's task easier by providing a simple, consistent set of commands, and a library of macros for such things as counter manipulation, syntax analysis, and list management. As a last resort, though, the designer has the full power of TEX at his/her disposal.

3. DSL—The Writer's View

Production systems all have one basic need—to be robust, to recover from errors automatically, wherever possible. DSL, the set of writer's controls, tries to detect and recover from the most common errors.

It detects syntax errors wherever possible, issues messages, and takes some form of corrective action. When a chapter is called out, any currently open structures, such as tables, or lists, are closed with warning messages. Controls disallowed in certain environments are redefined, so their accidental use only provokes a warning. By isolating errors, and continuing on, CapTEX reduces the number of iterations required, and hopefully the writer can identify the problems quickly.

Automation is the main goal: writers simply add material to a document, then turn the crank, and out comes a new revision. The material they add is usually document-style independent, because the technical writers work to a great degree independently of the final layout.

Besides having less markup to handle than other systems, writers also need less training—not in technical writing, of course, but in learning the markup language. Many of the little coding tricks that writers need with other systems can be forgotten. That's not to say that the markup will always be at a structure level; there are always exceptions.

4. Graphics Support

CapTEX and the CAP device drivers use a special representation for graphics data, and support both raster and vector graphics. Import utilities convert a variety of external representations, from mainframe and workstation-based CAD systems, to drawing programs for microcomputers such as MacPaint or PC-DRAW. Graphics are automatically scaled and positioned to fit their allotted area, but always maintain their natural aspect ratio. Graphics can also be used as

overlays, reproducing pre-printed forms or adding a complex letterhead. CapTEX permits graphics in headers and footers, so drawn logos can also be used. It's much better to define logos with METAFONT, but that isn't always easy.

In most cases, graphics information is stored in its original format, only converted at the last possible moment. This permits editing in the original form, without any information loss that may result from the import process.

5. Unusual Features

There is no need to dwell on such features as are common to all or most similar packages. All the typical features are provided. However, CapTEX has a few unusual features which deserve to be mentioned.

5.1 Font controls

Font controls deserve special mention, even if they aren't unusual, since they are a very important part of the infrastructure. CapTEX uses a size/typestyle mechanism very similar to that used in LATEX. CapTEX expects all typestyles to be defined at all sizes, so it becomes easy to maintain the typestyle when moving a section head to a table of contents, for example, where the size may be different. Struts and other parameters dependent on font size follow suit. Math fonts can also be linked to the current size.

5.2 Spacing and dimensions

Besides providing the regular units for spacing, such as printer's points or picas, inches, or millimeters, two additional units of size are used:

- Two values called "average character width" and "average line height" are also used. These values are relative to the design size of the basic font, so will grow or shrink with the choice of fonts. For proportionally spaced fonts, "average character width" is a purely arbitrary dimension, and is usually made equivalent to `\em`. It is defined as a `\skip` parameter, but is not required to have any stretch or shrink. The "average line height" is usually set to `\baselineskip`. These devices are only partially a holdover from older, character-oriented systems. In fact, they provide a very convenient shortcut for writers, and are very easy to visualize.
- Percentage values may also be used in some areas. Graph sizes may be specified in percentages of logical page width and height, table cell widths also. Logical page width and height depend on columns, and whether a graph or table style specifies that the object is to be placed within the column structure, or to occupy the full page width. It is much easier to determine that a graph should occupy 75% of the page width than it is to figure out how many points, inches, or whatever, that portion represents. If the document designer changes the media for output, the graphs will still occupy the same

portion of a page as before. This works out well only because graphs are auto-scaled to fit.

5.3 Counters

The counters used to count pages, chapters, figures, etc, are all provided with default formats. These formats are attached to the counter, and used to guarantee that wherever a counter is printed, it will appear in the same format. For example, were the figure counter format defined as `\romannumeral`, everywhere that figure counter was printed, in the body of the text, in references, in the index, etc., the figure counter would always appear as a roman numeral.

5.4 Multi-column layout

CapTEX's multi-column layout produces balanced columns on partial pages, letting the format switch from single to multi-column anywhere on the page. Multiple columns can apply independently to floats, such that footnotes and figures, tables, or graphs can be within or without the column structure. Different chapter styles can implement different column structures, so the index can be double column when the text is single column. Multiple columns also apply in internal vertical mode, i.e., within figures, or captions, for example. Thus, the text in a figure may be double column, while normally it's single column.

5.5 Index entries

Based on the index style, index entries can be automatically permuted, i.e., a second index entry is created, with term and sub-term exchanging roles. This effectively cross-indexes all such entries. Section headings (but not chapter headings) are also automatically added to the index, so even without explicit index entries, a simple index is created.

5.6 ASCII output

There are two forms of output that need to be ASCII compatible: line printer and possibly screen output, and output destined for help databases. In the interests of readable ASCII copies, CapTEX doesn't try to maintain the same pagination for laser or typeset copies as it does for ASCII. ASCII output has its own table of contents, index entries, etc. TEX is overkill for this kind of thing, where any of the simpler formatters could do a credible job. However, being able to print a document in either form (ASCII or typeset) without changing the source in any way can be very useful.

Internally, CapTEX understands the basic requirements of ASCII output. Mapping all the fonts to a single ASCII font is only the first step; CapTEX also rounds all spacing and margin values to multiples of character sizes. Dimensions for graphs and tables are also rounded, as are column widths. In a sense, CapTEX

lays out the document in ASCII format, so the bulk of the work is in CapTEX, not in the device drivers.

5.7 HELP database construction

An entirely different, independent structure is required to build HELP databases, whose organization is quite different from that of a technical document. CapTEX provides the necessary tools to define a *shadow* structure, superimposed over the normal document, removing some text entirely, reordering the rest. Even this is style based, making HELP database construction a fluid and adaptable process.

5.8 Subsets

Often technical manuals are printed with only slight variants: a compiler may be available for several operating systems, and the manual may be slightly different for each system. CapTEX supports multiple distinct subsets, thus providing for manuals (and HELP databases) that vary in more than one way. The compiler manual may have variants for different operating systems, another set of variants for different computers, and another for interfaces to other languages. This is a very important feature, because it permits writers to maintain a single source document that can be customized at any time simply by selecting the appropriate subsets.

5.9 Change Bars

CapTEX supports change bars both for deleted and changed material.

5.10 plain TEX import

Although CapTEX tries to remain compatible with `plain`.TEX, inevitably some problems will occur. In particular, CapTEX changes the `\catcodes` of some characters. To make it easy to import TEX source files, or macros, CapTEX provides controls to specifically read them. This adds a final level of customization, giving layout designers complete access to TEX.

Conclusion

This has been a brief look at CapTEX, designed to provide a high level of automation for technical documents, while still maintaining typeset quality. I have tried to describe here only those areas of CapTEX that help to make it robust enough for use in large-scale production environments. Many features were simply mentioned in passing, or not at all, because they are not really different from similar packages.

FASTEX: A PC Text Editor and Front-End for TEX

PAUL M. MULLER

231 Easy Street #3
Mountain View
CA USA 94043

ABSTRACT

FASTEX is a PC text editor and full-function *front-end* to TEX. The FASTEX editor is quickly learned and yet remains powerful in expert text processing through employment of a transparent AI user-interface. A system of macros has been integrated into μTEX (David Fuchs' 2.0β) to form FTEX, the typesetting and simultaneous screen-preview system. The ArborText printer driver *DVILASER* is employed for output to any laser printer.

The FASTEX system facilitates TEX output of ordinary text without the user needing to learn this arguably most difficult aspect of TEX. Display mathematics is created according to TEX standards in a special mode of the editor. New users of TEX thereby avoid the difficult details of ordinary text typesetting in TEX and concentrate on the mathematics. It differs from LaTEX and $\mathcal{AMS}$TEX in providing clean screen proofreading (print format markers and macros are hidden) and flexible (often fully automatic) handling of *general* text presentation in the manner of a high-quality word processor, while retaining full TEX compatibility and mathematics capabilities. It is a full-function front-end and integrated system, as distinct from a special-format add-on.

The FASTEX system has been comprehensively tested[1] by a user's group of twenty technical personnel at the Ford Aerospace Corporation and Jet Propulsion Laboratory in their day-to-day work. It afforded them a rapid and time-cost-effective entry into high quality general and technical typesetting by both technical and non-technical personnel, achieving the very low learning overhead demanded in today's world by aware employers and professionals.

---

[1] This document was created, edited, and typeset by the FASTEX system, output to the Imagen *Innovator* laser printer at 300dpi using cm fonts. Other than the logos, every print format marker was hidden and in the *FLIPGUIDE*.

Paul M. Muller

Introduction

Most TEX users experience very substantial learning times when coming to the system. This is generally regarded as the key disadvantage (or price) of TEX. Its *immense* advantage and raison d'être is the ability to typeset virtually *any* technical expression regardless of complexity or notation. The FASTEX system mitigates this "disadvantage" of TEX: the high learning and mastery time.

It has also been the author's experience, shared by many, that by far the most difficult aspect of TEX for a newcomer is the complexity involved in typesetting *ordinary* text. The *TEXbook*'s introductory sample on page 24 is an example (Knuth 1984). Yet the mathematical aspects have proven for many to be relatively straightforward, with an entirely reasonable and efficient user-interface and acceptable learning time (Chapters 16–19). I have heard TEXperts say to newcomers, "The easy part is hard, and the hard part is easy."

Of course, there are immensely successful existing enhancement systems for attacking that problem such as LaTEX and $\mathcal{AMS}$TEX. So what are the differences in approach here?

Consider the page 24 example, or a typical enhancement system's text, on-screen at a PC or terminal. Another key problem emerges at once: the text is *unreadable* or at least, unproofreadable. The print format markers intrude everywhere. When we use TEX or its enhancements, we lose the valuable asset of on-screen readability one normally takes for granted in a text processor.

When screen editing, FASTEX hides (or shows on command) most print format markers, including general access to TEX macros when necessary. Yet, the system maintains compatibility with TEX and plain ASCII file structures (there are no special add-on records or files). FASTEX can edit your FORTRAN program, or include it in a TEX document (using a general `\verbatim` mode), etc.

The FASTEX system also supports fully generalized page formatting, and is not limited to special "named" layouts as with the other enhanced systems. It provides automatic font selection, line spacing, and conveniently "toggled" rather than delimited *e.g.*, `\bf{`**this is bold**`}` special handling. The user interface was designed to operate and "feel" like a text processor, rather than a programming language.

The FASTEX price is that you must learn yet another editor. This is for many a fate worse than death! The system uses AI techniques to minimize this cost. It is a "what you think is what you get" editor of innovative design, and with a unique user interface. No, we don't plug wires into the back of your head (at least not yet). It is commanded by a simple "natural" language which you mix right into text as you type or edit it; a little language which you can learn in twenty minutes to attain *mastery*. This bold claim has been successfully field-tested and demonstrated.

The editor provides both instant user-defined and permanent function keys for those operations where this is preferred, as well as a powerful macro capability which recognizes English text units such as words and paragraphs. A customized installation file for the Borland *Lightning* spelling checker is provided to support that program for those requiring this function. FASTEX is an integrated system which can and does support a wide variety of applications and user styles. There is little or no need for reference to thick manuals.

The learning time for the print formatting is extra, of course, but can be adequately covered in the rest of the first hour's time; then come *techniques of typesetting* and TEX mathematics. Our goal has been to limit the learning-to-mastery time for the editor and text processing functions to a small fraction of the total learning time in order to attain a reasonable command of technical typesetting overall. FASTEX is sufficiently simple and memorable to be efficiently usable casually, or occasionally. Yet it provides sophisticated support for the demands of expert users. The field trials in our view have demonstrated and achieved these goals.

The basic user's manual is also unusual, consisting of only 40 very small pages, arranged like an electronic typewriter's flip-guide. Two facing sample pages are provided in the Appendix, and the guide is fully indexed. The *whole* system, including all essential TEX and general text format markers, fits within this modest compass. It is therefore practically accessible to technical secretaries and others for general and technical work. The *FLIPGUIDE* is supplemented by a Reference Manual and a set of appendices.

Memory requirements for the editor are about 120K; for TEX with the macro systems, about 460K which allows 64K for the largest page of text in a 640K PC AT).

History of Development

I have been a consultant in microcomputer systems application software since 1971, and entered full time practice in 1977. In the spring of 1986, while consulting on a long-term contract at the CalTech Jet Propulsion Laboratory (JPL), I was introduced to TEX for the first time. My personal study of competing systems concluded that it was the only completely general approach to technical typesetting by the professional.

Part of any technical professional's key to success is efficiency and quality of work. This goes double for consultants, who also must produce written reports *on their own* without the support of in-house production groups. Quality of presentation counts, and the facility to produce top quality results rapidly is a strong professional and competitive advantage.

I read the *TEXbook* through twice, and must admit that I found it simultaneously fascinating, and generally opaque. The capabilities were clearly there to attack any problem whatsoever, from highly technical mathematics display, to (if resources permitted) the development of *any* or all features and facilities of a first class desk-top publishing system. As the man said, however, *power* comes at a price (and technical risk).

My background in software design and development left me with a tantalizing conundrum. Here was an amazing system, obviously conceived, designed (and I later learned) *executed* by a genius working mostly alone. Yet, my discreet inquiries to TEXpert colleagues, "Over here where the Lord Chancellor can't hear us," (W.S. Gilbert 1885) revealed that the average learning time to reasonable mastery was *six man-months*. The value of the system to just one technical division in JPL is so great that it funds and maintains a near full-time TEXpert consultant to provide training, technical liaison, and macro development for its staff.

Another key difficulty for someone approaching TEX seriously was the loss of reasonable screen readability. I had designed and developed a PC text editing and processing system of novel design which seemed ideally suited to "front end" for TEX. It had the capability to provide a clean screen, with all or most of the print format markers discreetly hidden *behind* the text characters. It was controlled and operated by a compiled syntax table which permitted fast and cheap alteration of the user interface without reprogramming. It also provided all of the basic, and most of the advanced features and facilities of a normal text processor. All of this operated within a normal ASCII file structure, and was therefore capable of being read by a TEX macro system.

A preliminary design structure for an integrated editor and typesetting system was prepared. It utilized the editor as the front end, TEX with a suitable *front end* macro set as the print (typeset) output formatter, followed by a suitable driver for laser printers.

I made contact with Addison Wesley, who put me on to David Fuchs, the cognizant programmer of μTEX version 1.5A which they were then marketing. The plans were discussed among us all, and there was a consensus that this was worth pursuing. David Fuchs agreed to make two key changes to μTEX so that this approach could be supported:

- Unrestricted paragraph length, [CR] to [CR]
- Access to ASCII 128–255 as active characters

The first was necessary because floating text paragraphs, where [CR] means "end of the paragraph" and not "end of the line" is a fundamental characteristic of text processing, in feel, style, and substance. The second provided the ability

to use the editor's QKEY system for rapid keying of PC foreign and special screen characters. David incorporated these capabilities in his μTEX version 2.0β. It was intended that this system would be the next version published by Addison Wesley.

My preliminary design had estimated that six man-months of my time would be required to achieve level one: the creation of an FTEX macro set which provided all the relevant text processing capabilities of FASTEX while maintaining TEX compatibility for mathematics typesetting. Between May and November of 1986 this was achieved by working evenings and weekends. The FASTEX system was placed in β test by myself and one senior colleague at JPL.

We determined to conduct a test of the user interface quality and accessibility. The first *target* user had never before used a microcomputer. He had what he described as "desperate need" for a quick turnaround, high-volume, desktop technical typesetting system. It had to go into effective operation with a minimum of learning time. Six man-months was out of the question; he could afford about a week. As a compensating factor, this gentleman is world class, from start to finish.

I provided the system on a Monday morning, with a one hour introductory, at-the-keyboard hands-on tutorial. We had agreed that he would then be left alone for the rest of the week. His goal was to outline, write, input, edit, proof, and produce a significant technical memorandum in that time using the system, beginning from scratch. The only written support document available was the first version of the *FLIPGUIDE*; see the Appendix for a sample page. He had need of six short consultations with me during the week in areas not adequately covered by the existing documentation. The task was completed and published as a JPL Engineering Memorandum on Friday. We judged the trial a success, albeit then a statistical sample of only one user and document. He has continued to use the system daily down to the present time in all of his technical work, and was the first member of what is now a 20+ strong user's group.

There were three basic things he had to learn. First, the editor. This is comfortably done in an hour to essential mastery (there have been a number of successful trials on this). Second, text processing. He had no experience in direct text processing on a microcomputer, and there are many fundamentals and important *niggles* involved in those skills. This had to be extended to typesetting at least to the degree covered in the documentation. Third, he needed to extract the necessary TEX display math formats from Chapters 16–19 of the *TEXbook*.

This illustrates the three key elements in the design approach. The editor is easily learned, and must be, because everybody hates a new editor. This is essential to providing the clean screen handling which I feel is key. We then take advantage of the fact that many people already have text processing skills and experience of relevance and value. This largely avoids what is for many, the *hard* part of TEX. Finally, we make good use of TEX where it shines best,

in mathematics, which FASTEX accesses transparently. It is a combination and integration of three factors:

- A quickly learned yet competent *clean screen* editor
- Existing text-processing skills and environment
- The *mathematics* of TEX (Chapters 16–19)

My consultancy then shifted to a long-term contract at the Ford Aerospace Corporation. The first task was to participate in a rapid design and documentation effort with very tight deadlines. The system worked well for me in that environment, and caught on in the company among related technical personnel. The development continued, bringing successively more features into the convenient user interface represented by FASTEX. Version 1.00 was frozen and released in May of 1988 to the user's group. This milestone was two years in the making, and represented the group's belief that the system had achieved a level of completeness and reliability comparable to commercial version release software.

We believe that this is an efficient, capable, and effective approach to practical technical typesetting for practicing professionals. It possesses reasonable learning-times, high memorability, applicability, and TEX compatibility.

Technical Developments

There are a number of technical developments embodied in the system which may be of interest. It is the purpose of this section to discuss each in the context of the overall systems integration.

The FASTEX editor was written in the C language for transportabilty and its unique combination of macro and micro code levels. It has operated successfully on every close PC compatible, and nearly all less compatible computers, from the Sanyo 550, and Olivetti M21 (ITT 6300), to the latest DOS 80386 systems. The editor uses only normal DOS screen calls, and functions in all color and monochrome environments. The editor's structure is shown in Figure 1.

The *Syntax Table* is an ordinary text file prepared under the rules of an AI language called SXPP. It is a top-down parsing system with returns. Its function is to provide the logical instructions linking user input and program subroutine calls. An unlimited number of "states" or *modes* can be supported transparently to the user. It is relatively easy to design, create, and modify a very complex user interface, and it can be done by changing only the syntax table's text, recompiling it with the SXPP compiler (into C), compiling the C into relocatable object code, and re-linking the program executable. No changes to any program C code are necessary unless one wishes to introduce an entirely

FASTEX Editor Structure

- SYNTAX TABLE [Logical Instructions for Program Operation]
- PARSER [Operates Program from Compiled Syntax Table]
- USER INTERFACE [Created by Above to Process User Input]
- SUBROUTINE LIBRARY [Actions Executed by Parser's Calls]
- SCREEN OUTPUT [Scans Text Memory to Support Dynamic Screen]
- FILE I/O and FASPRINT Output

Figure 1

new executable feature via a new subroutine (which would then have to be programmed). A sample is provided in Figure 2.

Syntax Table

```
varname  :(subroutine) /* Invokes Subroutine */
gx       :"^g" (goag) /* Go in current direction */
g1       :<"+D" (goed) /* Part of syntax for possible go */
```

Figure 2

The *Parser* is the almost trivial "main" program. It executes the compiled syntax table based upon processing of user keyboard input. The *brain* of the AI system is the logic contained in the syntax table itself. The *functionality* is contained in the *Subroutine Library*.

The *Screen Output* routine is very sophisticated, as in any effective text processing program. It is built on a multiple-level, prioritized, logical interrupt scheme which guarantees that the most important operations are done first. It may not be generally recognized that screen updating based upon user keyboard inputs is a very difficult task. One might assume that 10 keystrokes per second would be trivial. It is manifestly not. The text in memory must be shuffled about, "gassified" and "degassed", examined for a great distance above and below the cursor, with color and display characteristics determined in real time on the basis of complex algorithms and instantaneous user input.

The problem is made worse for FASTEX because it was determined from the beginning to maintain ASCII file compatibility. FASTEX is, therefore, a memory editor, that is, the whole of text must fit in memory (about 400K bytes available in a 640K PC). The screen handler must examine a large body of text in memory with all of its print format markers before the screen layout can be computed. An attempt has been made to obtain the highest practical level of display readability (and *what you see is what you get*) within the limitations of a non-graphics (character-oriented DOS compatible) protocol.

Screen Features

- Print format markers are hidden *behind* text characters
  Their character shows in a colored box
  Appear on screen-bottom when cursor is upon them
  "Show" screen display mode to reveal all
- CR and TAB unambiguously shown when cursor on them
- Screen scroll is automatic, vertical and wide horizontal
  There are no "scroll" keys or commands necessary
  Four-line context top/bottom on PgUp PgDn keys
  Approximate wrap-warning in large font sizes
- Type styles show in various colors (or b/w) enhancements
- Indentation, hanging indents, tabs, and wide columns display
- Numerical data display:
  Current font's typesize
  Current typestyle(s) (in addition to the color highlight)
  Estimated words above cursor
  Remaining text space
  File under edit and operational information

Figure 3

The screen handler shows on-screen all the text features listed in Figure 3. As Pooh would say, "it takes a lot of pencil to do ... that " (Milne 1926). Without this ASCII file compatibility, however, it would have been impractical to create an interface to TEX. Likewise, support for an unrestricted number of print format markers was essential.

File I/O is entirely standard. The only niggle is that line feeds must be removed on input, and restored on output. This was necessary so as to maintain

a 1:1 relationship between characters in memory and characters typed and processed. It is possible to take "quick prints" of any text units, such as paragraphs, to an attached dot or daisy printer (or a laser emulation of same) directly from the editor, while editing. This is a generalization of the screen dump commonly provided by text processing editors. It is also possible to produce a loosely formatted, rough-draft text-processor style printed output on an attached common printer.

The normal print output mode is to leave the editor and invoke TEX *i.e.*, FTEX and perform a typeset to ∗.dvi with simultaneous screen preview. This is then delivered to the laser printer via a suitable driver program.

Font support and development were significant. A reasonably convenient system should make font invocation transparent to the user. He or she wants to simply name a type size, face, and/or style. This needs to be conveniently changeable at any time. The normal protocols followed by quality text processors were adopted.

This was not easy in TEX for several reasons. TEX expects one to define each font, size, style, super/subscript sizes, and face individually. The existing add-ons *e.g.*, LaTEX provide macros such as "`\big`" and "`\Bigger`" and so on, but it was still necessary to think about font details. Furthermore, TEX is inherently an explicit language system; one is encouraged or required to write constructs such as {**`\bf`** This is Bold} rather than the more natural "toggled" user interface for such functions *e.g.*, [B]This is Bold[B] where [B] is the **bold** key or format marker. Similar remarks apply to centering and many other functions.

Any text processor-*cum*-typesetter really must provide all of this font structure transparently. FASTEX therefore had to adopt a TEX-compatible scheme to access most of the nominally available fonts and styles. The maximum range of sizes is 5–42pt, that is, 5pt base, to 17pt at **`\magstep5`**. These must be linked to super/subscript sizes.

Typefaces in Roman and Sans Serif are commonly provided, and both are essential to a basic environment. The math fonts must automatically follow along with the adopted text size. Likewise the type styles.

Type styles *e.g.*, this italic, are used fluidly in most text processing applications. Worse, a *common* printer is able to *combine* styles at little or no cost in equipment or complexity. No problem with bold, expanded, double-strike, underlined, italic on the lowly Epson FX80!

Professor Knuth notes for starters, that one needs a special font family to do proper underlining. The minimum reasonable standard set of font styles was determined to be *italic*, **bold**, *slanted*, and `typewriter`. The minimum sensible number of *combinations* was deemed to be any two; we therefore support all pairs: ***bold italic***, ***bold slanted***, **`BOLD TYPEWRITER SET AS CAPS SMALL CAPS`**, *`italic typewriter`*, *`slanted typewriter`*, and last but not least, the

"funny italic" font, as *Slanted Italic*, to round out the scene and make most of the standard TEX distribution fonts available to direct calling.

It is left as an exercise for the reader to estimate the number of font files in the FASTEX system. The base faces Roman and Sans Serif together with bold expanded, italic and slanted are supported at all magsteps between 5 and 42 points; 5, 6, 7, 8, 9, 10, 11, 12, 14, 17, 20, 24, 30, 36 and 42. The math faces and combined styles are supported for the set 5–24pt. All of these except the combination styles are supported for super/subscript sizes related to base size. The screen previewer provides a full set of all fonts between magstep −2 (demagnifications) and +5 magnifications. The laser output is supported between magstep 0 and 5. The disk load overall is just under 10Mb, with well over 1000 files in the system. It required over two *weeks* of PCAT computer time to generate the font files which were not part of the normal TEX distribution package using METAFONT!

The ease with which they can be invoked belies the complexity and depth of the programming. To specify a font size: `^f12^f` for 12 point; for a face change to Sans Serif: `^f\setsanserif^f`; and for a style: `^u` for "underline" which typesets as slanted or `^fB^f` for boldface. The system also supports both normal bold, and expanded bold, at user selection or change, so we can print in **Both** styles **Both**. The control key `^f` is used to delimit all `FONT`s, `FORMAT`s, and `FILENAME`s in the system, which are named as briefly and memorably as possible.

In addition to this, up to five fonts can be user-defined for convenient invocation by the unused font numbers 0–4 as above, including super/subscript sizes. In this way one can pick up a little Dunhill for example, and cover virtually all the remaining standard TEX fonts, or a reasonable number of proprietary fonts which might be acquired. For the really adventurous, or if all this is not enough, the general and unlimited TEX formats can be used.

FASTEX reserves only the truly "free" characters {, }, \ and |. The last is the FASTEX mode "comment" character. This leaves the remaining TEX reserved keys as ordinary text, which is convenient for the user: ~, #, $, %, ^, &, _. To use TEX standards, one invokes `\dotex` mode *e.g.*, for math display. Most TEX formats (except display math) can be used in the FASTEX mode, so the user rarely needs to think about it unless something really deep in TEX is required. There is a convenient and fully complete `\verbatim` mode covering all ASCII characters between 32 and 255_{10}. It is easily turned off (the classical dilemma) by any `^f` format (`^f` being ASCII 6).

Super and subscripts, accents, and most special characters including all non-graphics PC screen characters, typeset in all modes under the same convenient structures. These include QKEYS which type in as `qe` for example which yields é on screen and in text. There are 128 of these covering the non-graphic PC screen characters, plus the basic "quarter-box" graphics set. The super and subscripts will wrap and justify in text modes exactly as one expects in a text

processing environment (again, no problem for our Epson, but a real challenge in typesetting with TEX). Gendered quote marks are automatic, and in typewriter type, the genderless ' and " are substituted.

Referencing to the bibliography herein has been by (Author date). However, reference numbers can be called for by unique user-chosen "context-names" defined by their citation *e.g.*, `\TEXbook` at the first occurrence using the automatically numbered reference scheme in FASTEX. Then to reference Knuth, I might have keyed (between superscript toggles) "`\TEXbook`". One can cite the number in *any* format, style, or font; *e.g.* as [`\TEXbook`] → [3]. End notes are also separately supported, as are ordinary references, bibliography, table of contents, forward and backward page and section referencing, as well as context-named footnotes. When you are prepared to pay the price in macro programming, TEX is absolutely wonderful, and uniquely powerful, with almost infinite potential.

There are many other examples of the general policy to establish automatic features wherever practical, and to give the normal user what is expected as the default, reserving special handling to special requirements.

Natural Language Editing

It is the author's opinion that the user-computer interface is often the weak link in software, and that solving this problem justifies careful thought, planning, design, and sensible innovation.

Frequent reference to thick manuals is very unappetizing. An intuitive, natural language approach is preferable, supported by on-line context-sensitive assistance. FASTEX carries this to what we feel is near the cutting-edge of the state-of-the-art. The user of such a system becomes deeply involved in a new style of software. It is therefore necessary also to support the more familiar *e.g.*, function-key style of editing and working, as has been done.

What is the typical software's user-interface? Each command, task, format, and operational mode is given a name and/or menu location. Pull down, push across, mouse or arrow keys notwithstanding, this is to my mind a pain in the brain. Either I cannot remember the name, or I get lost in the menus. But we will let this pass for now.

What about program capabilities? How do I find out (especially in the beginning) precisely or even crudely what the program can do? Tutorials can begin to answer this need, and there are many very professionally engineered systems out there which give one a reasonable shot. Nevertheless, the 300-page manual with 500+ concepts is typical, even for a text processing editor. I find it tedious to read a 300-page manual, and digest it, *before* the functionality can be appreciated. Can it do this or that? Answering such a question usually forces

one to the manual, with perhaps a 50/50 chance of finding it even via the rare excellent index.

Is there an alternative? It is suggested here that there is. Consider the creation of a "what you *think* is what you *get*" text editor. I don't mean WYSIWYG display; I mean just what it says. You are in the middle of an editing session, and you want to *go 3 paragraphs*; exactly 3 paragraphs, and be at the beginning of the third one below you. Arrow keys will do the job, to be sure, inefficiently. So will a "go paragraph" function key (and you can install such easily in FASTEX if you wish *while editing*).

Well, if "*Go 3 Paragraphs*" is what I think, then I will key " ^g 3 P " (Ctrl g, 3P, case *in*sensitive), and the editor dutifully "goes three paragraphs." Same for sentences, words, and chapters. I merely translate my thought into the natural language of the editor, and it *flows* out the fingers onto the keyboard, at full typing speed, mixed right in with text, transparently. The program simply does it, on key, on command, on thought.

Text processing is all about language. There are five, or maybe seven different meanings for the hyphen key; several kinds of spaces; and so on. The simplest things become complex. That is because our language is so subtle and powerful. We took at least ten years to learn it well. We are linguistic creatures; we think and analyze linguistically. A language-based, user-computer interface is entirely sensible and practical. It additionally offers many indirect benefits by analogy with, and derived from, our linguistic abilities and experience.

Here, now, is the complete "language" of the FASTEX editor, with *nothing* left out. There are five verbs (actions): GO, EXAMINE, DELETE, PUT into, and SEARCH for, respectively the control keys, ^g , ^e , ^d , ^p , ^s. If you count program controls, there are three: ^t to terminate an action; ^o for the MENU of options; and ESC for you know what.

There are seven basic nouns (names for things): WORD, SENTENCE, PARAGRAPH, CHAPTER, DOCUMENT, BAG, ALPHABAG, respectively the case-insensitive W, S, P, C, D, B, A. In addition, there are six special nouns: ^f ...^f to "enclose" or delimit FONTs, FORMATs, and FILENAMEs; ^r ...^r to delimit replacement text; L for a single character (Letter) of text; O to name an output device; –^d the "previous" deletion; and { } to enclose the text of an editor's macro (we call it a UTILITY).

There are two adverbs: "+" meaning "forward" or "to the end of"; and "–" meaning "backward" or "to the beginning of". One adjective completes the syntax: *e.g.*, **3**, an integer number.

Finally there are two *order* rules. One PUTs *into* the destination object *from* the source object. Second, numbers come *before* the object modified. That's all, folks. Except for perhaps some of the *special* nouns, which fit naturally into the scheme later, you have already learned the editor!

In the traditional editor, you wish to search for something; so what happens? You hit an access key for the menu (it pulls down flashy menus amid pretty colors); but it's the wrong one, so you hit another key or two to get the right one. Then you mouse or key-select the **SEARCH** option. You are then prompted for the search text (CR and some other characters or format markers are probably not allowed). Then you release the search text (with our old friend CR like as not) and the program finds it. Oh, sorry, wrong one; you want the next one down, or the third or fourth *preceding*; back to the menu, and like as not, key in the search text again, under the **SEARCH BACKWARD** menu option.

Well, perhaps this is all just a bit too pessimistic. It must be admitted that *some* programs make it a bit easier.

Can you *guess* how to search for, say, "find me" in FASTEX without being told? Stop here and try to *intuit* it; use the natural language approach. Think "search" for "find me", and let it flow to the fingers. Don't peek now; really, try to do it "naturally" without condescension to the "program" or programmer. Just let it flow. I'm going to tell you now, so stop peeking! Have a go!

You think "search" and it's obviously **^s**, then "find me", so **^s find me**... is flowing down towards the fingers, but you stop with a jerk after the "me"; what next? You're all finished, and the computer doesn't know it, stupid thing! Well, until we can plug wires into the back of our heads, you're going to have to key something else that means "I'm done, dummy." After a pause, the *novice* correctly keys "**^s**" completing the thought, and FASTEX dutifully goes off searching. If it fails, it leaves the cursor where it was, otherwise the cursor is cleanly on the found text, with no menus littering the screen; no fuss, no muss.

The *expert* probably didn't even pause, and keyed "**^s find me** CR" (and is still waiting) with the command line at screen-top staring back at him:

```
^s find me <CR> _
```

without a care in the world. Perfect example of culture shock. I have watched this very thing happen many times while giving hands-on training or user response evaluations (there's material for a paper in that too, I'm thinking).

Of course, "find me" is in "quotes" both on paper here, and in your head; and **^s** is the "search" quotation mark. So it's **^s find me^s** (no CR). Then, of course, it's the wrong one; probably next along. So you want the Same Search again, and the command line is empty. Don't key in the search text again; think: Same Search **^s ^s**; then key it! To be fair, most users have to be asked the question "how do you make the Same Search over again" before they will guess **^s ^s**.

To complete our task as originally posed, "Go Backwards Same Searching", will probably require us to look in the *FLIPGUIDE*, or have a consultation with the AI-based HELP system F2. We can find out how to do this by typing the

question in plain English: "How do I do the same search backwards?" whereupon the answer is provided: "**^g - ^s ^s**." In our experience with users, you will never have to ask again; it will flow naturally into the fingers next time with little pause for thought, and after the tenth time, it will come naturally *without* thinking about it consciously. It is a fact that when I use FASTEX, it is now entirely automatic, and indeed has become for me my best computer friend, my "what I think is what I get" editor. Next best thing to wires in the back of my head.

It might be concluded from the simplicity of the foregoing that, "Yes, it's easy to learn, but it isn't powerful; how can it be with so few commands?" Sophisticated and exhaustive user experience has shown the contrary. The language itself is simple, but the fact that *all* meaningful "sentences" (and even whole paragraphs in Go-able utilities **^g** { ... }) will work, means that thousands of distinct practical and useful editing constructs can be created and will be supported without complaint by the program. The user's group has joint experience of more than a score of editors, and FASTEX stands up as well as the best to heavy demands from the expert. This can only be fully appreciated in use.

Thank you, dear reader, for bearing with me through such an Odyssey. I *trust* you not to think me unprofessional for taking a perhaps rather light approach to what is a serious subject, worthy of careful study and development. There it is, then, a taste of *natural language* program interfacing. The vast majority of our developmental (β) users and guinea pigs have enjoyed learning the FASTEX editor; even those experts who became frustrated by the *complete* absence of command delimiters like [CR] which they have used ubiquitously for ten or twenty years. It *is* not only possible, but normal, for a new user to learn and master this editor in less than an hour at the keyboard. Many have learned it by reading the *FLIPGUIDE*'s first 20 pages (about 10 minutes) and then hitting the keyboard with a real task.

User-Computer Interface

The clean screen approach to how we relate what is keyed in to the display, and typeset output, is the other key raison d'être for the system. It is useful to provide some examples of the basic structures and philosophy.

In general, there has been an effort to make the screen look as clean as possible, and as much like the typeset output as practical within the limitations of a non-graphics, character oriented, DOS-compatible screen handler. Wherever possible, two keystrokes only are used for Greek, special, and foreign (accented) characters. Particular emphasis is laid upon displaying screen characters which are the same as typeset, or remind one easily of what is represented. This was done to make proofreading as reliable and easy on the eyes and brain as possible. Fatigue in handling complex computer input and display is a key factor for many

users, and we have confidence that this has been reduced to a near optimum and minimum by the combination of techniques employed.

The most basic screen functionality is the "hidden format." The FASTEX editor *hides* any character sequence between `^f` ... `^f` behind the character upon which the cursor rested when it was entered. For example, if the cursor is standing under the w in [w]ord as here, and if one keys for example, `^f 12 ^f` in order to set 12 point type, then on the screen, the w is displayed in the "enhancement color" or "fuzzy box" to show that a print format is lurking behind it. Any number may be placed behind a single character (we tested the program with thousands). If after typing in the font change format, the ← key is struck, the cursor remains under w but is now "on" the format, and it shows on the screen at the lower right in a special reserved area. The `^d` key will now delete it, for example, just as the [DEL] key would have deleted it just after it was typed (even though invisible from that cursor location).

All this seems much more awkward when explained than it is in practice. One quickly learns to notice the little (blue) boxes (in my color selection) which indicate the presence of a format, and then easily move the cursor upon it when necessary to delete or manipulate it etc. A quick trip to the Menu of options `^o` (one of its few regular uses) followed by `S` will change the screen to Show Mode, when all print format markers are revealed as they would appear in a normal editor with no hidden marker capability *i.e.*, like a normal TEX screen display. A quick `^o H` or `^o W` will restore `Hide` or `Wide` screen mode; there is no need to stop on the menu. In practice, one almost never uses the show mode, though it is occasionally useful if a format has gotten lost or forgotten.

We will conclude this section, and the document's descriptive text, with some examples of keystroke input, versus screen display, versus typeset text. This is presented as Table 1 in a 3-column format. The first column is keyboard input. The middle column is screen display, shown in typewriter type; the boxes around characters show the screen's enhanced color or fuzzy box highlighting of format markers lying behind it. The screen display is fixed width, of course, whereas the boxes shown here are somewhat wider than the character space. The third column shows the typeset output corresponding to the input.

Incidentally, tables typeset in columns are quite clean in this system. One can mix and match lines with any number of equally spaced columns, and can introduce or change unequal width columns at any time. To obtain the three-column form of Table 1 below, it was only necessary to specify: `^f TAB=3^f` beforehand, and `^f TAB=^f` afterwards. The tabline at screen top shows the [TAB] areas split into 3 columns (extra wide beyond 80 columns in wide screen mode to allow plenty of space for text), so that screen and typeset output can often look very much alike. Each line is merely preceded by `^a` to effect alignment. If omitted, a normal line is typeset, etc. For unequal columns:

`^f \samplecols^f` Text_1 [TAB] Text_2 [TAB] ... Text_n [CR]

Keyboard	Screen	Typeset
1) [TAB] This is text.	1) This is text.	1) This is text.
^f B^f Bold ^f B^f It.	[B]old[ ]It.	**Bold** It.
qb qe Qv ^q# qB QKEYS	β é | £ • QKEYS	β é | £ • QKEYS
A ^b^b is "emspace"	A[ ]is "emspace"	A is "emspace"
~ # $ % ^ & _	~ # $ % ^ & _	~ # $ % ^ & _
\$math^b e^f S^f-x^f S^f\$	\$math[ ]e[-]x[\]$	*math* e^{-x}
Text it e^f S^f-x^f S^f.	Text it e[-]x[.]	Text it e$^{-\mathrm{x}}$.
^f \greek^f D	[D]	Δ
^f \subst^f R	[R]	$\Re$
^f \hat^f qd	[δ]	$\hat{\delta}$

Table 1: Keyboard vs Screen vs Typeset

to create the unprinted sample column line, then the ^a to begin each aligned line. The user can modify the tablines installed in FASTEX to reflect frequently used (approximate) screen tabs for unequal width columns, or select simultaneously ^f TAB=1^f which provides several wide tabs for general \samplecols work. This is an important example of screen handling intended to minimize the number of occasions when the screen must depart materially from the output.

Plain tabs also work for both ordinary indentation and simple columns. Each tab measures from the position of the last, so one merely counts tabs (not screen position), and quick alignment is possible. The first example in Table 1 is the simplest case, where one is using item numbers. This emphasizes that [CR] and [TAB] (even though tabs are an abomination which, unfortunately, will always be with us) are treated as in a text processor. The first introduces vertical white space, and the second, horizontal.

Summary of Capabilities

FASTEX Editor:

- Full function text editor with text processor style and feel
- Proofreadable clean screen edit; print format markers hidden *or* visible
- Screen: automatic scrolling, PC special characters and font information show
- Quick learning, memorable, yet has the full power an expert expects
- Short yet complete *FLIPGUIDE* "electronic typewriter" style manual
- Strict ASCII file structure; no conversion programs needed
- Editor operated by compiled syntax table; user interface is via an AI parser
- Therefore no programming needed to change/update user and TEX interface
- Relatively cheap to alter or expand the user interface and capabilities
- AI "natural-language" editing: "What you think is what you get"
- AI on-line interactive help answers questions typed in plain English.
- "Knows" text units: Letter, Word, Sentence, Paragraph, Chapter, Document
- Supplied and Instant user `FUNCTION` keys while editing; powerful edit macros
- Flexible, natural, multiple cut and paste; alphabetic sorting
- Undelete (editor's last 9 deletions)
- Convenient, clean screen display of many European accents and specials
- Quick-type QKEY keyboard for all PC and many foreign characters
- Instant QKEY finder/reminder help screen; TEX symbol keyboard
- Quad, em, en and thinspace quick-keys; neat screen display as fuzzy space
- Screen column tab markers; very wide screen column and text display
- Common printer text-processor style rough draft output (not typeset)

FTEX Typesetter:

- Screen preview of pages while typesetting; HP and Imagen laser outputs
- TEX compatibility; all macros available
- [CR] is "natural" Carriage Return and puts white space on the page
- Also [TAB] is "natural" and sets "quickie" variable width columns
- Easy fixed width, decimal, and unequal "sample" width column alignment
- Handles 92 base ASCII and 128 upper-ASCII characters (8 bits)
- Typesets PC screen characters; verbatim typesets *all* 96 ASCII
- Accent and special character "keyboards" OK in all TEX and FASTEX modes
- Automatic leading/trailing "quote" marks from ["] key; widow elimination
- Free text [^c] centering and/or [^c] flushing like this
- Free text "toggled" **Boldface**, *Italics*, *Slanted*, and `Typewriter`

- Also combination pairs: ***Bold Italic***, **Bold Slanted**; *Slanted Italic*
- And *Italic Typewriter*, *Slanted Typewriter*, **BOLD TYPEWRITER**
- Free text "toggled" Super script and Sub script and it all wraps and justifies with all expected text processor features such as automatic typesizing and specials $\beta\delta$ etc
- Conditional hyphen; – break/nobreak dash and minus shows on-screen
- Absolute page break; conditional page break (request "n" lines remaining)
- Automatic Roman pages to 4999; automatic CH.PG style page numbering
- Unlimited free selection of one-line, multi-line, and odd-even head/footers
- Justification on/off; hidden labels and messages to the FTEX typeset screen
- Convenient quick-key hanging or block indents; show on-screen
- Centered "poetry" style paragraphs; square paragraphs (see Abstract)
- Quick-key typeface changes: Roman, Sans Serif, FIVE User defined Fonts
- **ORDINARY BOLD** (cmb) and **EXPANDED BOLD** (cmbx) selectable
- Automatic numbered footnotes, plain and context-named
- A4, A5 and Letter page sizes plus unrestricted user-defined page sizes
- Landscape and portrait orientations; text-overlay typewriter-like "esc" key
- Automatic/manual line spacing *e.g.*, 1/2, single, 1 1/2, double: (8, 12, 16, 20pt)
- User-defined margins; kern, raise, lower; Push-down text to page bottom
- Floating top, mid and page inserts (including basic plot/graphics work)
- Automatic DOS date in 3 formats: 11–22–88; 22–11–88; November 22, 1988
- $ & # ^ % ~ _ are text characters in FASTEX; only { } \ | are reserved
- Style sheets and TEX macro definitions with TEX compatibility
- Continuous underline ______ from _ _ _; any mode Underline and overline

out-left • Marginal notes and insertions as shown here out-right

- General `\fill` + + + + + + + + + + + with any non-alphabetic character
- Math text in FASTEX; FASTEX to/from TEX toggle; TEX display math mode
- Automatic table of contents; end-notes; bibliography
- Automatically numbered references by context-name
- A basic plot and graphics facility (example below)
- File merge; basic graphics; basic plotting
- Automatic direct and hidden indexing
- Quick-key convenient changes of type sizes from five to 42pt

Under development:

- Page and section forward/backward references by context-name
- Multiple column pages typeset; lined tables
- Parsing all TEX macro names to *prevent* input errors (via editor's parser)

Bibliography

1 Knuth, Don. E. *The TEXbook*. Reading, Mass.: Addison-Wesley. 1984.

2 Gilbert, W.S., and Sir Arthur Sullivan. *The Mikado*. London: Macmillan. 1928.

3 Muller, Paul. FASTEX *FLIPGUIDE*. Privately published. 1986–88.

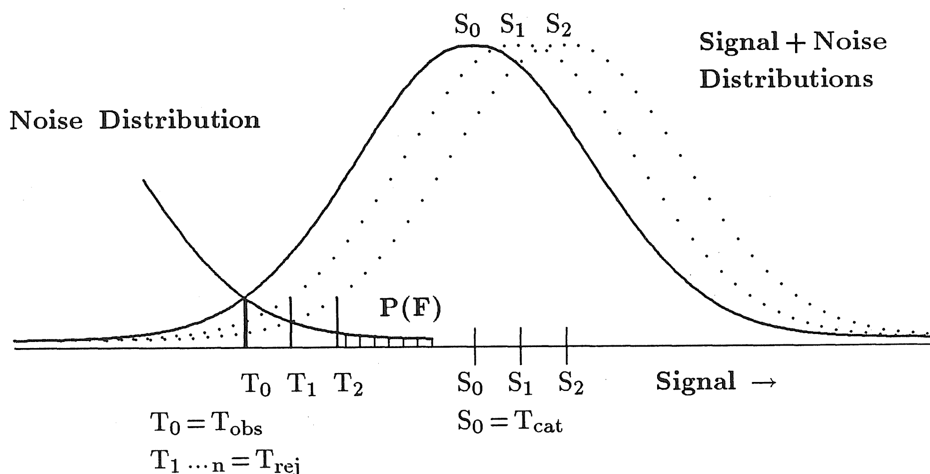
4 Milne, A.A. *Whinnie the Pooh*. London: Methuen. 1970.

Acknowledgements and Notes

Hardware Requirements: FASTEX editor – PC 256K with 2 floppy drives. FTEX and *DVILASER* – 640K, 10Mb hard disk space, EGA or Hercules.

Thanks to: Mr. Peter Scott, Mr. David Fuchs, Mr. Kent Russell, and the FASTEX Users Group for constructive criticism, bug finding, suggestions for improvement, help in understanding TEX, and many professional courtesies.

An Example Plot



Appendix

Sample Pages From FASTEX *FLIPGUIDE*

SCREEN EDIT MODES Page 17

- The SCREEN can display in SHOW, HIDE, & WIDE, modes. Select them from `^o` OPTIONS MENU. You may set INSTALL to begin with whichever you wish.
- SHOW mode "shows" on-screen text with the PRINT FORMATS exactly as keyed in *i.e.*, unhidden; wraps line-ends to suit the screen. Look only; don't edit!
- HIDE mode hides PRINT FORMATS and limits line wrap to lesser of (1) screen width[1]; or (2) *approximate TEXSET* line break as warning in wide[1] fonts.
- A PRINT FORMAT "hiding" means letter following is shown in "examine" color. Move CURSOR to it ←→ or `^s`; see it screen lower-right; `^d` it, type in front, etc.
- WIDE mode: "infinite" width wide-format for COLUMN work *e.g.*, `^a` ALIGN in *TEXSET*. [ALT]F8/[ALT]F9 select tablines for Hide/Wide using {58}/{59}.
- [1]If a screen line "wraps" short, you are in a *wide* font size. If off screen-right, you are in WIDE mode. Horizontal scroll is automatic, just move CURSOR there.

FASTEX FUNCTION KEYS Page 18

- FASTEX comes with some FUNCTION keys (left) and ALT keys (right) pre-set:

Key	Function	Key	Function
F1	Go Top of Document: `^g-D␣`	ALT –	"en" DASH non-breaking (p. 38).
F2	HELP from Menu: `^o?`	B	`^f\inbox^f{` [put in text & close}].
F3	Go Word Left: `^g+W-2W␣`	F	`^f\fill^f` ["fill" character next].
F4	Go Word Right: `^gW␣`	G	`^f\greek^f` ["greek" character next].
F5	Delete Word Left	O	`^f\oline^f{` [put in text & close}].
F6	Delete Word Right	Q	QKEY keyboard from HELP screen.
F7	Delete to start of ¶ (LINE)	R	`^f\ref^f{` [put in text & close}].
F8	Delete to end of ¶ (LINE)	S	`^f\subst^f` ["subst" character next].
F9	First File Save: `^p^f^f D.`	U	`^f\uline^f{` [put in text & close}].
F10	Go End of Document: `^g+D␣`	X	`^f\index^f{` [put in text & close}].

- See *REFMAN* for details. OK to replace with your own. [AF1–7] Spell-check and reserved; [AF8–9] Hide/Wide screen tabs; [AF10] Go Current Utility {...}.

Participants, 1988 TeX Users Group Meeting

McGill University
Montréal, Canada
August 21–24, 1988

Notes: 175 participants
* indicates exhibitor
† indicates presenter

Clifford Alper
TeX Users Group
Providence, Rhode Island

Bernadette V. Archuleta
Los Alamos National Laboratory
Los Alamos, New Mexico

Mary Armstrong
TeX Users Group
Providence, Rhode Island

Ronna Bailey
NCAR
Boulder, Colorado

† **Elizabeth Barnhart**
TV Guide
Radnor, Pennsylvania

Michael Barr
McGill University
Montréal, Québec, Canada

Karen T. Barry
Martin Marietta Energy Systems, Inc.
Oak Ridge, Tennessee

Robert P. Batzinger
United Bible Societies
Chiang Mai, Thailand

† **Stephan von Bechtolsheim**
Integrated Computer Software, Inc.
West Lafayette, Indiana

Nelson H. F. Beebe
University of Utah
Salt Lake City, Utah

Barbara N. Beeton
American Mathematical Society
Providence, Rhode Island

Marc Blanchet
University Laval
Ste-Foy, Québec, Canada

Chris Bohn
Personal TeX, Inc.
Mill Valley, California

Cathy M. Booth
University of Exeter
Exeter, England

Virginia Ann Brower
Stanford Linear Accelerator Center
Stanford, California

Lonnie Brown
Shepard/McGraw-Hill, Inc.
Colorado Springs, Colorado

John Bruce
Digital Equipment Corporation
Nashua, New Hampshire

Justin Bur
University of Montreal
Montréal, Québec, Canada

Mimi Burbank
Florida State University
Tallahassee, Florida

William P. Butler
TeX Users Group
Providence, Rhode Island

Alain J. Cadorette
TV Guide
Toronto, Ontario, Canada

* **Lance Carnes**
Personal TeX, Inc.
Mill Valley, California

Chris J. Carruthers
University of Ottawa
Ottawa, Ontario, Canada

François Chahuneau
Berger-Lavrault
Neuilly, France

† **S. Bart Childs**
Texas A & M University
College Station, Texas

Esther C. Clerin
University of Ottawa
Ottawa, Ontario, Canada

David M. Cobb
Science Applications International Corp.
Oak Ridge, Tennessee

Arvin C. Conrad
Menil Foundation
Houston, Texas

Edgar Cooke
Software Research Associates, Inc.
Tokyo, Japan

Mary Coventry
University of Washington
Seattle, Washington

Jackie Damrau
University of New Mexico
Albuquerque, New Mexico

Dian De Sha
California Institute of Technology
Pasadena, California

Michael DeCorte
Clarkson University
Potsdam, New York

Andrew Dobrowolski
ArborText, Inc.
Ottawa, Ontario, Canada

Michael Doob
University of Manitoba
Winnipeg, Manitoba, Canada

Marcel Dupras
Université Laval
Ste-Foy, Québec, Canada

Allen R. Dyer
Computer Law Laboratory
Ellicott City, Maryland

Mark Edwards
University of Wisconsin, Madison
Madison, Wisconsin

Carl J. Egetter
Lockheed Aeronautical Systems
Burbank, California

† **Shawn Farrell**
McGill University
Montréal, Québec, Canada

† **Michael J. Ferguson**
Université du Québec à Montréal
Verdun, Québec, Canada

Frank Flynn
University of British Columbia
Vancouver, British Columbia, Canada

Barbara Forrest
Los Alamos National Laboratory
Los Alamos, New Mexico

Jim Fox
University of Washington
Seattle, Washington

Robert Fry
Smiths Industries
Grand Rapids, Michigan

* **Frank C. Frye**
Computer Composition Corporation
Madison Heights, Michigan

Rick Furuta
University of Maryland
College Park, Maryland

Sam Gassel
University of Chicago
Chicago, Illinois

Thaddeus Gerards
The Open University
Heerlen, Netherlands

Helen M. Gibson
Wellcome Institute for the
History of Medicine
London, England

Regina Girouard
American Mathematical Society
Providence, Rhode Island

† **Jacques Goldberg**
Technion IIT
Haifa, Israel

Max Goldstein
New York University
New York, New York

Frederic Gooding Jr.
Vassar College
Poughkeepsie, New York

Raymond E. Goucher
TEX Users Group
Providence, Rhode Island

John S. Gourlay
ArborText, Inc.
Ann Arbor, Michigan

John R. Green
The Friary
St. Bonaventure, New York

Regina Gregory
NCAR
Boulder, Colorado

* **Gayla Groom**
Blue Sky Research
Portland, Oregon

Nancy K. Groschwitz
Talaris Systems, Inc.
San Diego, California

* **Paul Grosso**
ArborText, Inc.
Ann Arbor, Michigan

* **Dean Guenther**
Washington State University
Pullman, Washington

Jane M. Hahn
Mission Research Corporation
Santa Barbara, California

Richard Hainebach
Electrical Publishing
Management & Service
NA Banholt, Netherlands

Hope Hamilton
NCAR
Boulder, Colorado

Marvin V. Harlow
Los Alamos National Laboratory
Los Alamos, New Mexico

† **Robert Harris**
Micro Programs, Inc.
Syosset, New York

Laura Hawks
Northwest Computer Service
Lakeville, Connecticut

Doug Henderson
University of California, Berkeley
Berkeley, California

* **Amy Hendrickson**
TEXnology, Inc.
Brookline, Massachusetts

David Hitchcock
Honeywell Bull
Los Angeles, California

Mildred H. Hoak
Los Alamos National Laboratory
Los Alamos, New Mexico

Alan Hoenig
City University of New York
Huntington, New York

Ezra Holston
Harcourt Brace Jovanovich
Cambridge, Massachusetts

Anita Z. Hoover
University of Delaware
Newark, Delaware

Don Hosek
Harvey Mudd College
Claremont, California

Nancy M. Hunt
Sandia National Laboratories
Livermore, California

Sam Hunting
Electric Book
Dorchester, Massachusetts

Patrick Ion
Mathematical Reviews
Ann Arbor, Michigan

Calvin W. Jackson
California Institute of Technology
Los Angeles, California

* **Peter Jacobsen**
Micro Publishing Systems, Inc.
Vancouver, British Columbia, Canada

Charles L. James
University of California, San Diego
La Jolla, California

Jeanette M. Jenness
Lawrence Livermore National Laboratory
Livermore, California

Yvonne Johnson
Los Alamos National Laboratory
Los Alamos, New Mexico

Orall Joseph
Library of Congress
Washington, District of Columbia

† **Erik Jul**
OCLC, Inc.
Dublin, Ohio

Helmut Jürgensen
University of Western Ontario
London, Ontario, Canada

William Kaster
Personal TEX, Inc.
Mill Valley, California

Jeffrey Katz
Montréal, Québec, Canada

David Kellerman
Northlake Software
Portland, Oregon

Benjamin J. Kennedy
Xybion Corporation
Cedar Knolls, New Jersey

* **Richard Kinch**
Kinch Computer Company
Ithaca, New York

* **Robert L. Kister**
K-Talk Communications
Columbus, Ohio

† **Kazuhiro Kitagawa**
Keio University
Yokohama, Japan

Carol Klos
Stratus Computer, Inc.
Marlboro, Massachusetts

David H. Kratzer
Los Alamos National Laboratory
Los Alamos, New Mexico

† **Robert L. Kruse**
Saint Mary's University
Halifax, Nova Scotia, Canada

Ryoichi Kurasawa
ASCII Corporation
Minato-ku Tokyo, Japan

John C. Lane
Technical Typesetting, Inc.
Baltimore, Maryland

Steven Lapham
Tribune TV Log
Glen Falls, New York

Daniel Latterner
Mathematical Reviews
Ann Arbor, Michigan

Paul K. Leeper
Texas A & M University
College Station, Texas

Pierre MacKay
University of Washington
Seattle, Washington

† **Laurie Mann**
Stratus Computer, Inc.
Marlboro, Massachusetts

Don Marlette
American Institute of Physics
Woodbury, New York

Marian V. Martinez
Los Alamos National Laboratory
Los Alamos, New Mexico

Philip Martinicchio
TV Guide
Radnor, Pennsylvania

† **Mary McCaskill**
NASA Langley Research Center
Hampton, Virginia

Robert W. McGaffey
Martin Marietta Energy Systems, Inc.
Oak Ridge, Tennessee

Claudia McNellis
Library of Congress
Washington, District of Columbia

† **James D. Mooney**
West Virginia University
Morgantown, West Virginia

Mary-Jean Moore
California Computer Resources, Inc.
Oakland, California

† **Paul M. Muller**
Norman Paul Consultants
Palo Alto, California

Norman W. Naugle
Texas A & M University
College Station, Texas

† **David Ness**
TV Guide
Radnor, Pennsylvania

Trang Doan Nguyen
Fermi National Accelerator Laboratory
Batavia, Illinois

Daniel D. Olson
ETP Services Co.
Portland, Oregon

† **Berkeley Parks**
University of Washington
Seattle, Washington

David W. Parmenter
Digital Equipment Corporation
Nashua, New Hampshire

Clement Pellerin
McGill University
Montréal, Québec, Canada

Richard Perline
CitiCorp
New York, New York

David Peterson
Massachusetts Institute of Technology
Cambridge, Massachusetts

Noel C. Peterson
Library of Congress
Washington, District of Columbia

Craig R. Platt
University of Manitoba
Winnipeg, Manitoba, Canada

Wayne Podaima
National Research Council
Ottawa, Ontario, Canada

† **Jean J. Pollari**
Rockwell International
Cedar Rapids, Iowa

† **Lynne Price**
Hewlett-Packard Company
Palo Alto, California

Jon Thomas Radel
Stanford Telecommunications, Inc.
Leesburg, Virginia

Pat Rau
Northlake Software
Portland, Oregon

Thomas J. Reid
Texas A & M University
College Station, Texas

† **James T. Renfrow**
Jet Propulsion Laboratory
Pasadena, California

Nicola Richards
McGill University
Montréal, Québec, Canada

Don Riley
Sandia National Laboratories
Livermore, California

* **David Rodgers**
ArborText, Inc.
Ann Arbor, Michigan

Eugene S. Rodolphe
New York University
New York, New York

David F. Rogers
United States Naval Academy
Annapolis, Maryland

† **Kauko Saarinen**
University of Jyväskylä
Jyväskylä, Finland

Shashi Sathaye
University of Kentucky
Lexington, Kentucky

Brian T. Schellenberger
SAS Institute
Cary, North Carolina

† **Michael Schmidt**
Honeywell Bull
Iberville, Québec, Canada

Herbert Earl Schulz
College of DuPage
Naperville, Illinois

Larry Sharlow
Flagstaff, Arizona

Linda Sirney
NCAR
Boulder, Colorado

James Slagle
TV Guide
Radnor, Pennsylvania

* **Barry Smith**
Blue Sky Research
Portland, Oregon

Laurel Stegle
American Institute of Physics
Woodbury, New York

David K. Steiner
Rutgers University
Piscataway, New Jersey

Jan Stewart
NCAR
Boulder, Colorado

Thomas E. Strack
Technical Association of the Pulp and Paper Industry
Atlanta, Georgia

Peggy Sutherland
American Physical Society
Woodbury, New York

Steve Sydoriak
Los Alamos National Laboratory
Los Alamos, New Mexico

Cheryl Taub
American Institute of Physics
Woodbury, New York

William J. Taylor
Technical Typesetting, Inc.
Baltimore, Maryland

Christina Thiele
Carleton University
Ottawa, Ontario, Canada

Margaret Thomas
Talaris Systems, Inc.
San Diego, California

David W. Thompson
Lawrence Livermore National Laboratory
Livermore, California

Jeanne Torres
NCAR
Boulder, Colorado

Frederick M. Trietsch
TV Guide
Radnor, Pennsylvania

Laurent Valosek
Personal TEX, Inc.
Mill Valley, California

Kent Wada
University of British Columbia
Vancouver, British Columbia, Canada

* **Don Wagner**
Micro Publishing Systems, Inc.
Vancouver, British Columbia, Canada

† **Alex Warman**
TEXworks Pty. Ltd.
Melbourne, Australia

Stacy Waters
University of Washington
Seattle, Washington

Michael Weinstein
Random House, Inc.
Cambridge, Massachusetts

Samuel B. Whidden
American Mathematical Society
Providence, Rhode Island

Kendall Whitehouse
University of Pennsylvania
Philadelphia, Pennsylvania

Janene Winter
American Mathematical Society
Providence, Rhode Island

† **Alan Wittbecker**
TEX Users Group
Providence, Rhode Island

William B. Woolf
Mathematical Reviews
Ann Arbor, Michigan

Thomas H. Wright
Clarkson University
Potsdam, New York

† **Ken Yap**
University of Rochester
Rochester, New York

Deborah Young
Technical Association of the Pulp and Paper Industry
Atlanta, Georgia

Ralph Youngen
American Mathematical Society
Providence, Rhode Island

Andrew Yull
Addison-Wesley Publishing Ltd.
Don Mills, Ontario, Canada

Mona Zeftel
Addison-Wesley Publishing Company
Reading, Massachusetts

List of Exhibitors

ArborText Inc.
535 West William Street
Suite 300
Ann Arbor, Michigan
48103 USA
313/996-3566

Blue Sky Research
534 SW Third Avenue
Portland, Oregon
97204 USA
800/622-8398
503/222-9571

Computer Composition Corp.
1401 West Girard Ave.
Madison Heights, Michigan
48071 USA
313/545-4330

Kinch Computer Company
501 South Meadow Street
Ithaca, New York
14850 USA
607/273-0222

K-Talk Communications
50 McMillen Avenue
Columbia, Ohio
43201 USA
614/294-3535

Micro Publishing Systems, Inc.
300–1120 Hamilton Street
Vancouver, British Columbia
Canada V6B 2S2
604/687-0354

Personal TEX
12 Madrona Valley
Mill Valley, California
94941 USA
415/388-8853

TEXnology, Inc.
57 Longwood Avenue
Brookline, Massachusetts
02146 USA
617/738-8029

***TEXT1* Distribution**
Computing Service Center
Washington State University
Pullman, Washington
99164-1220 USA
509/335-0411

Other TEX Conference Proceedings

There are now a number of conference proceedings devoted to TEX; two are from Europe, and two from North America. All are available through the TEX Users Group, Providence, Rhode Island.

Proceedings of the First European Conference on TEX for Scientific Documentation. Dario Lucarella, ed. Reading, Mass.: Addison-Wesley. 1985. [16–17 May 1985, Como, Italy.]

Proceedings of the Second European Conference on TEX for Scientific Documentation. Jacques Désarménien, ed. Berlin: Springer-Verlag. 1986. [19–21 June 1986, Strasbourg, France.]

Conference Proceedings, 8th Annual Meeting of the TEX Users Group. Dean Guenther, ed. *TEXniques* Number 5. 1987. [24–26 August 1987, University of Washington, Seattle, Washington.]

Conference Proceedings, 9th Annual Meeting of the TEX Users Group. Christina Thiele, ed. *TEXniques* Number 7. 1988. [22–24 August 1988, McGill University, Montréal, Canada.]